Algorithmic Map Recognition and Edge Detection with Point to Point Pathfinding

Computer Science NEA

Name: Rubens Pirie

Candidate Number: 1749
Centre Number: 58231

Centre Name: Barton Peveril Sixth Form College

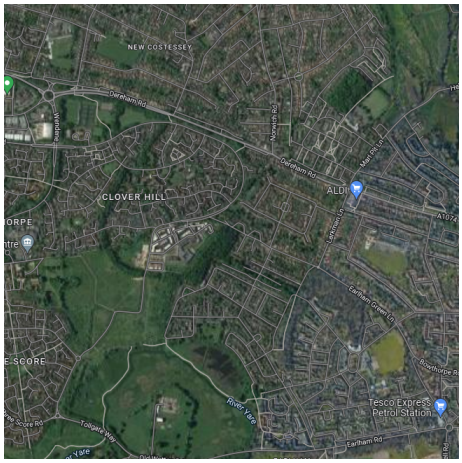
Contents

1 Analysis

1.1 Statement Of Problem

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

(1) Map without labels on roads



(2) Map with labels on roads

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

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

1.2 Background

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

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

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

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

1.3 End User

1.3.1 First Interview

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

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

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

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

   "Yes"

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

   "Yes, **NEEDS TO BE ADDED TO**"

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

   "No. I passed my driving test before we had sat-nav or internet, so clearly they're not vital - we survived without them!

   They are quite helpful though as we used to have to buy a new road map every year, whereas web maps can be updated as things change, instead of only annually!"

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

   "Clarity and simplicity. Nothing needlessly complicated."

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

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

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

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

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

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

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

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

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

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

1.3.2 Evaluation of First Interview

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

1.4 Initial Research

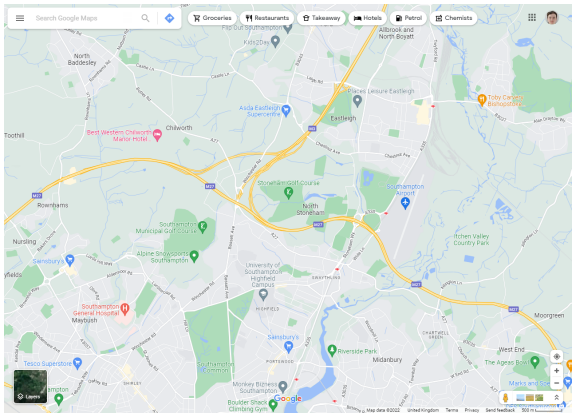
1.4.1 Existing Solutions

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

Google Maps

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

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

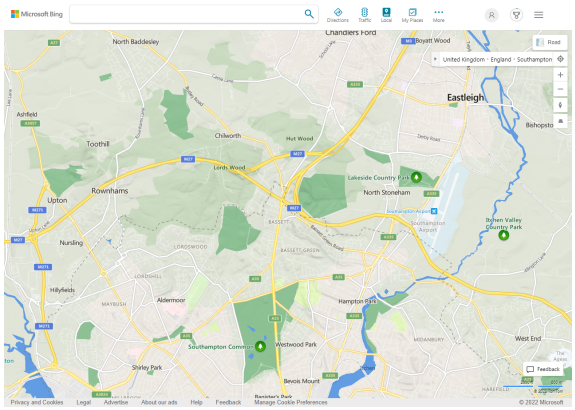


(1) Example of Google Maps' GUI

Sourced from Google Maps©

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

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

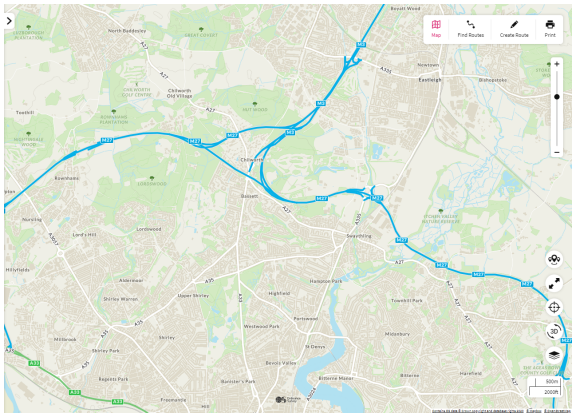


(2) Example of Bing Maps' GUI

Sourced from Bing Maps©

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

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



(3) Example of Ordnance Survey's Map GUI

Sourced from OSMaps.com©

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

1.4.2 Possible Algorithmic Solutions

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

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

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

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

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

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

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

1.4.3 Key Components Required

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

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

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

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

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

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

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

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

Graph Traversal and Output

1.5 Further Research

1.5.1 Dive into Specific Algorithms

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.5.2 Second Interview

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

HAS BEEN ASKED WAITING FOR RESPONSES

1. **Bobbert?**

   bobbert.

2. **Cobbert?**

   cobbert.

3. **Dobbert?**

   dobbert.

4. **Fobbert?**

   Fobbert.

5. **Norbert?**

   norbert

6. **Dilbert?**

   dilbert.

7. **Bobbert?**

   bobbert.

1.5.3 Evaluation of Second Interview

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

Apart from that however I feel the interview went…

1.6 Prototyping

1.6.1 Prototype Objectives

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

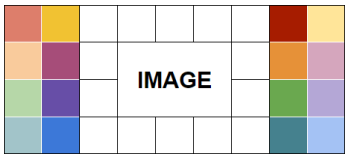
- A version of edge detection
- A graph class with basic traversal
- A forms interface for showing images

1.6.2 Edge Detection

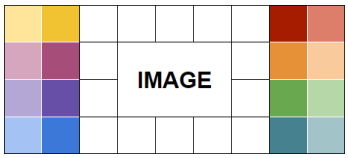
For the example of edge detection which I am going to prototype I have chosen Canny Edge Detection, this is the most common of the types of edge detection and is relatively simple. It is also widely documented which allows me to focus more on the application and less on the finding of resources.

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

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



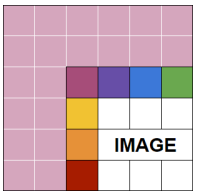
(4) Wrapping



(5) Mirroring

(6) Extension

(7) Constants



(8) Duplication

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

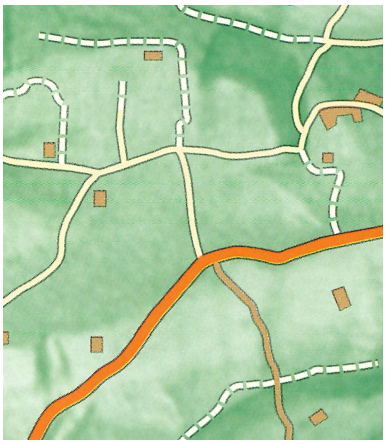


Figure 1: Original Image

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

The implementation in code of this is as below:

```
1  public double[,] BWFilter(Bitmap image)
2  {
3      double[,] result = new double[image.Height, image.Width];
4
5      for (int i = 0; i < image.Height; i++)
6      {
7          for (int j = 0; j < image.Width; j++)
8          {
9              Color c = image.GetPixel(j, i);
10             double value = c.R * 0.299 + c.G * 0.587 + c.B * 0.114;
11
12             result[i, j] = Bound(0, 255, value);
13         }
14     }
15
16     return result;
17 }
```

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

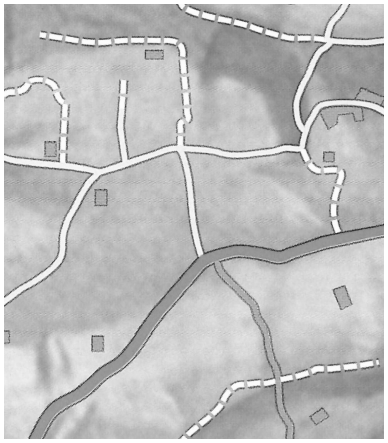


Figure 2: Black and White Filter

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

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

    Matrix gaussianKernel = GetGaussianKernel(kernelSize, sigma);

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

    return result;
}

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

    double sum = 0;

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

```
34              sum += result[halfK + cntY, halfK + cntX];
35              cntX++;
36          }
37          cntY++;
38      }
39
40      for (int i = 0; i < k; i++) for (int j = 0; j < k; j++) result[i, j] /= sum;
41      return new Matrix(result);
42  }
```

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

Figure 3: Gaussian Filter

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

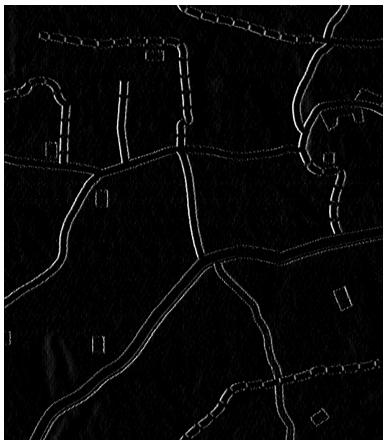
$$M_y = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad M_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

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*.

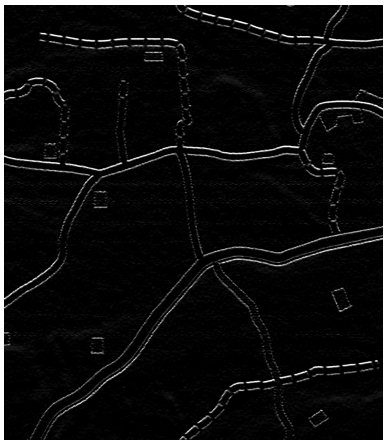
```
1  public double[,] CalculateGradientX(double[,] imageArray)
2  {
3      double[,] result = new double[imageArray.GetLength(0), imageArray.GetLength(1)];
```

```
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
20      return result;
21  }
```

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



(1) Gradient in X



(2) Gradient in Y

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

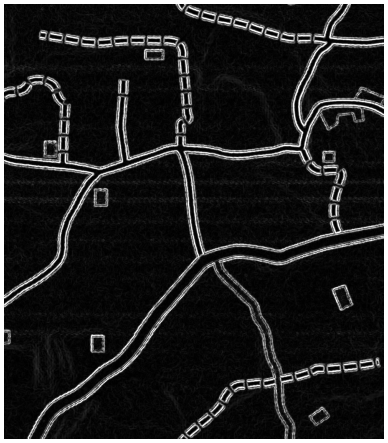


Figure 4: Gaussian Filter

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

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

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

Figure 5: Gaussian Filter

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

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

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

Figure 6: Gradient Direction

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

```
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] < magnitudeKernel[1, 0])
15                  {
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:

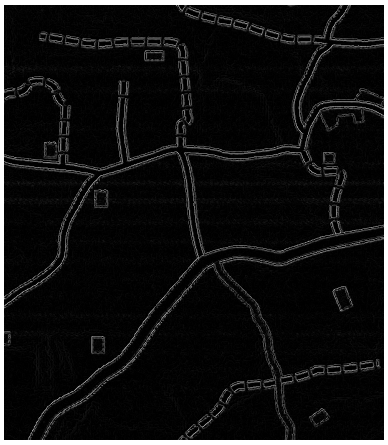


Figure 7: Magnitude Threshold

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 he other steps of the edge detection we are not interested yet at the pixels which are surrounding the ones we are looking at. We are just interested in its specific value. The code to perform this is as follows.

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

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

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

    return result;
}
```

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

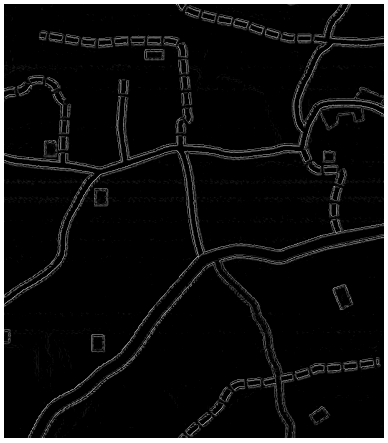


Figure 8: Magnitude Threshold

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

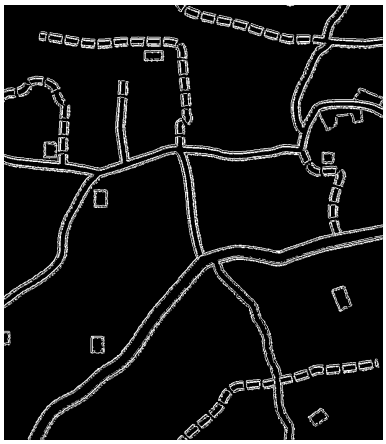
7. Edge tracking by Hysteresis

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

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

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

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

    return result;
}
```

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

(1) Original Input Image



(2) Image after Classic Edge Detection

As is visible in the final image we can see that after the edge detection there are holes in the

lines. As well as this there are occasional gaps this is where I came up with a extra couple of steps. This allows the image to be properly formed and connect any miscellaneous roads which have small gaps.

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

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

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

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

    Matrix embossMatrix = new Matrix(new double[,]
    {
        { -2, -1, 0 },
        { -1, 1, 1 },
        { 0, 1, 2 },
    });

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

    return result;
}
```

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

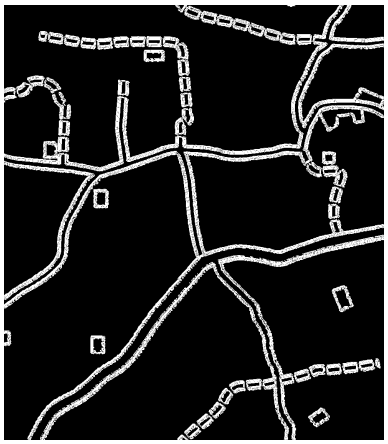


Figure 9: Magnitude Threshold

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

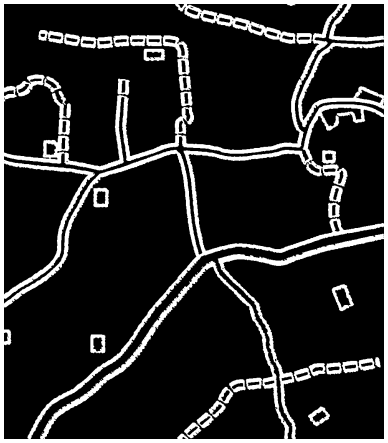


Figure 10: Magnitude Threshold

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

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

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

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

    return result;
}
```

1.6.3 Graph Class and Graph Traversal

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

1. Adjacency Matrix
    - Advantages
        Very fast when needing to lookup connections.

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

- Disadvantages

  Very memory inefficient and will need to grow exponentially in each dimension with the amount of pixels in the image.
  When you have a sparse graph it is even more inefficient.

2. Adjacency List

   - Advantages

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

   - Disadvantages

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

.

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

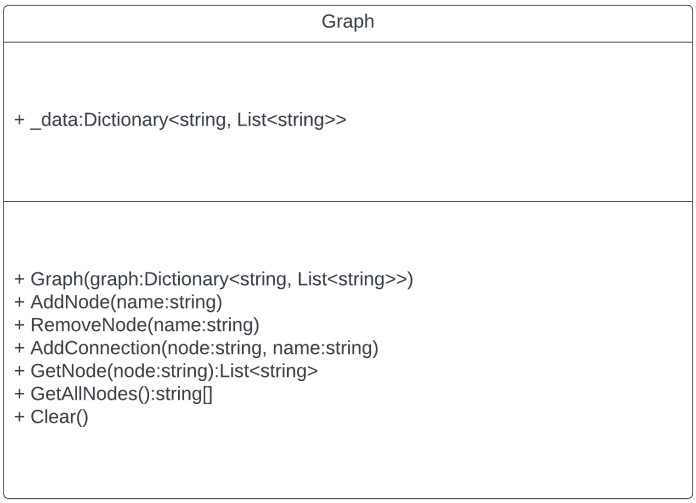


Figure 11: Graph UML Diagram

and in code

```
1  public class Graph
2  {
3      public Dictionary<string, List<string>> _data = new Dictionary<string, List<string>>();
4
5      public Graph(Dictionary<string, List<string>> graph)
6      {
7          _data = graph;
8      }
9
10     public void AddNode(string name)
```

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

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

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

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

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

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

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

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

    // Kick Start
    stack.Push(start);

    while (!stack.IsEmpty())
    {

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

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

        connections.Reverse();

        foreach (string s in connections)
        {
            if (visited[s] == false)
            {
```

```
                stack.Push(s);
            }
        }
    }


    return path.ToArray();
}

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

    // Kick Start
    stack.Enqueue(start);

    while (!stack.IsEmpty())
    {

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

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

        connections.Reverse();

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

    return path.ToArray();
}
```

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

1.6.4 Windows Forms with Images

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

Example

```
partial class ShowImage
{
    /// <summary>
    /// Required designer variable.
```

```
5       /// </summary>
6       private System.ComponentModel.IContainer components = null;
7
8       /// <summary>
9       /// Clean up any resources being used.
10      /// </summary>
11      /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
12      protected override void Dispose(bool disposing)
13      {
14          if (disposing && (components != null))
15          {
16              components.Dispose();
17          }
18          base.Dispose(disposing);
19      }
20
21      #region Windows Form Designer generated code
22
23      /// <summary>
24      /// Required method for Designer support - do not modify
25      /// the contents of this method with the code editor.
26      /// </summary>
27      private void InitializeComponent()
28      {
29          this.imageBox = new System.Windows.Forms.PictureBox();
30          this.next = new System.Windows.Forms.Button();
31          this.content = new System.Windows.Forms.RichTextBox();
32          ((System.ComponentModel.ISupportInitialize)(this.imageBox)).BeginInit();
33          this.SuspendLayout();
34          //
35          // imageBox
36          //
37          this.imageBox.Location = new System.Drawing.Point(12, 12);
38          this.imageBox.Name = "imageBox";
39          this.imageBox.Size = new System.Drawing.Size(500, 450);
40          this.imageBox.TabIndex = 1;
41          this.imageBox.TabStop = false;
42          //
43          // next
44          //
45          this.next.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 24.75F, System.Drawing.FontStyle.Bold,
    ↪ System.Drawing.GraphicsUnit.Point, ((byte)(0)));
46          this.next.Location = new System.Drawing.Point(518, 384);
47          this.next.Name = "next";
48          this.next.Size = new System.Drawing.Size(354, 78);
49          this.next.TabIndex = 4;
50          this.next.Text = "Continue";
51          this.next.UseVisualStyleBackColor = true;
52          this.next.Click += new System.EventHandler(this.next_Click);
53          //
54          // content
55          //
56          this.content.AcceptsTab = true;
57          this.content.Font = new System.Drawing.Font("JetBrains Mono SemiBold", 15F, System.Drawing.FontStyle.Bold);
58          this.content.Location = new System.Drawing.Point(518, 12);
59          this.content.Name = "content";
60          this.content.ReadOnly = true;
61          this.content.Size = new System.Drawing.Size(354, 366);
62          this.content.TabIndex = 5;
63          this.content.Text = "";
```

```
64          //
65          // ShowImage
66          //
67          this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
68          this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
69          this.ClientSize = new System.Drawing.Size(884, 474);
70          this.Controls.Add(this.content);
71          this.Controls.Add(this.next);
72          this.Controls.Add(this.imageBox);
73          this.Name = "ShowImage";
74          this.Text = "ShowImage";
75          this.Load += new System.EventHandler(this.ShowImage_Load);
76          ((System.ComponentModel.ISupportInitialize)(this.imageBox)).EndInit();
77          this.ResumeLayout(false);
78
79      }
80
81      #endregion
82
83      private System.Windows.Forms.PictureBox imageBox;
84      private System.Windows.Forms.Button next;
85      private System.Windows.Forms.RichTextBox content;
86  }
87
88  public partial class ShowImage : Form
89  {
90      private Bitmap _image;
91      private string _content;
92
93      public ShowImage(Bitmap image, string content)
94      {
95          this.ControlBox = false;
96
97          _image = image;
98          _content = content;
99
100         InitializeComponent();
101     }
102
103     private void ShowImage_Load(object sender, EventArgs e)
104     {
105         imageBox.SizeMode = PictureBoxSizeMode.StretchImage;
106         imageBox.Image = _image;
107         content.Text = _content;
108     }
109
110     private void next_Click(object sender, EventArgs e)
111     {
112         Close();
113     }
114 }
```

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 be have its description changed
   - 7.7 The saved binary file should be able to be able to be deleted
8. The program should have re-callable settings
   - 8.1 Map Algorithm
   - 8.2 Random Save Names
   - 8.3 Map Approximations
9. The program settings should be easily movable.
10. The program save files should be easily movable.

1.8 Modelling

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:

2.1.1 Linq

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

2.1.2 Bitmap

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

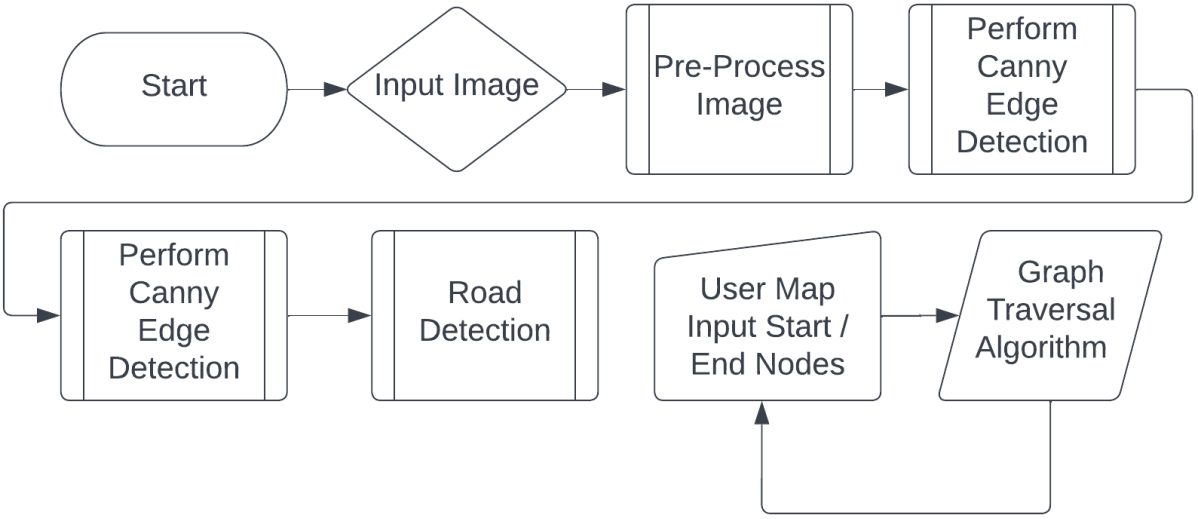
2.1.3 Windows Forms

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

2.2 High Level Overview

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

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



(l) High Level Overview Of Program

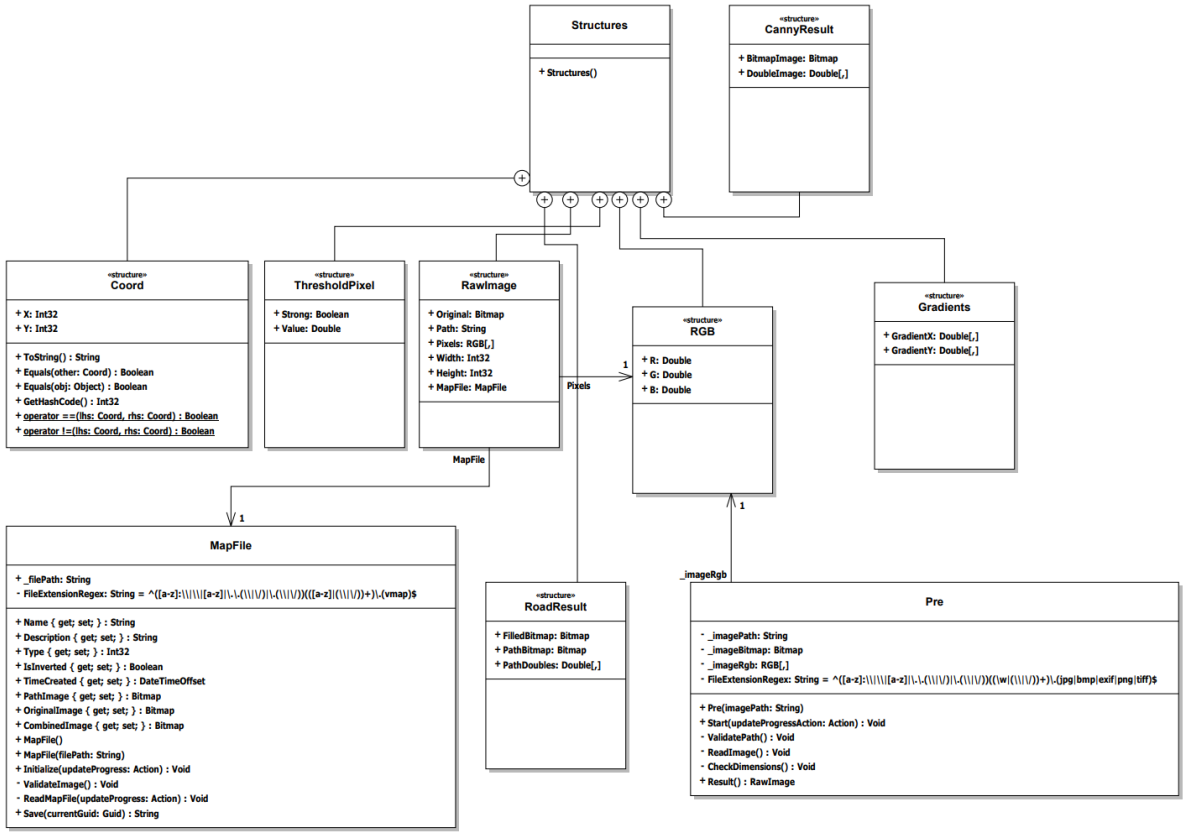
The version of Edge Detection I will be using as previously stated will be Canny Edge Detection, this is as opposed to Sobel Edge Detection. The main version of filling I will be using is flood fill due to it simple nature to implement ad due to the fact that it does not take much memory and can be made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

2.2.1 Backend Library

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

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

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



(2) Overview of Backend Structures

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

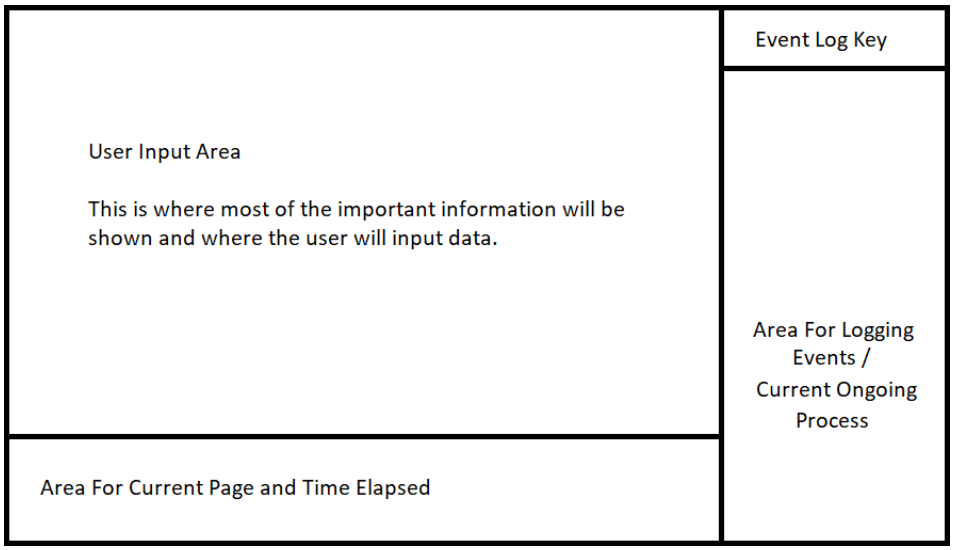
1. **Modularity** - By separating the backend from the frontend one is able to be built without the other. This means that when working on my project I can take time to perfect one without impacting the other.
2. **Reusability** - As previously stated being able to be reused is a large reason as to why to separating the elements is a good idea. Since if I wanted to expand this project for example and make a web interface for it, I could take the maths of the backend and recreate the front end in a web framework like Razor Pages.
3. **Maintainability** - It is allot easier to maintain code when it has been organised into classes and by extension into libraries where a library is a collection of classes. It means that should something throw an error in the backend I would be able to easily isolate the issue and be able to fix it.
4. **Testability** - In a similar vain to the Maintainability of the program being modular also means that it is very easy to implement testing. This means that as I go thorough making my program it will make it allot easier to separate variables and make isolated testing conditions. Furthermore it means that I can test the maths of the Canny Detection without having to worry about making an interface to it using the UI.

2.2.2 Local Application

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

Design of User Interface (Console)

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

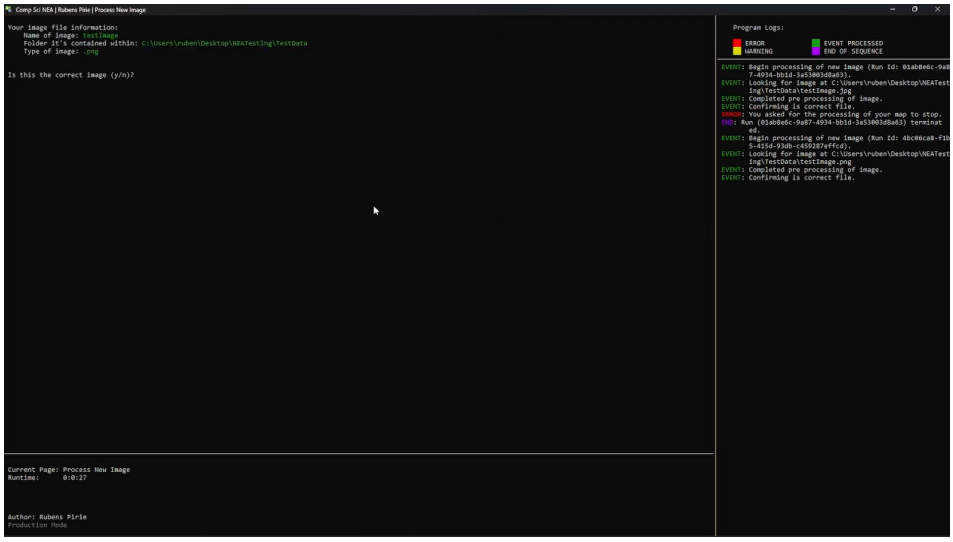


(3) Mock-up of Console Interface

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

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

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

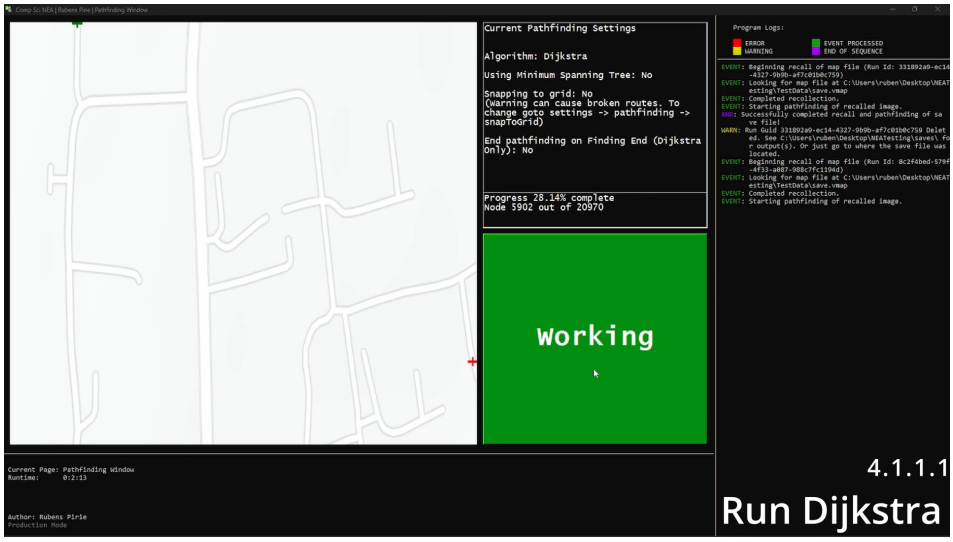


(4) Mock-up of Console Interface

Design of User Interface (Windows Forms)

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

(5) Windows Form For Confirming Image

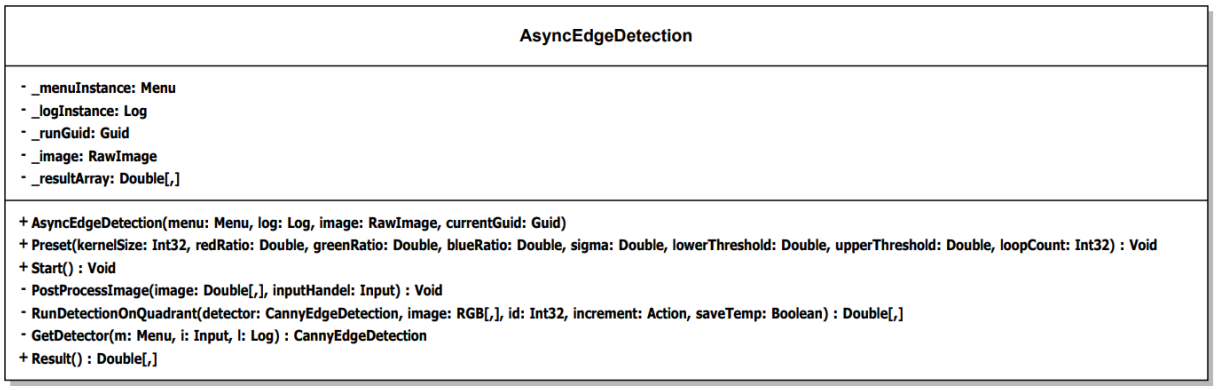


(6) Windows Form For Pathfinding Image

2.3 Class Overviews

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

Async Edge Detection *(Class)*



(7) Async Edge Detection Class Diagram

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

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

starting the edge detection. The Result method will, as the name suggests return the result of the edge detection.

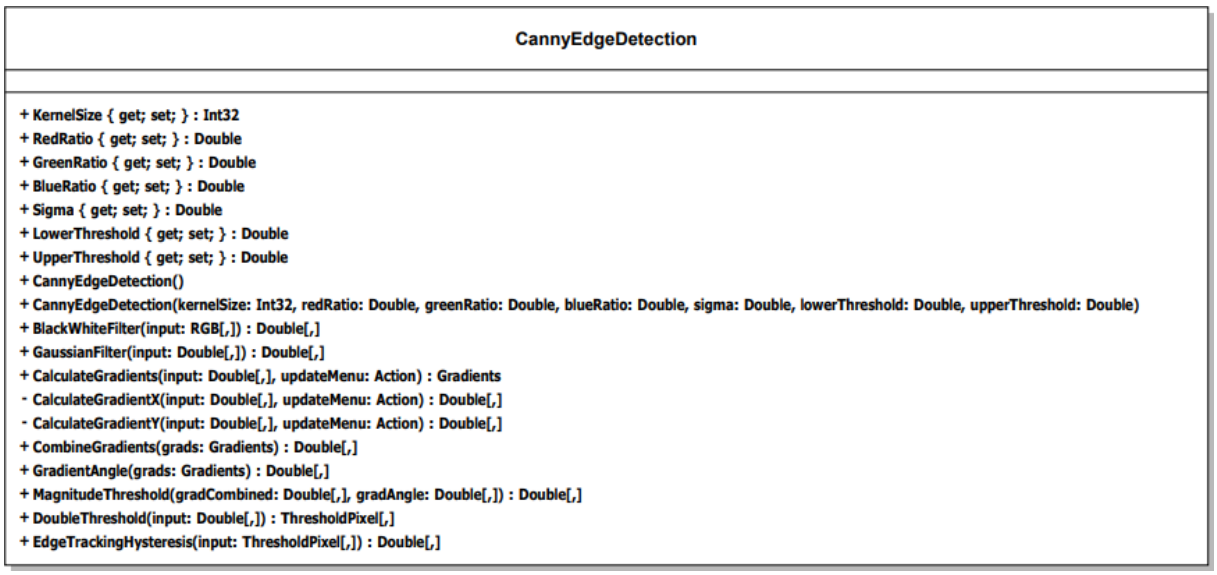
The PostProcessImage method in the class is responsible for the custom embossing which runs through the result of the canny edge detection and fills in any gaps that may have appeared. It also applies an embossing kernel on the image to embolden the lines. It will also prompt the user to enter the amount of times they want to run the embossing process.

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

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

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

Canny Edge Detection *(Class)*



(8) Canny Edge Detection Class Diagram

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

The first set of methods shown in the class diagram are in essence properties on the class which are used in the various calculations. The reason that I will go with getter and setter variables is that if at some point in the future I wish to restrict access to the properties

on the class or perhaps mutate the way in which they are stored this could be easily done without the need to change many different things.

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

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

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

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

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

It is stated that the larger the kernel is the less effect noise will have on the result however it will impact the performance of the detector, for this reason the Canny Edge Detection class defaults to a kernel size of 5x5.

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

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

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

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

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

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

The next stage in canny edge detection is gradient magnitude threshold's, this is a method of removing lines which would not be "thick enough" to be a propped edge. This is accomplished by the use of the MagnitudeThreshold method and the bullet point description:

At every pixel N in a given image, it will be suppressed (removed) if,

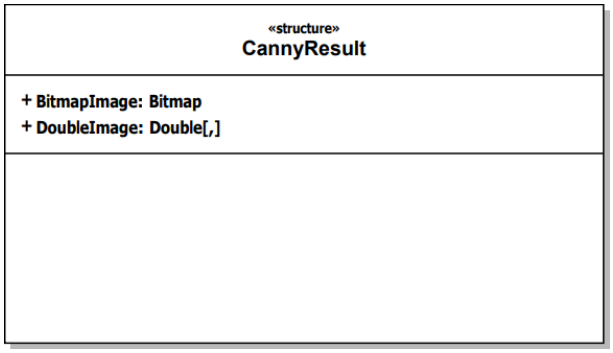
- the rounded gradient angle is 0° (i.e. the edge is in the north-south direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the east and west directions.
- the rounded gradient angle is 90° (i.e. the edge is in the east-west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

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

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

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

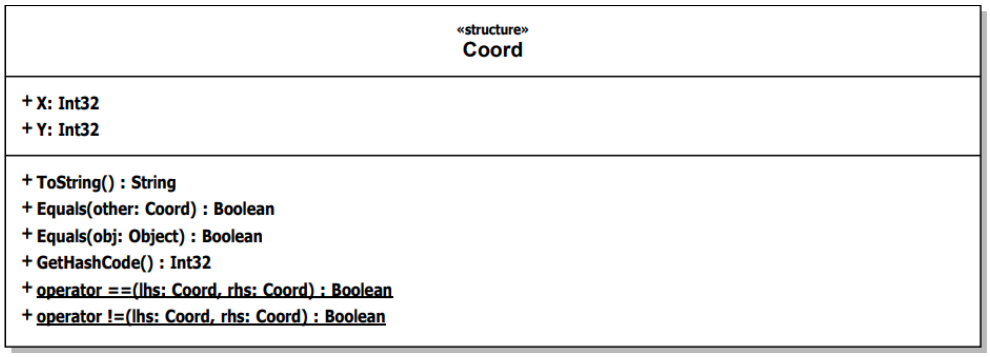
Canny Result *(Structure)*



(9) Canny Result Class Diagram

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

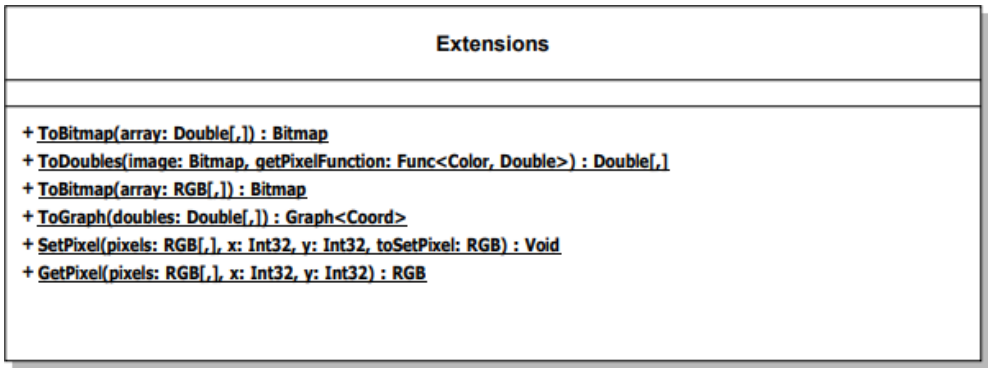
Coord *(Structure)*



(10) Coord Class Diagram

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

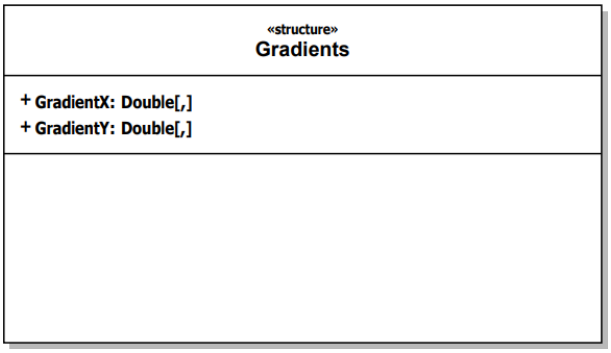
Extensions *(Class)*



(11) Extensions Class Diagram

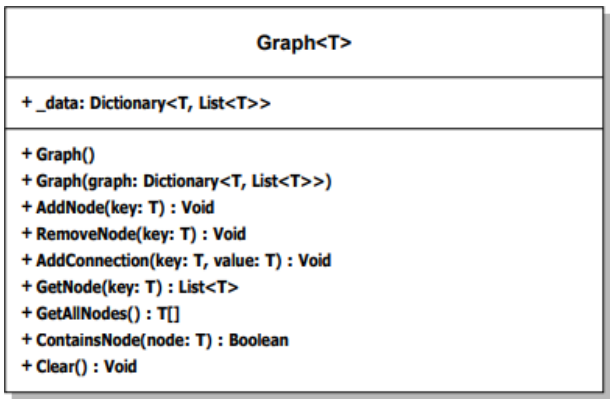
This class is responsible for altering the default behavior of

Gradients *(Structure)*



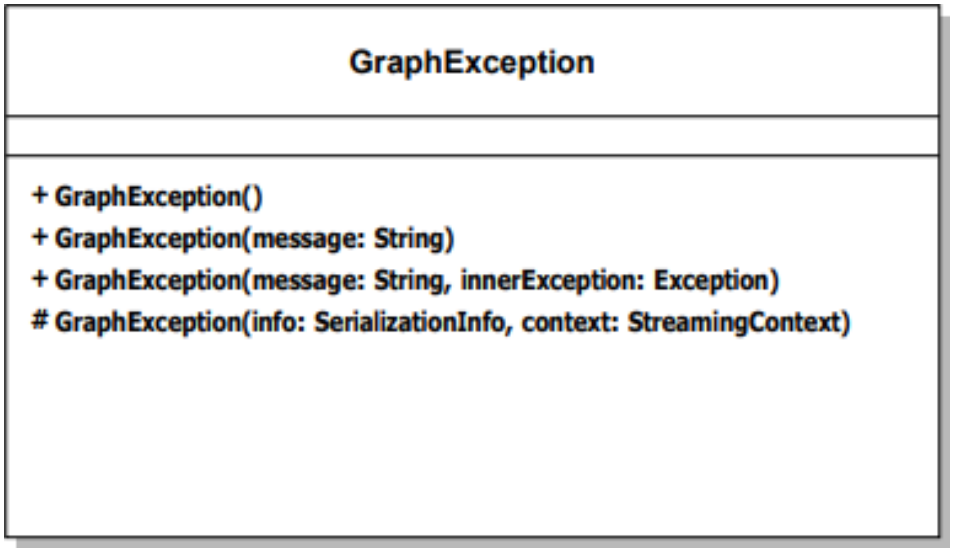
(12) Gradients Class Diagram

Graph *(Class)*



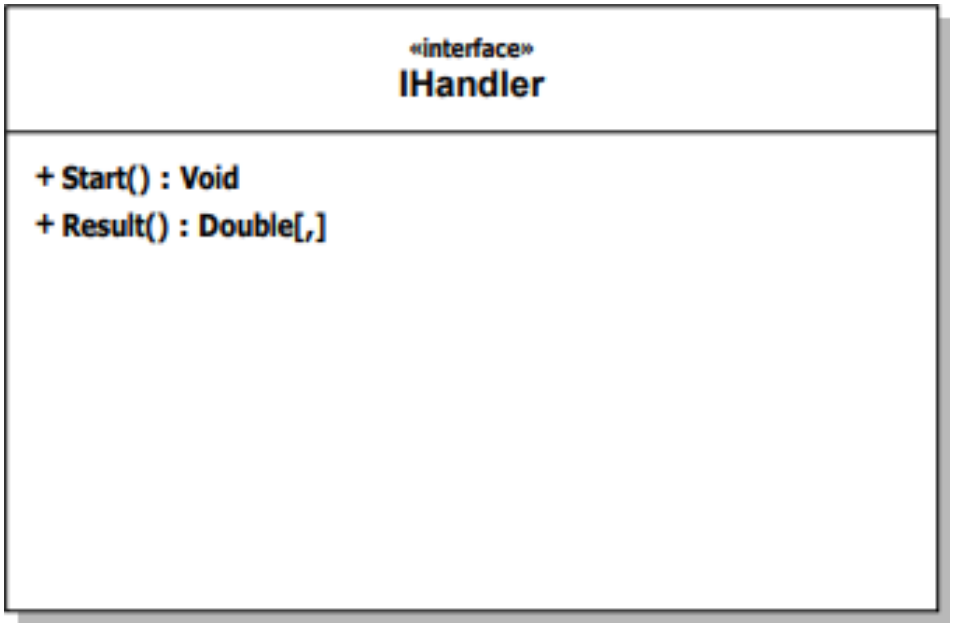
(13) Graph Class Diagram

Graph Exception *(Exception)*



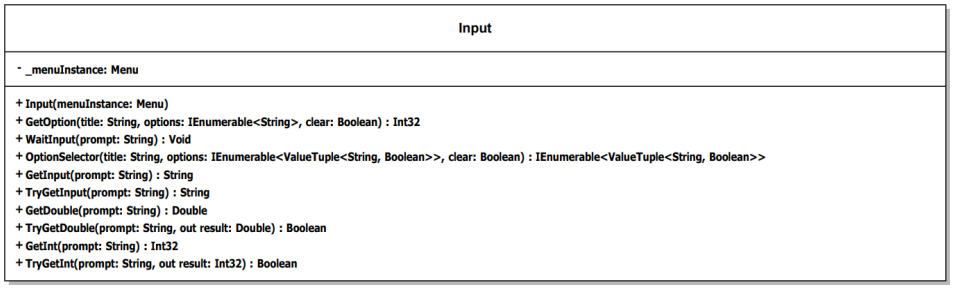
(14) Graph Exception Class Diagram

IHandler *(Interface)*



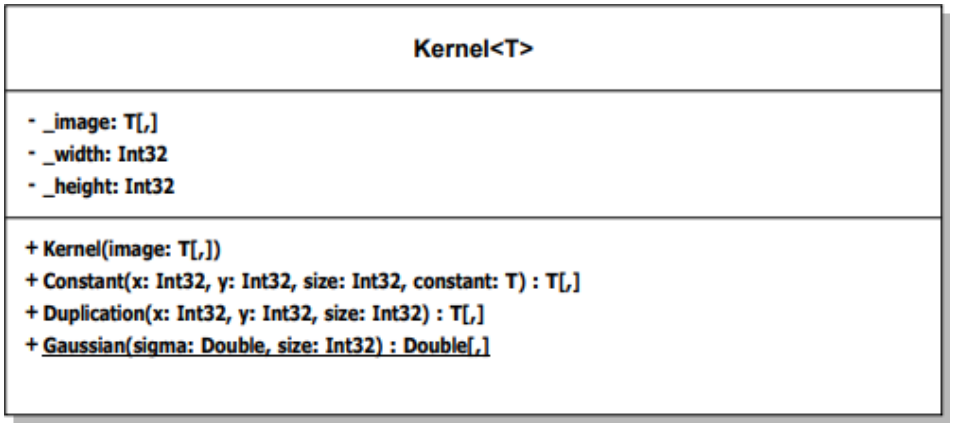
(15) IHandler UML Diagram

Input *(Class)*



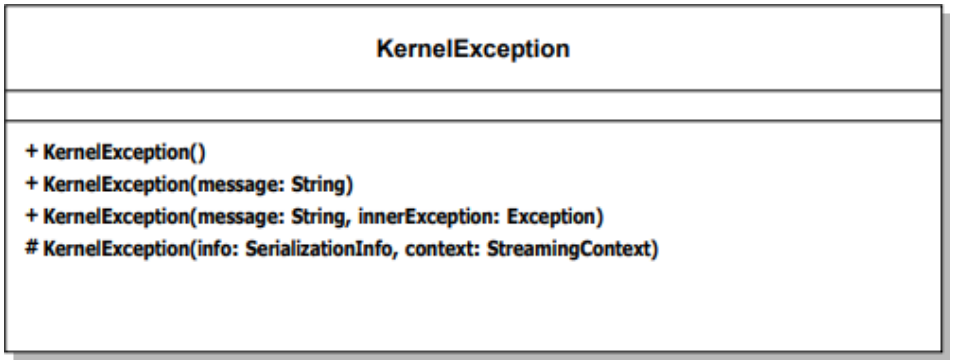
(16) Input Class Diagram

Kernel *(Class)*



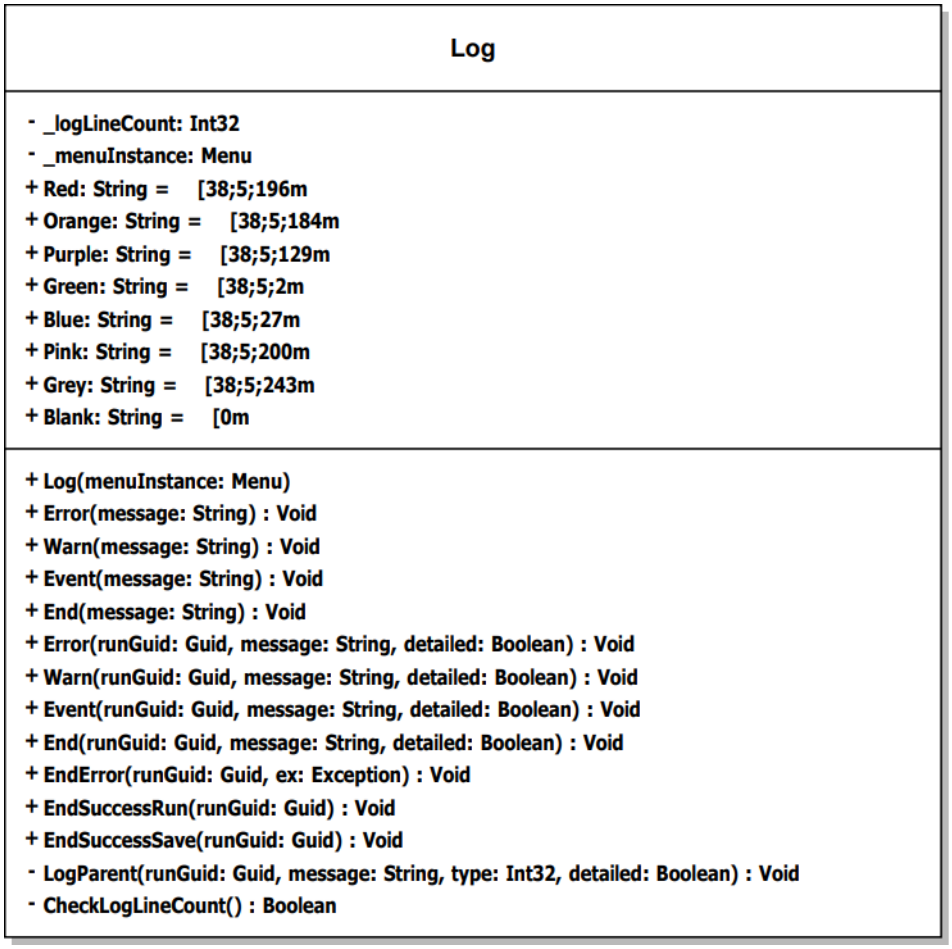
(17) Kernel Class Diagram

Kernel Exception *(Exception)*



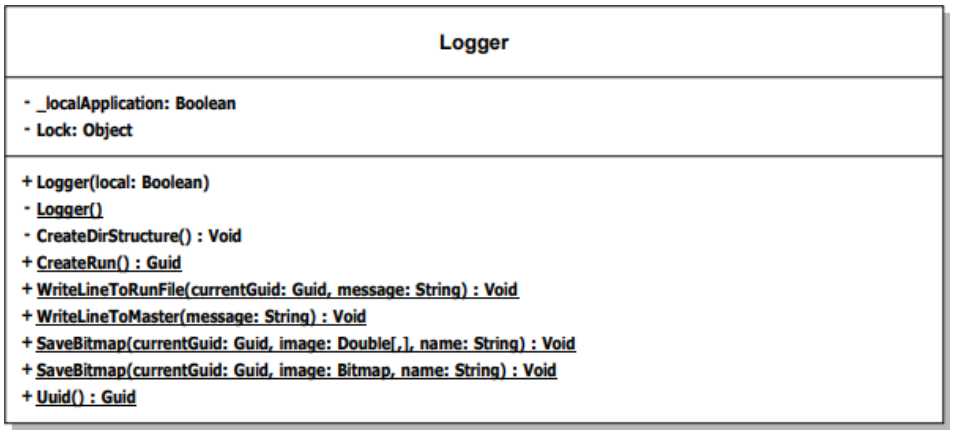
(18) Kernel Exception Class Diagram

Log *(Class)*



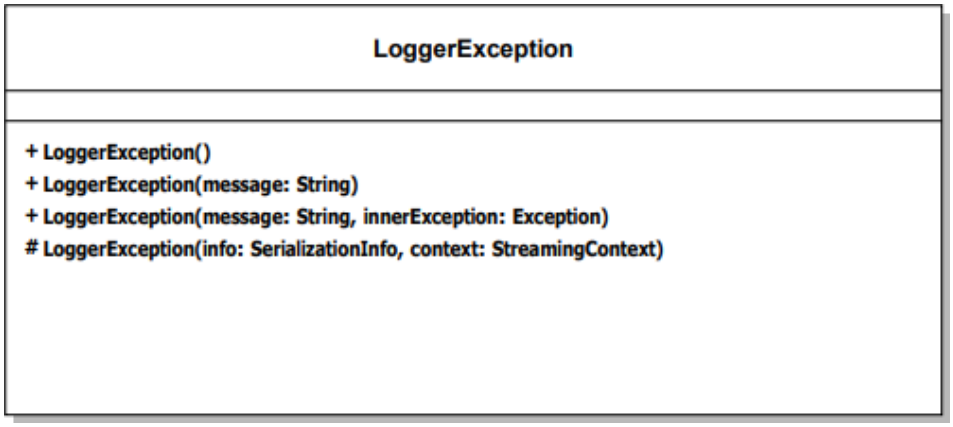
(19) Log Class Diagram

Logger *(Class)*



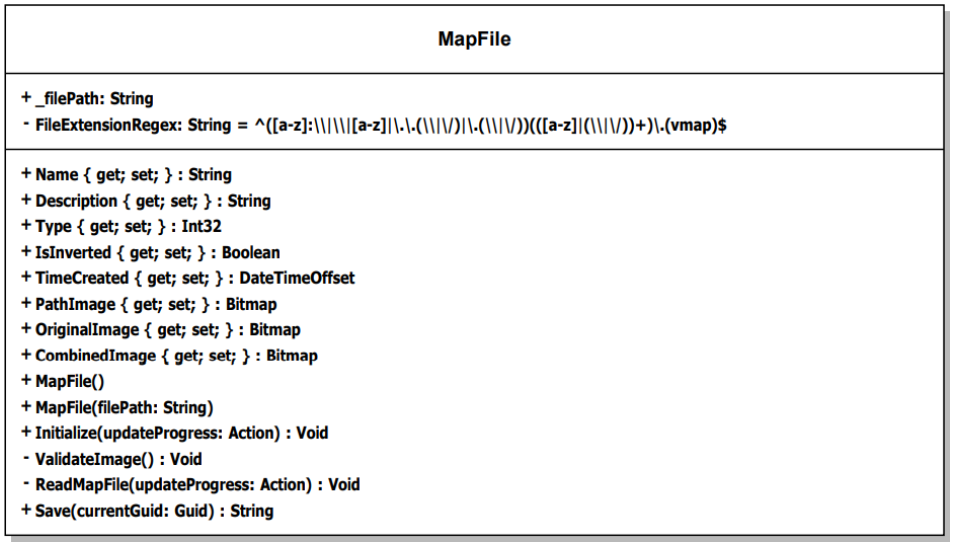
(20) Logger Class Diagram

Logger Exception *(Exception)*



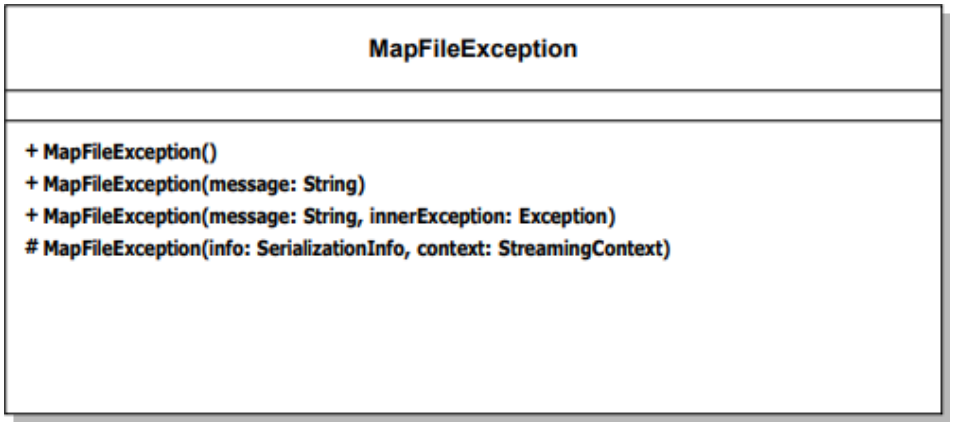
(21) Logger Exception Class Diagram

Map File *(Class)*



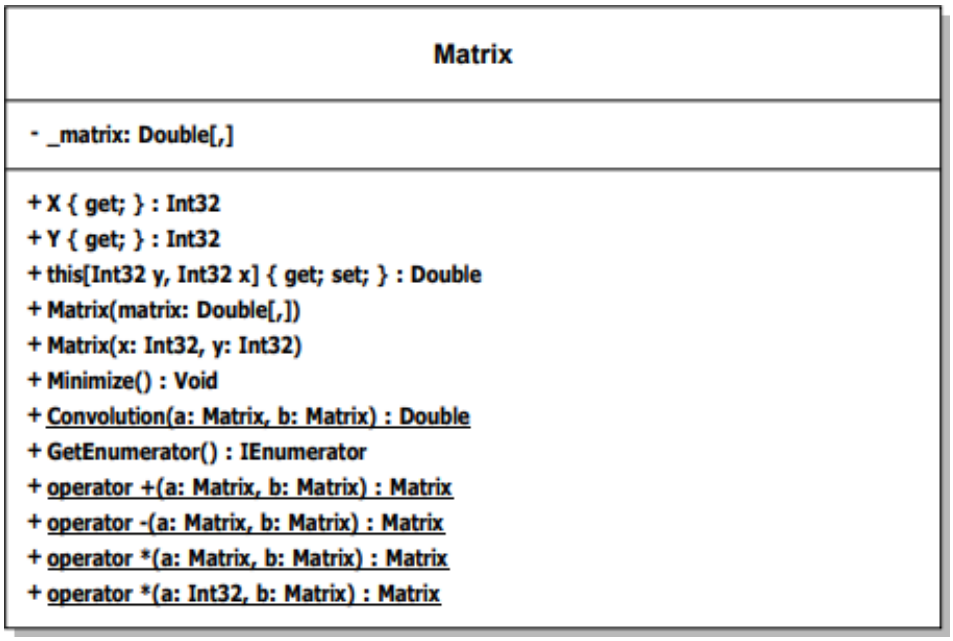
(22) Map File Class Diagram

Map File Exception *(Exception)*



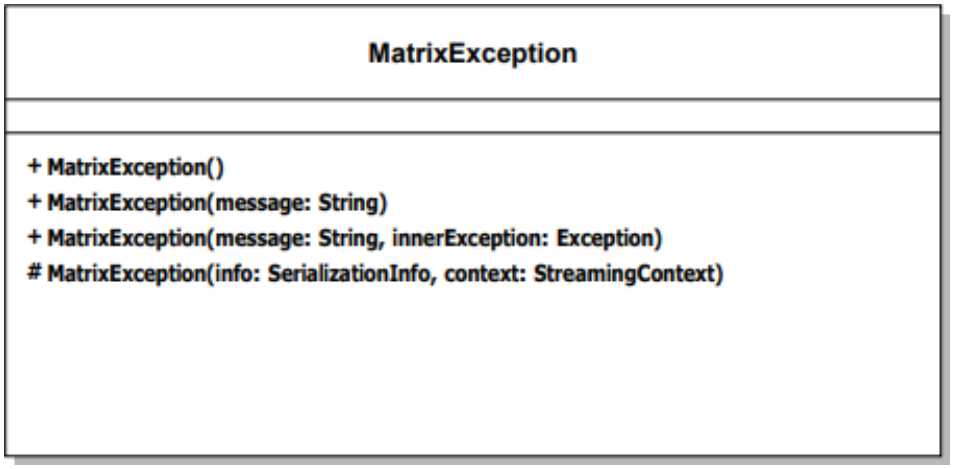
(23) Map File Exception Class Diagram

Matrix *(Class)*



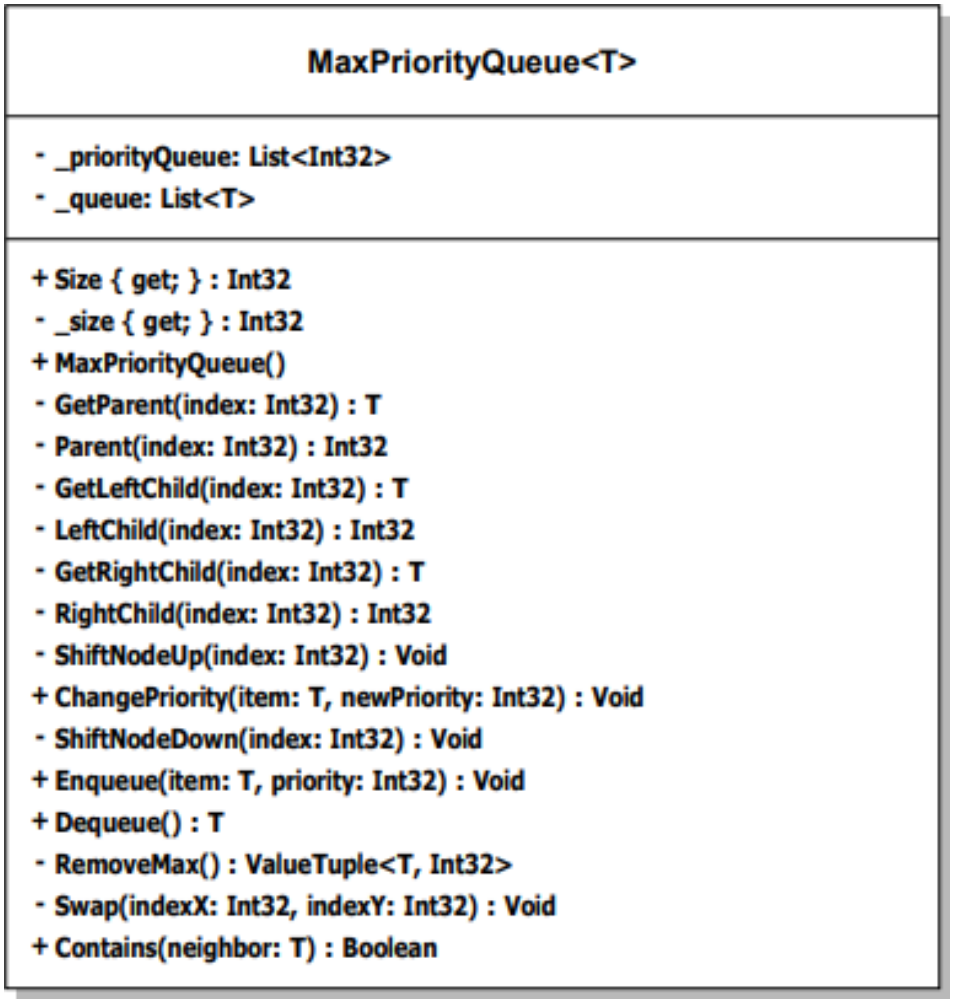
(24) Matrix Class Diagram

Matrix Exception *(Exception)*



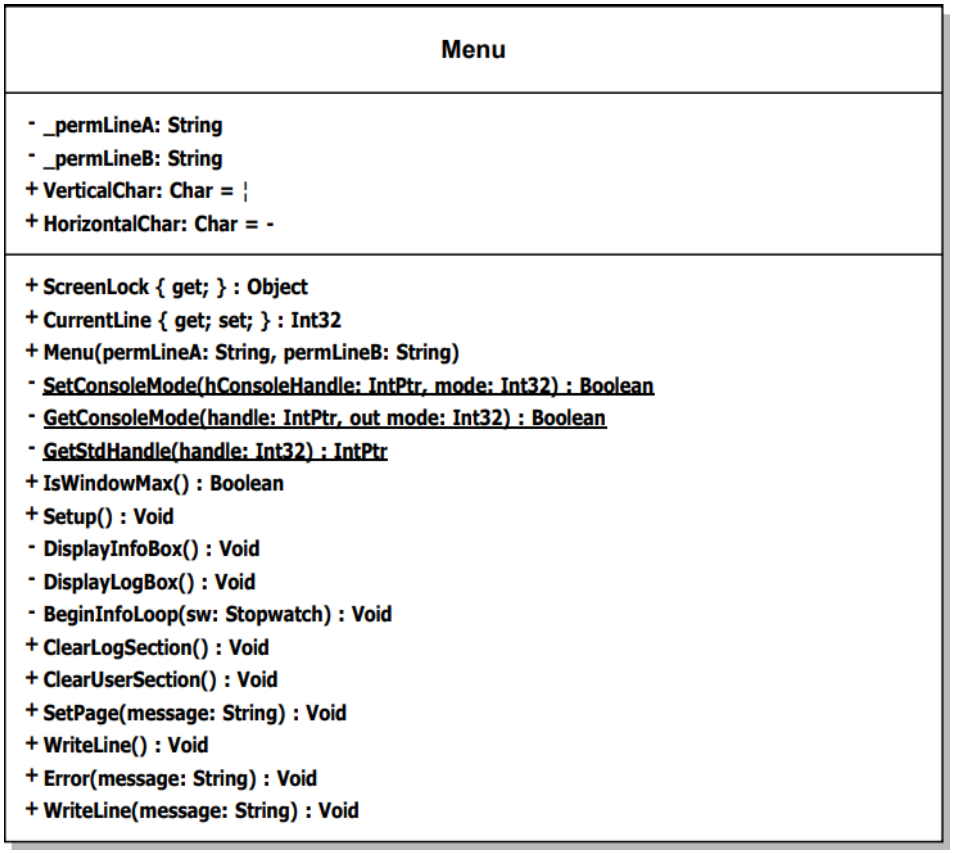
(25) Matrix Exception Class Diagram

Max Priority Queue *(Class)*



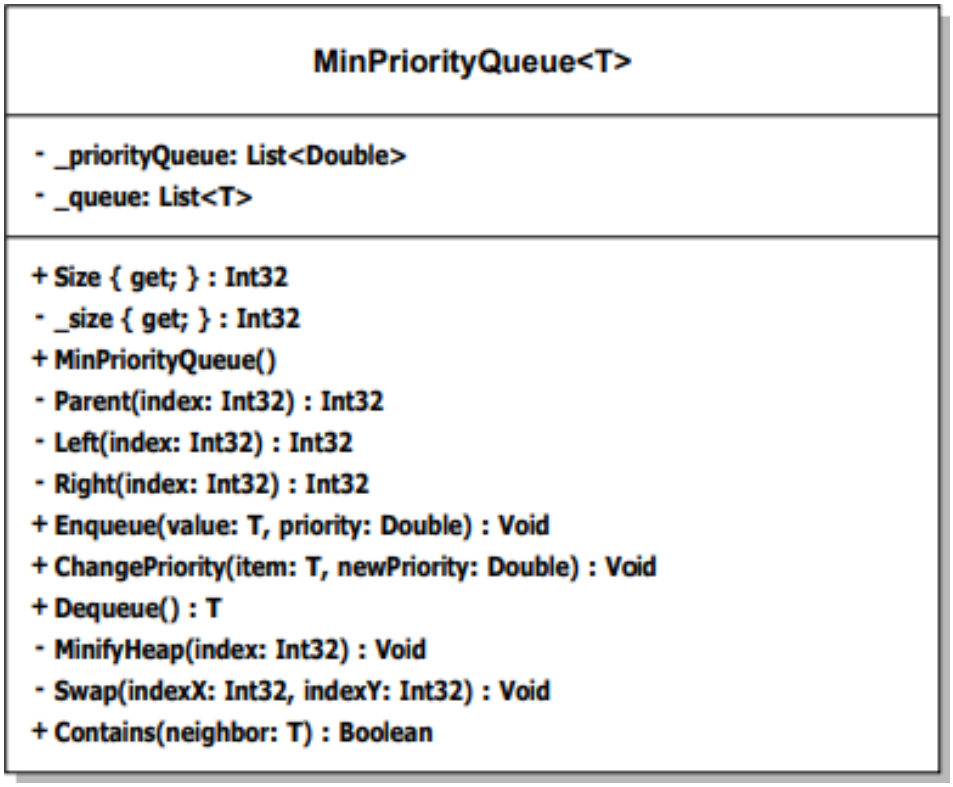
(26) Max Priority Queue Class Diagram

Menu *(Class)*



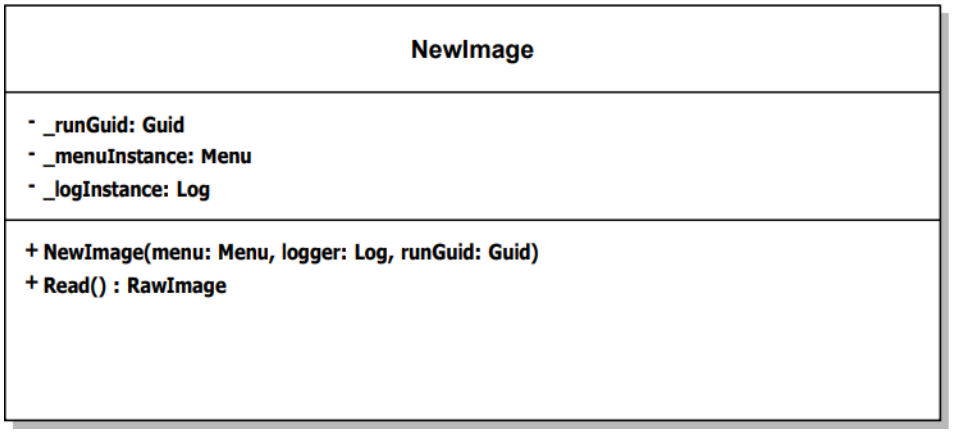
(27) Menu Class Diagram

Min Priority Queue *(Class)*



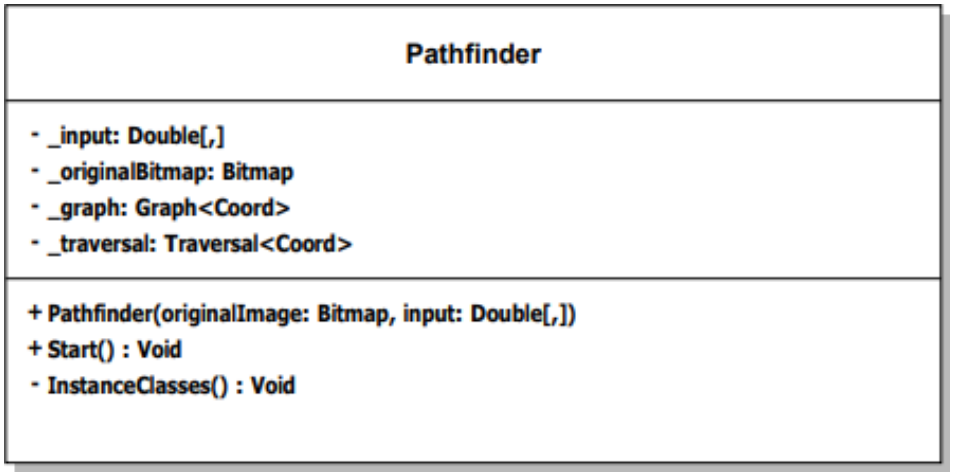
(28) Min Priority Queue Class Diagram

New Image *(Class)*



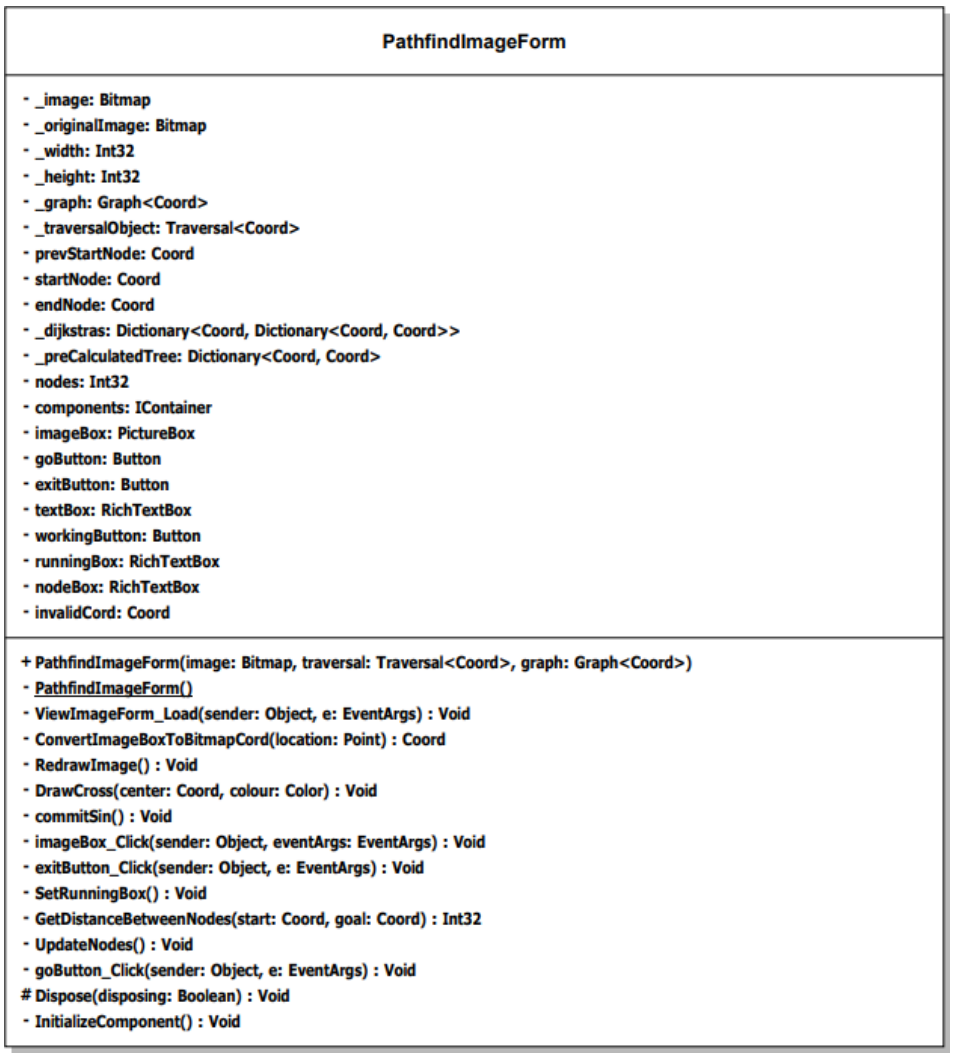
(29) New Image Class Diagram

Pathfinder *(Class)*



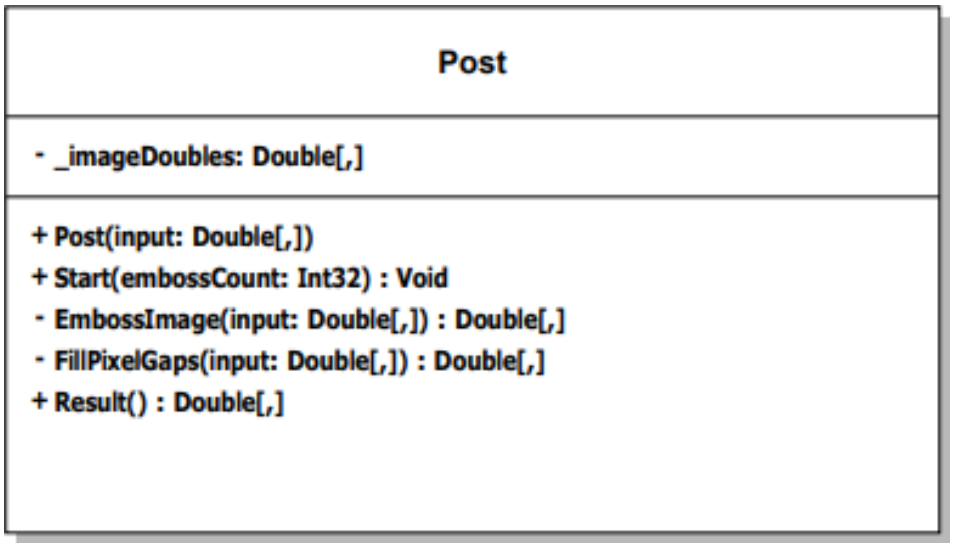
(30) Pathfinder Class Diagram

Pathfind Image Form *(Windows Form)*



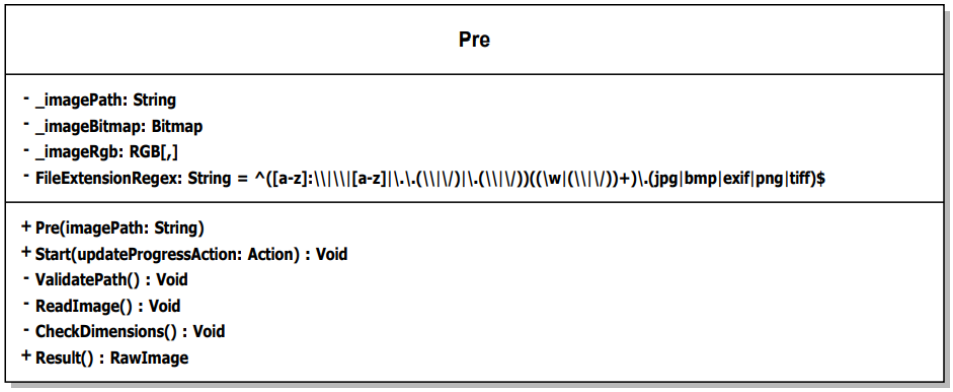
(31) Pathfind Image Form UML Diagram

Post *(Class)*



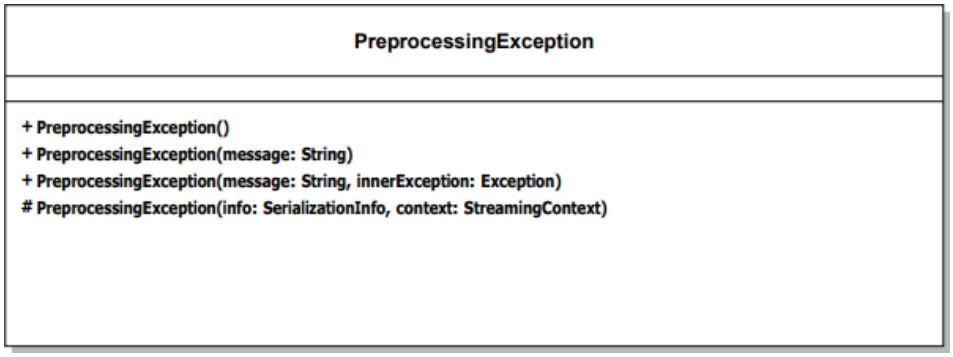
(32) Post Class Diagram

Pre *(Class)*



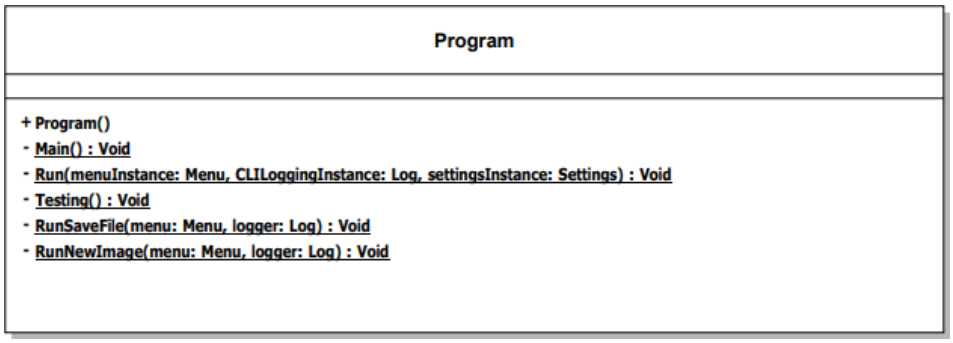
(33) Pre Class Diagram

Preprocessing Exception *(Exception)*



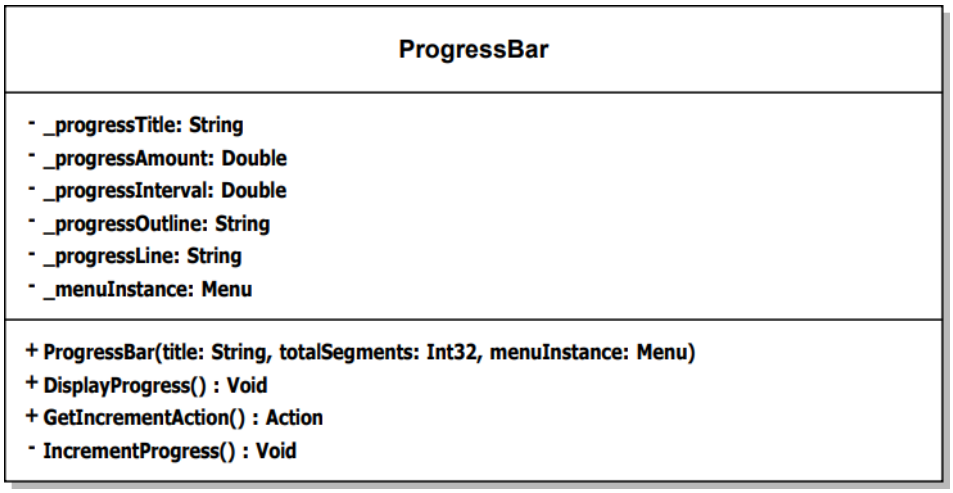
(34) Preprocessing Exception Class Diagram

Program *(Class)*



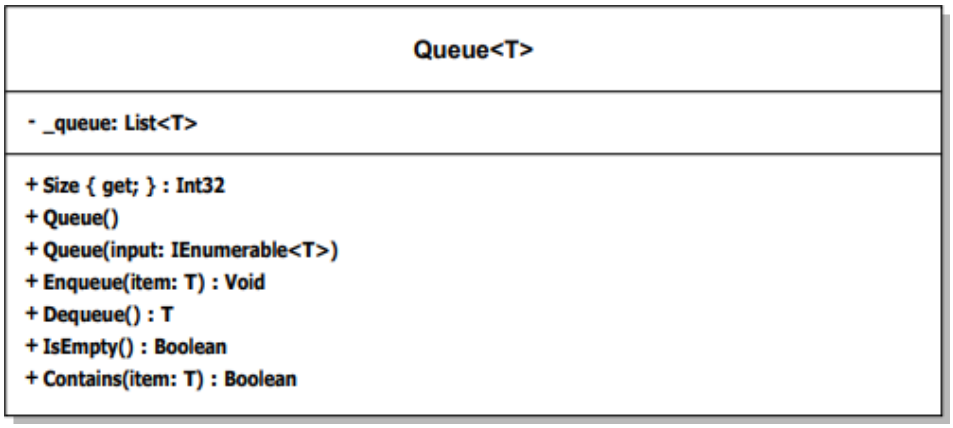
(35) Program Class Diagram

Progress Bar *(Class)*



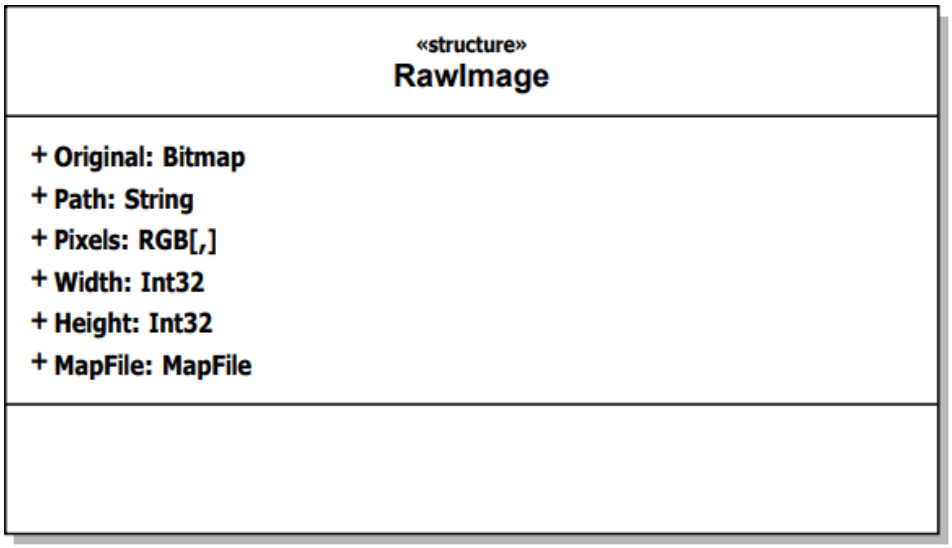
(36) Progress Bar Class Diagram

Queue *(Class)*



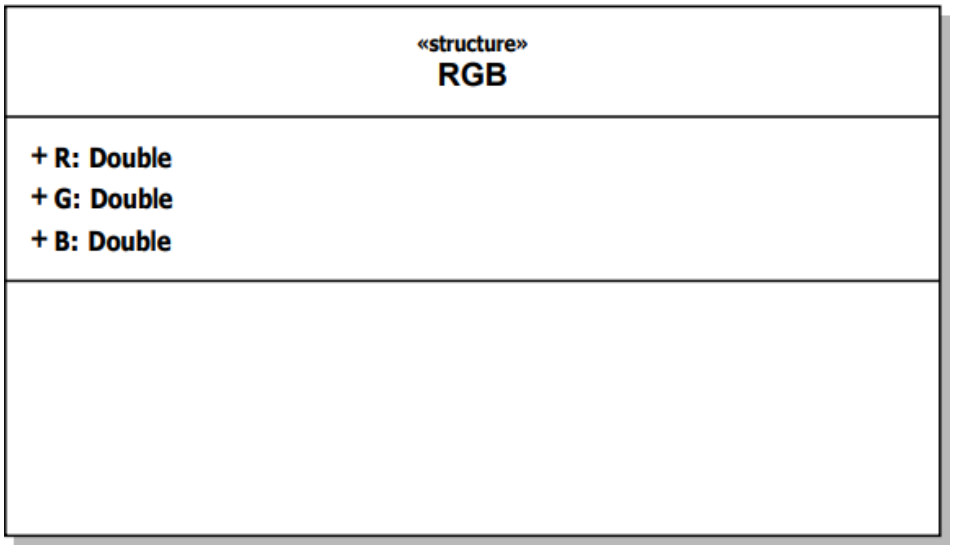
(37) Queue Class Diagram

Raw Image *(Structure)*



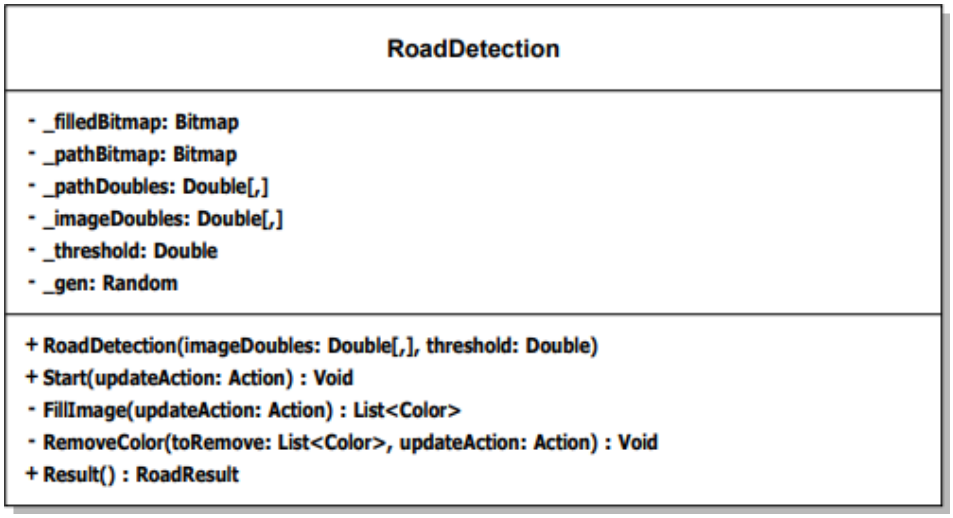
(38) Raw Image Class Diagram

RGB *(Structure)*



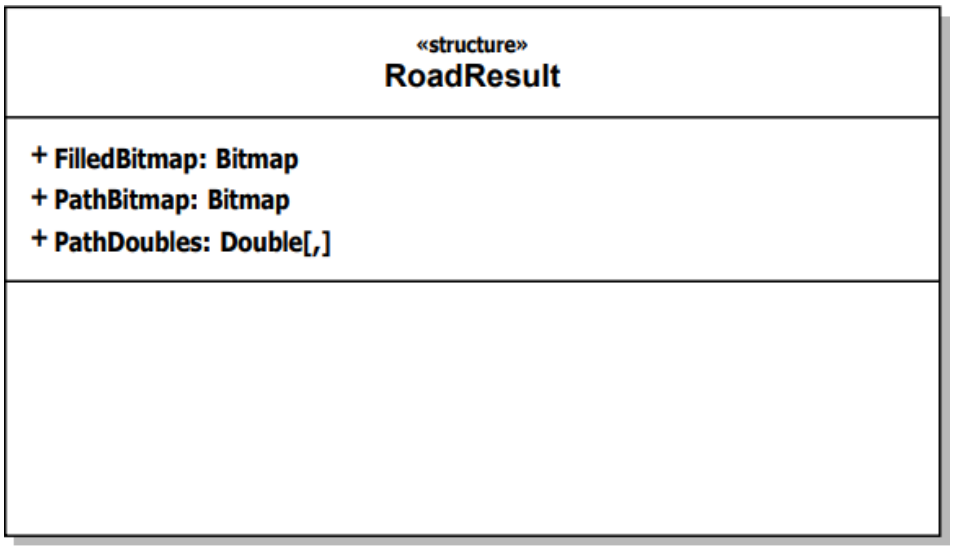
(39) RGB Class Diagram

Road Detection *(Class)*



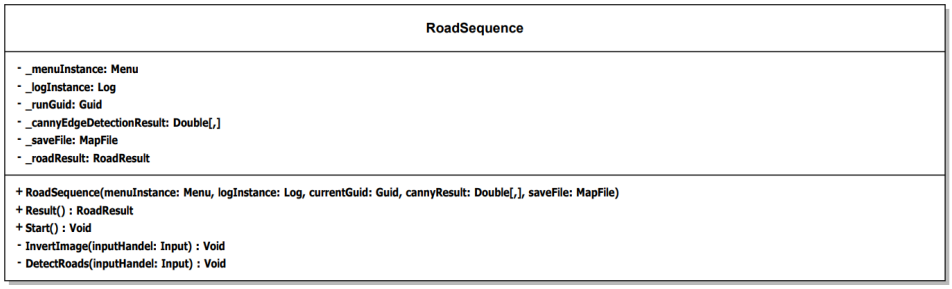
(40) Road Detection Class Diagram

Road Result *(Structure)*



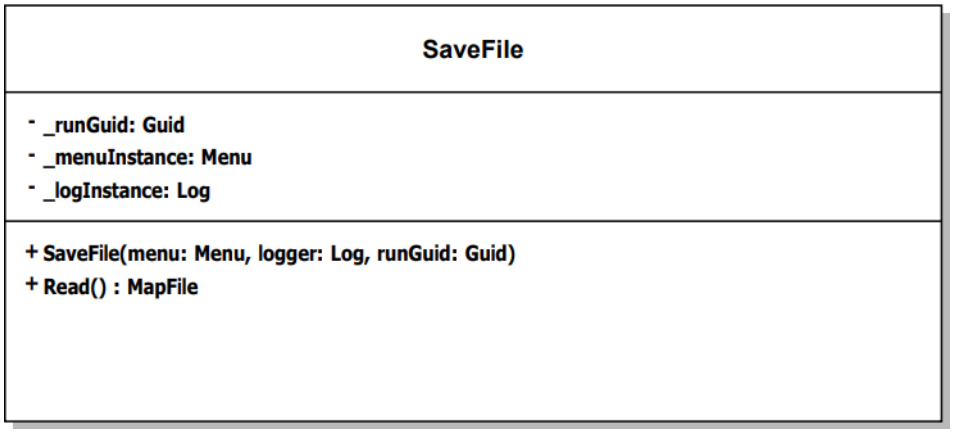
(41) Road Result Class Diagram

Road Sequence *(Class)*



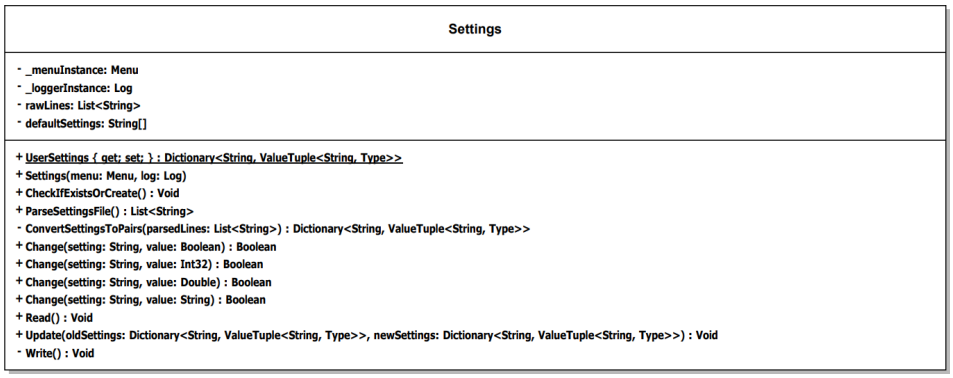
(42) Road Sequence Class Diagram

Save File *(Class)*



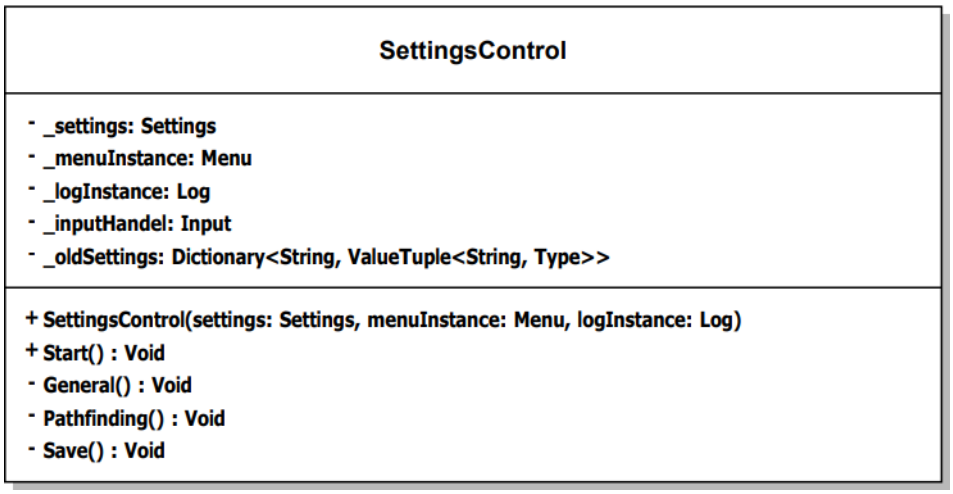
(43) Save File Class Diagram

Settings *(Class)*



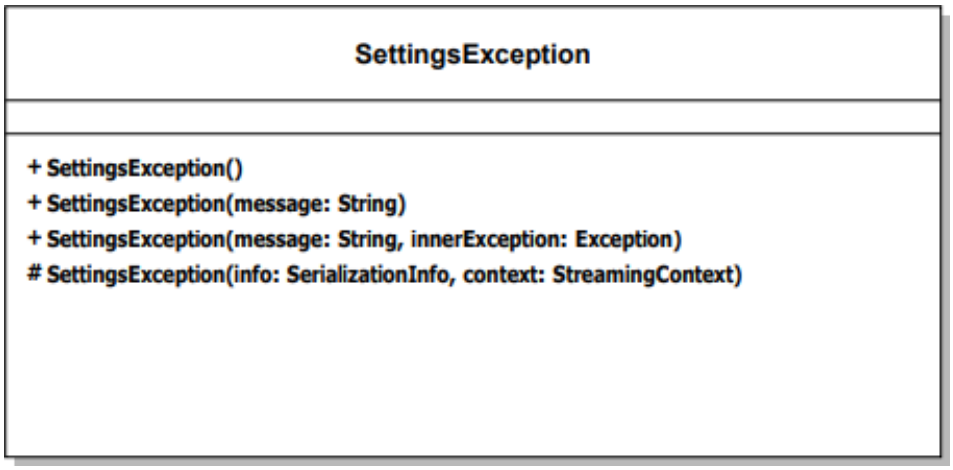
(44) Settings Class Diagram

Settings Control *(Class)*



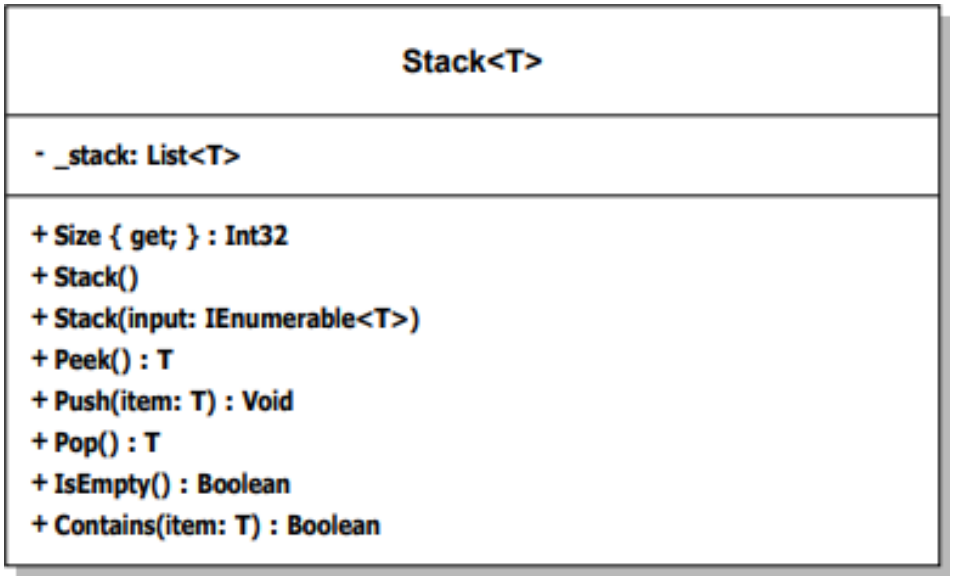
(45) Settings Control Class Diagram

Settings Exception *(Exception)*



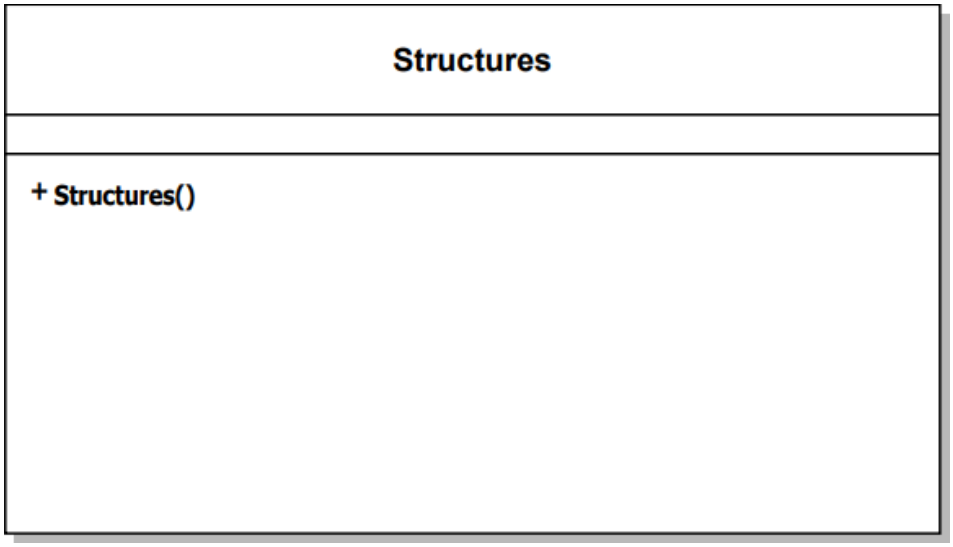
(46) Settings Exception Class Diagram

Stack *(Class)*



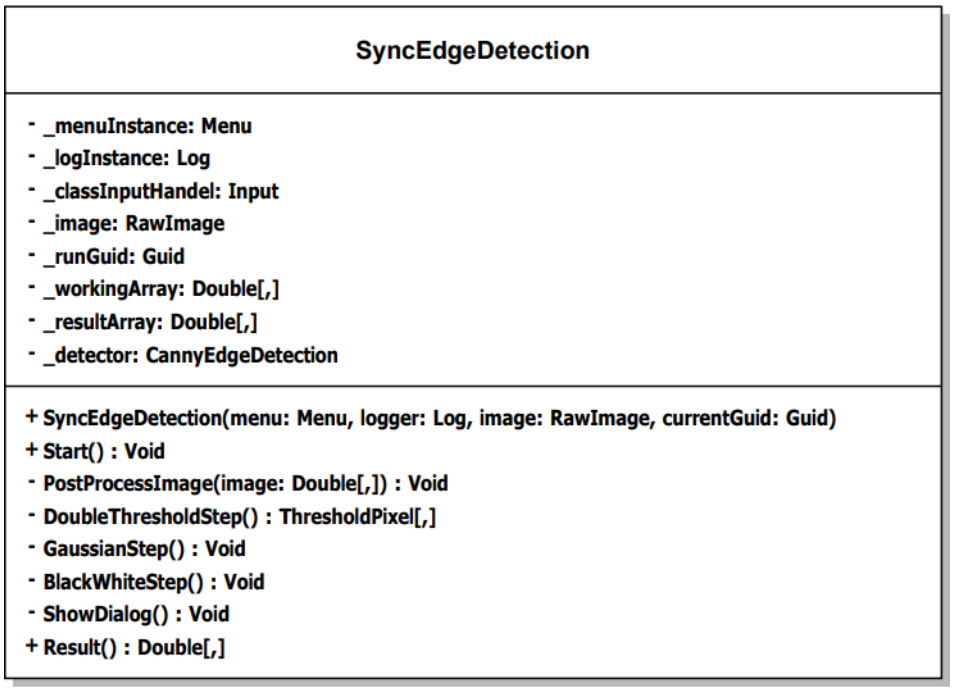
(47) Stack Class Diagram

Structures *(Class)*



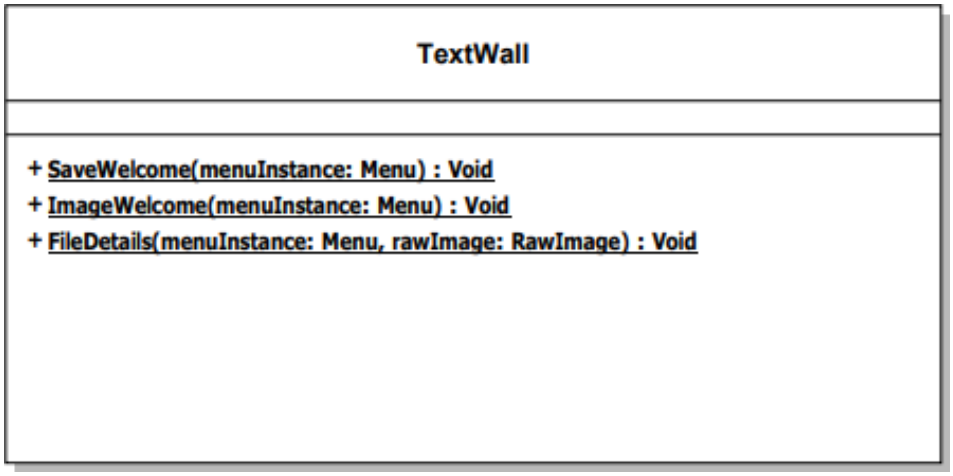
(48) Structures Class Diagram

Sync Edge Detection *(Class)*



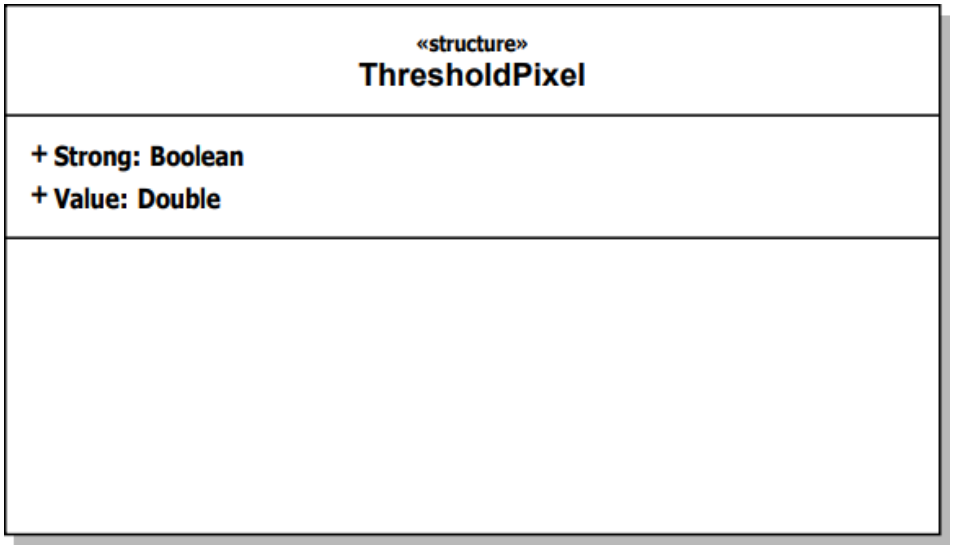
(49) Sync Edge Detection Class Diagram

Text Wall *(Class)*



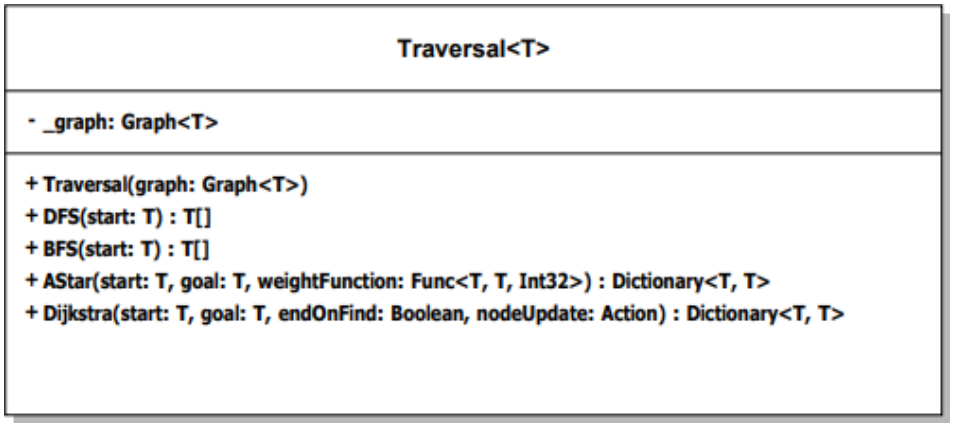
(50) Text Wall Class Diagram

Threshold Pixel *(Structure)*



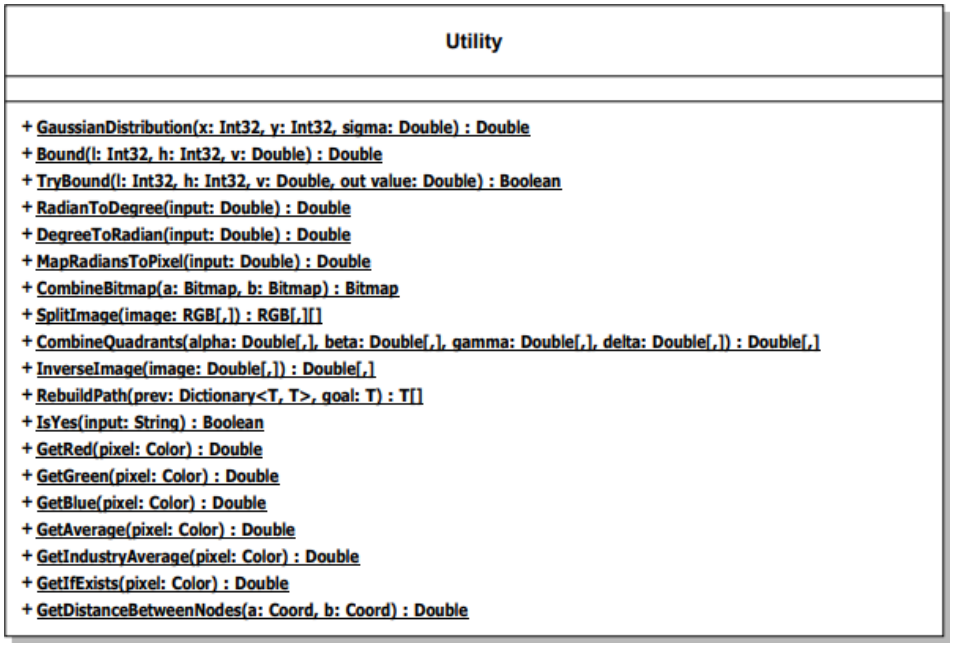
(51) Threshold Pixel Class Diagram

Traversal *(Class)*



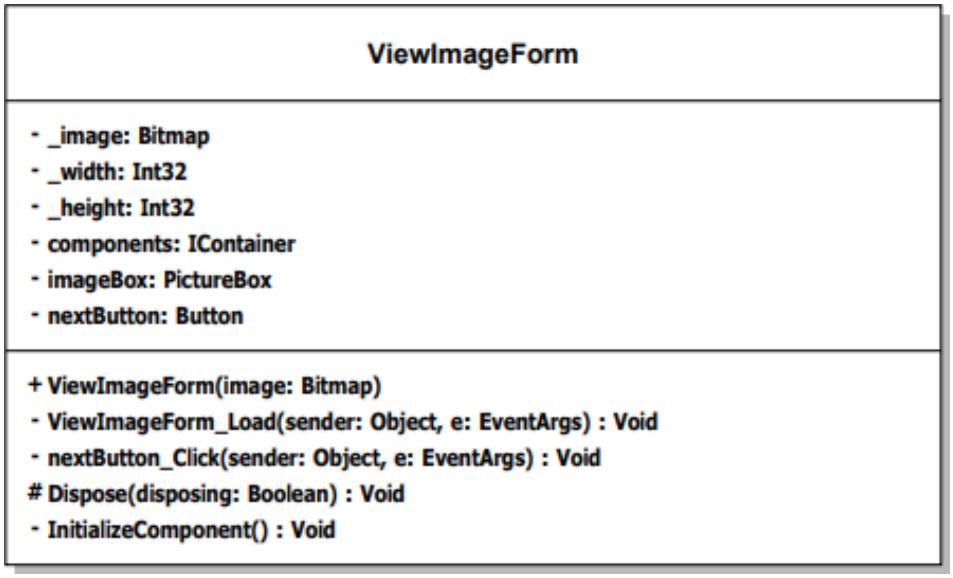
(52) Traversal Class Diagram

Utility *(Class)*



(53) Utility Class Diagram

View Image Form *(Windows Form)*



(54) View Image Form UML Diagram

3 Program Testing

3.1 Testing Tables

3.1.1 Targeted Testing Areas

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

1. User Map Inputs and Subsequent Outputs
   - 1.1 Loading In Image Files
   - 1.2 Creating The Save File
   - 1.3 Options Given To User
   - 1.4 Conversion To Graph
   - 1.5 Error Handling
2. Canny Edge Detection Operations
   - 2.1 User Variables
   - 2.2 Constructor Arguments
   - 2.3 Full Flow Thorough
   - 2.4 Individual Method Calls
   - 2.5 Exceptions
3. Road Detection
   - 3.1 User Variables
   - 3.2 Constructor Arguments
   - 3.3 Full Flow Through
   - 3.4 Individual Method Calls
   - 3.5 Exceptions
4. Graph Traversal
   - 4.1 Different Node Placements
   - 4.2 Different Algorithms
   - 4.3 Other Graph Settings
5. Logging and Saves
   - 5.1 Validity Of Save Files
   - 5.2 Contents of Log Files
   - 5.3 Save Settings
6. Miscellaneous Items + GUI
   - 6.1 GUI Elements
   - 6.2 Matrix Functions
   - 6.3 Extensions and Utilities
   - 6.4 Structures

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

3.1.2 User Inputs and Outputs Testing Table

Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
1.1.(2) The program should be able to parse a map from a file including...					
1	Entering a JPG	Enter the test image as a JPG into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
2	Entering a PNG	Enter the test image as a PNG into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
3	Entering a BMP	Enter the test image as a BMP into the "New Image" promp.	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
4	Entering a TIFF	Enter the test image as a TIFF into the "New Image" promp	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
1.1.1 A photograph of an map					
5	Entering a Photograph	Enter a photograph into the "new image prompt"	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
1.1.3 A hand drawing of suitable quality (if it is not a message should be shown)					
6	Entering a Hand Drawing	Enter a hand drawing into the "new image prompt"	The program should accept the image and be able to process it and show it to the user in the "Preview Form"	Pass	TODO
1.4 A hand drawing of suitable quality (if it is not a message should be shown)					
7	Entering a Small Image (less than 200x200)	Resize test image to be less than 200x200 and then input that into the "New Image" prompt	The program should reject the image and instruct the user as to how to fix the issue.	Pass	TODO

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

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

3.1.3 Canny Edge Detection Testing Table

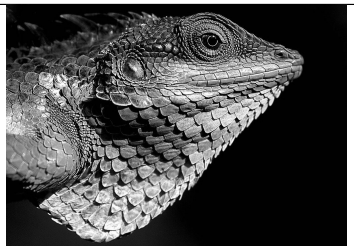
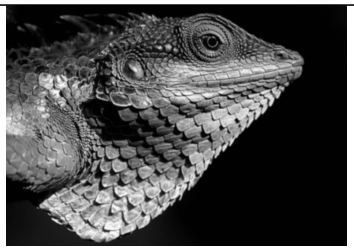
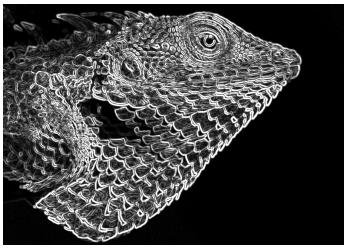
Test No.	Name	Input Data / Description	Expected Output	Pass Fail	Test Evidence
2.1 At each stage of the edge detection an image should be produced					
1	Canny Edge Detect Save Images	Run through a full map detection and at the prompt when it asks if the user would like to save an image at each stage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

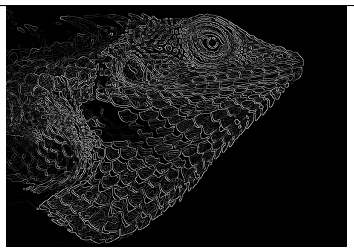
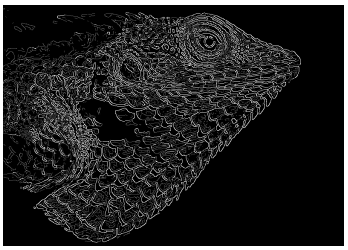
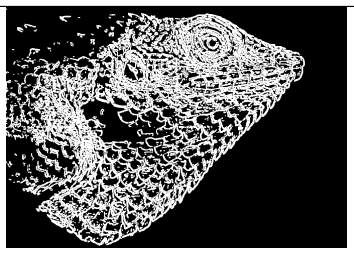
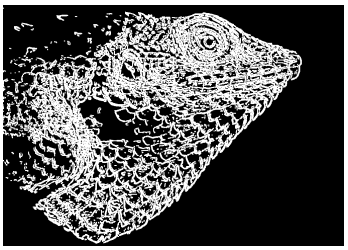
3	Cancel A Run	As above the test image should be entered. Both when it comes to the edge picking "Multi-threaded" then entering values then when the program confirms to continue select "No", and when the image is first read selecting "No" when the "Correct Image" prompt shows.	The program should stop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

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

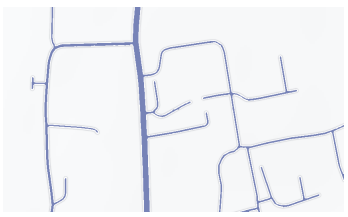
(*55*) Example Image Used

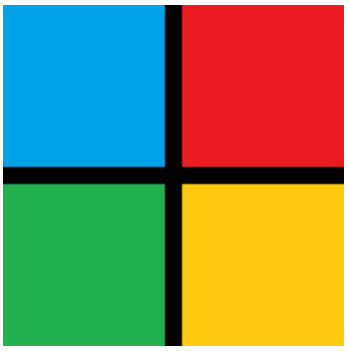
Sourced from Wikipedia©
`https://en.wikipedia.org/wiki/Canny_edge_detector#Walkthrough_of_the_algorithm`

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

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

3.1.4 Road Detection and Graph Conversion Testing Table

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

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

3.1.5 Graph Traversal Testing Table

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

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

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

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

3.1.6 Logging and Saves Testing Table

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

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

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

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

3.1.7 Miscellaneous Testing Table

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

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

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

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

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