

# Computing Coursework

Samuel Young

<candidate ID>  
<centre ID>

**Note**

Layout is *not final* and will be  
fixed closer to the end of the  
project.



# 1. Analysis

## 1.1 The task

---

The task is to create a painting program for a Windows PC, which allows users to create images, and then save and share them.

The program will feature:

- Brushes
- Other area filling tools
- Layer system
- Saving and loading files
- Importing and exporting images

The program would be suitable for a computer due to:

- A computer can save editing history, allowing for easy undoing of errors. Doing this in real life either requires an eraser (which can be messy) or is impossible.
- The mouse pointer allows precise locations to be selected quickly, and many points can be quickly inputted using the mouse
- Computational methods allow problems impractical with pens are solvable using a computing algorithms, such as filling an area, replacing a colour and creating a gradient
- Files can be saved and shared electronically, and quickly duplicated. This makes it easier and faster to share with users
- The keyboard allows text to be entered and manipulated quickly, often faster than by handwriting

## 1.2 Stakeholders

---

I have approached a small group of Stakeholders. They are students who use art programs often, such as art students, and other frequent art creators. They have agreed to provide feedback and help suggest features for the program.

Other people who may use the program are:

- Other graphical design students
- People wanting to make simple image edits
- Younger users who may be overwhelmed with more complicated image editing packages
- Those not wanting to make physical images

The painting program will be suitable for them as they all use computers and have an interest in creating images.

The program will result in digital images that can be easily shared, which is a value to the stakeholders as they primarily create digital art.

## Stakeholder profiles

**Name:** Alex G

**Art Background:** Uses GIMP primarily to create small web comics from scratch, using a variety of pen and fill tools.

**Would benefit from:** Tools for pens, fill tool, image exporting into formats, fast speed.

**Name:** Alex H

**Art Background:** Uses Paint to make small edits to existing images, does not need advanced features, just an easy-to-use package.

**Would benefit from:** Image importing and exporting, simple GUI, commonly use tools are easy to select

**Name:** Dheshpreat

**Art Background:** Uses a high end graphics creation tool to create large and detailed images, for an art qualification.

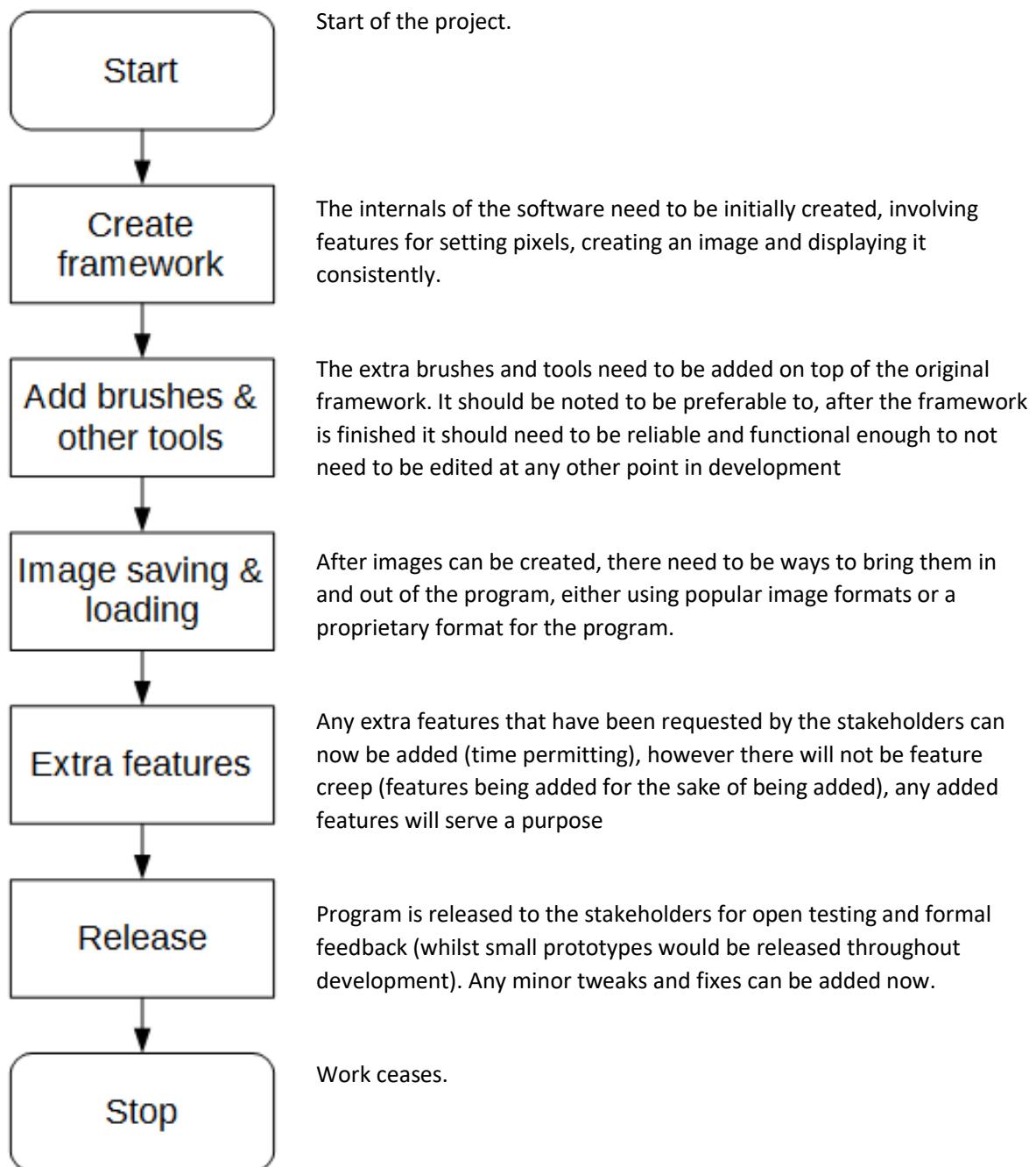
**Would benefit from:** More advanced tools, and support for larger images.

## 1.3 Computational Justification

The problem lends itself to computational methods very well as:

- There is a large amount of scope for **decomposition**. After the initial groundwork is set (setting pixels, resizing and displaying the image), most additional features can be built interdependently on top of the framework. This relies on a good and consistent level of **abstraction**, so that I am able to come back to previous classes and interact with them in an intuitive way.
- Because of these features **divide and conquer** can be implemented and so each tool can be worked on independent of any other, and then linked together on a GUI at project completion
- Areas for **abstraction** are:
  - The display can be abstracted into a two dimensional board of 'pixels', squares containing a single colour.
  - The picture will completely composed of these squares, no other shape will be possible
  - The brushes will be abstracted as methods to set these pixels in certain patterns.
  - All brushes and tools will have infinite 'paint', it is impossible to run out of a colour
  - All brushes and tools will be instantaneous, they will complete their task as fast as computationally possible
  - Paint will be abstracted to an RGB standard colour. The 3-byte standard does not cover every colour combination possible in the real world but will have a large enough colour depth for this program.
- The problem can be largely decomposed into these sections:

## Abstraction Diagram



## 1.4 Interview

---

### 1.4.1 Question Plan for online form

These questions will be used for an online form (likely using Google Forms) that will be sent to a large selection of students in my class. They are all competent computer users who will have had experience in creating images in the past, so should help provide some valuable feedback.

#### 1.4.1a Questions:

Question 1: What main features are you looking for in an image editing program?

This question is intentionally very broad. This is to provide initial pointers on what is a good decision for the program.

Question 2: What brushes (or other drawing tools) do you commonly use to create content? when creating an image and what are their important features?

This question is here to get an understanding of what tools are commonly used when the user edits an image. This is to get a feeling of what tools are most directly needed, and cannot be missed.

Question 3: What other tools in your image editing program do you commonly use when editing images, and what are their important features?

This question focuses less on the brushes and more on other image editing tools, such as layering or resizing of sections, to help gain other responses independent of the second question.

Question 4: What image editing program(s) do you currently use?

This question is here to get an understanding of what programs are already in use, so that they can be researched to find out what makes them so effective and popular.

Question 5: What image formats do you commonly use? (png, jpeg) Please put them in order of most used to least used.

This question is here to get an understanding of what image formats are wanted, so that they can be both imported and exported in the final program.

Question 6: What do you use image editing programs for?

This question is here to get an understanding of what the purpose of their image editing is. This can help inform me when making decisions as to what features should be prioritized.

#### 1.4.1b Responses:

The survey received seven responses from potential users, the feedback will be summarized here:

Question 1: What main features are you looking for in an image editing program?

There were many varying responses to this question, including:

- Speed
- Copying and Pasting tools
- Cropping images
- Text
- Layers
- Exporting and Importing images

Question 2: What brushes (or other drawing tools) do you commonly use to create content?

There were also a large amount of responses to this question, including:

- Straight line tools
- Magic select
- Eraser
- Fill (bucket) tool
- Hard pencil
- Soft brush

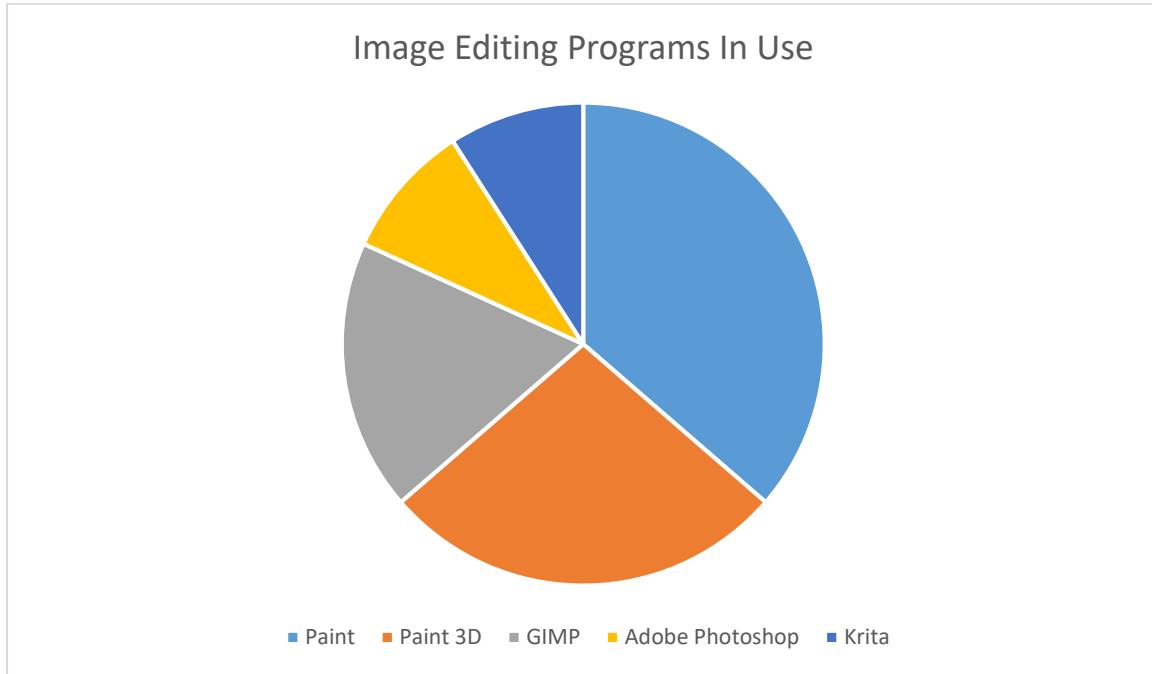
Question 3: What other tools in your image editing program do you commonly use when editing images, and what are their important features?

There was a large amount of image editing programs used, so many different features were suggested, including:

- Crop
- Marquee
- Image thumbnail display
- Changing visibility for certain layers

Question 4: What image editing program(s) do you currently use?

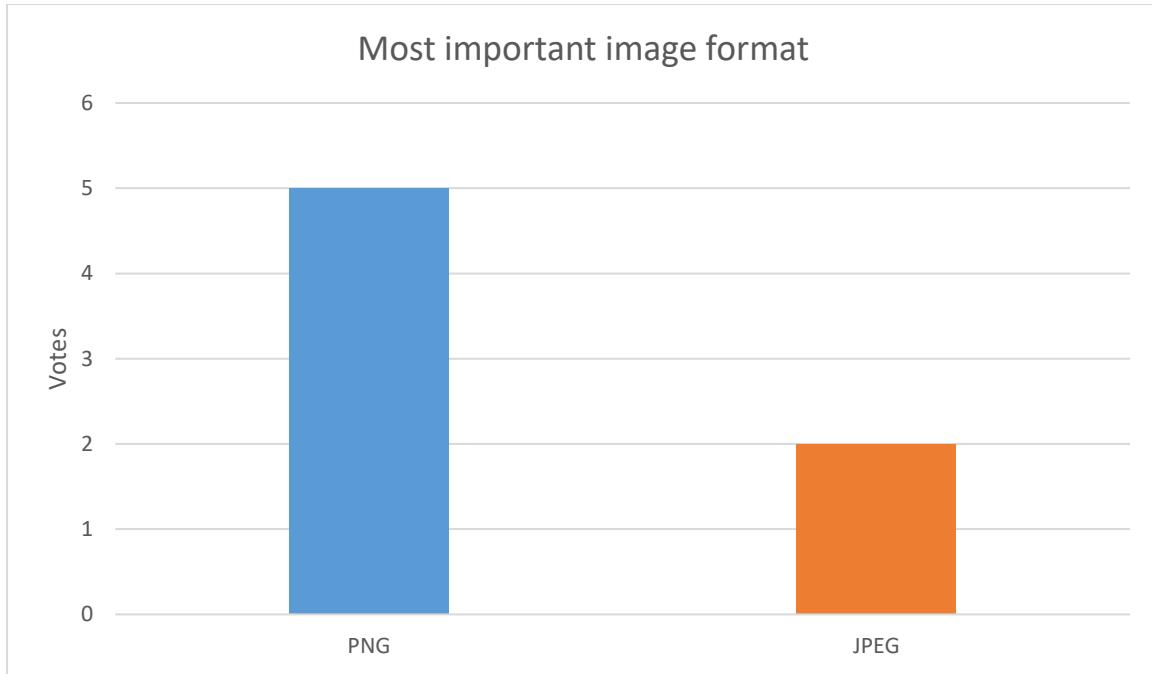
As stated above, many programs are in use, which are summarized in this pie chart:



From this diagram it becomes clear that the default Microsoft packages (Paint & Paint 3D) are the most popular ones, with GIMP coming in third place and the more powerful packages (Krita & Photoshop) less in use.

Question 5: What image formats do you commonly use? (png, jpeg) Please put them in order of most used to least used.

The split between which was the primary image format is split between png and jpeg, with first place priority given as follows:



This shows that there is a much higher demand for PNG images over JPEG images, so they should be prioritized during implementation. JPEG remained very popular however, almost always placing second where it did not place first.

Other image formats suggested were GIF, BMP and ICO.

Question 6: What do you use image editing programs for?

It became clear that many used image manipulation programs for very simple image edits (for 'memes'), whilst others used it for creating original artworks. It is clear that the user base is split between editing and creating, so there should be features to cater for both.

#### 1.4.1c Conclusion

From this market research it is clear that there is a lot of difference in opinion on necessary features for the program. The main pieces of advice from the market research is that:

- The requested tools are:
  - Copying and Pasting tools
  - Cropping images
  - Text
  - Layers
  - Changing visibility for certain layers
  - Exporting and Importing images
  - Straight line tools
  - Magic select
  - Eraser
  - Fill (bucket) tool
  - Hard pencil
  - Soft brush
  - Crop
  - Marquee
  - Image thumbnail display
- Paint and Paint 3D are the most commonly used image editing programs, so cues from them are the most valuable.
- PNG and JPEG formats are the most widely in use.

#### 1.4.2 One-to-One interviews

These interviews will be conducted with a very small and select group of art program users. After completing the larger online form interviews, I now have an idea of what is being looked for. I will ask follow on questions from the responses given.

Stakeholder A – Alex G

**Name:** Alex G

**Art Background:** Uses GIMP primarily to create small web comics from scratch, using a variety of pen and fill tools.

**Would benefit from:** Tools for pens, fill tool, image exporting into formats, fast speed.

Question 1: What main features are you looking for in an image editing program?

*“I am looking for a dark mode, potential dock-able windows, layer tools and exporting transparent PNGs.”*

Question 2: What brushes (or other drawing tools) do you commonly use to create content?

*“I mainly use the hard pencil tool when making images, and I like that you can exactly specify the brush size, which you cannot do in paint.”*

Question 3: What other tools in your image editing program do you commonly use when editing images, and what are their important features?

*“I use the control key + scroll a lot to zoom in and out of the image.”*

Question 4: What image editing program(s) do you currently use?

*“GIMP”*

Question 5: What image formats do you commonly use? (png, jpeg) Please put them in order of most used to least used.

*“Mostly large PNGs.”*

Question 6: What do you use image editing programs for?

*"I use GIMP to draw comics for an online audiences, which are large drawings made completely from scratch."*

Question 7: What features do you use less often?

*"I don't often use the smudge tool."*

Stakeholder B – Alex H

**Name:** Alex H

**Art Background:** Uses Paint to make small edits to existing images, does not need advanced features, just an easy-to-use package.

**Would benefit from:** Image importing and exporting, simple GUI, commonly use tools are easy to select

Question 1: What main features are you looking for in an image editing program?

*"In editing images, I often use copy and paste, brushes, fill tool, text is a must, and layers. I also like those artistic effects from Microsoft Word, such as grayscale and sepia."*

Question 2: What brushes (or other drawing tools) do you commonly use to create content?

*"I often use the brushes for large areas, and the finer single pixel pencil tool for fine work. I also use the Fill and a Rubber. Often I want to reuse a colour, so I would also like an eyedropper tool."*

Question 3: What other tools in your image editing program do you commonly use when editing images, and what are their important features?

*"I've used in the past the Blur tool, which averages the colour between two areas. I would also like to be able to resize images, and rotate them. I would expect rotations of 90° but would like to be able to rotate around a full 360°.*

*In terms of the colour picker, I would like it to allow me RGB input, as I often get colours from online. Also it would be good if it would save colours I've used temporarily, but they don't need to be stored with the actual art piece. A basic set of colours to use would also be good."*

Question 4: What image editing program(s) do you currently use?

*"I mostly used Paint and Paint 3D. I sometimes use piskel for making small sprites for games, and have used GIMP & Inkscape in the past."*

Question 5: What image formats do you commonly use? (png, jpeg) Please put them in order of most used to least used.

*"Mainly PNG or JPEG, what the default is. I've also sometimes used GIFs for animation."*

Question 6: What do you use image editing programs for?

*"I've recently used image editing programs for my Geography coursework, where I labelled routes on an existing map, created a 'North' arrow, and added a 'scale'."*

Stakeholder C – Dheshpreet

**Name:** Dheshpreet

**Art Background:** Uses a high end graphics creation tool to create large and detailed images, for an art qualification.

**Would benefit from:** More advanced tools, and support for larger images.

Question 1: What main features are you looking for in an image editing program?

*"I'm looking for a good selection of brushes, especially hard brushes, a blending tool, a good fill tool, and drawing tablet compatibility, as it's what I often use. Also I'd like to see layers, and clipping masks"*

Question 2: What image editing program do you most often use?

*"Krita"*

Question 3: What do you like about Krita?

*"It's easy to use, supports graphics tablets and has lots of advanced tools"*

Question 4: What do you use image editing programs for?

*“Mostly drawing images, or fixing up physical drawings.”*

Question 5: So you'd need to be able to import / export images?

*“Yeah, mostly PNG and JPEG”*

## 1.5 Existing solution research

---

There are many existing image manipulation packages available. I will research the features of a variety of these and tabulate the main features of them.

The plan for research cover these headings:

- Program type (open source, free etc)
- GUI and main features (with annotated screenshot)
- Table of features.

The tables of features will be combined in the end, and will include whether this feature will be included in the painting program with a justification of why.

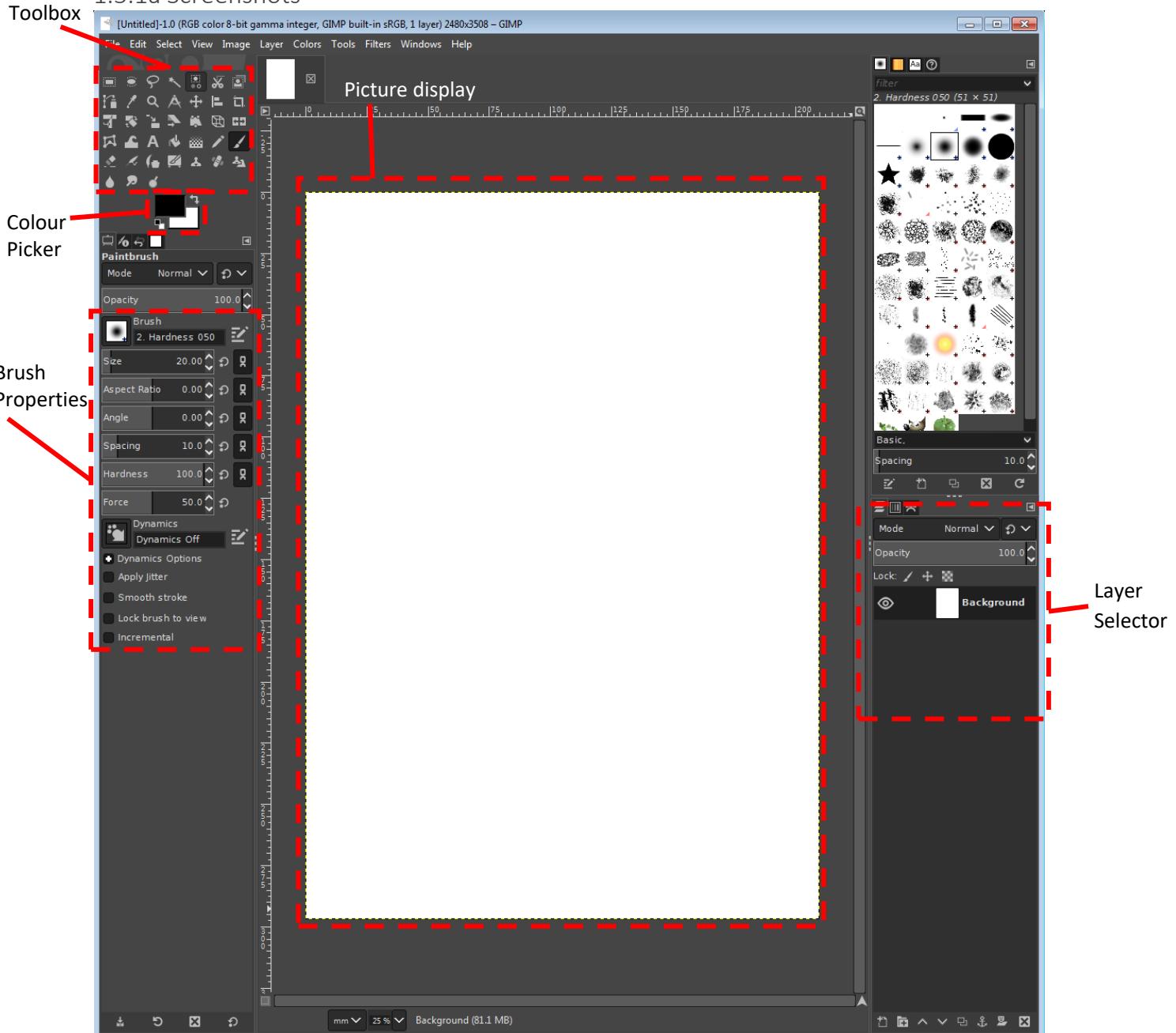


## 1.5.1 GIMP

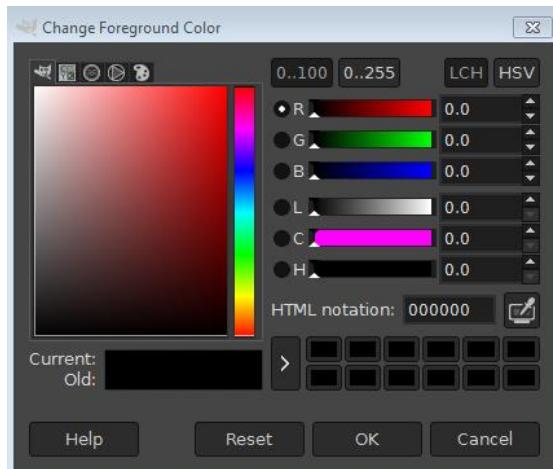
GIMP, (GNU Image Manipulation Program) is an open source image manipulation package for Windows, Linux and Mac. It is free to download and view its source code.

The version being viewed is GIMP 2.10

### 1.5.1a Screenshots



A screenshot taken showing GIMP 2.10



The RGB picker from GIMP.

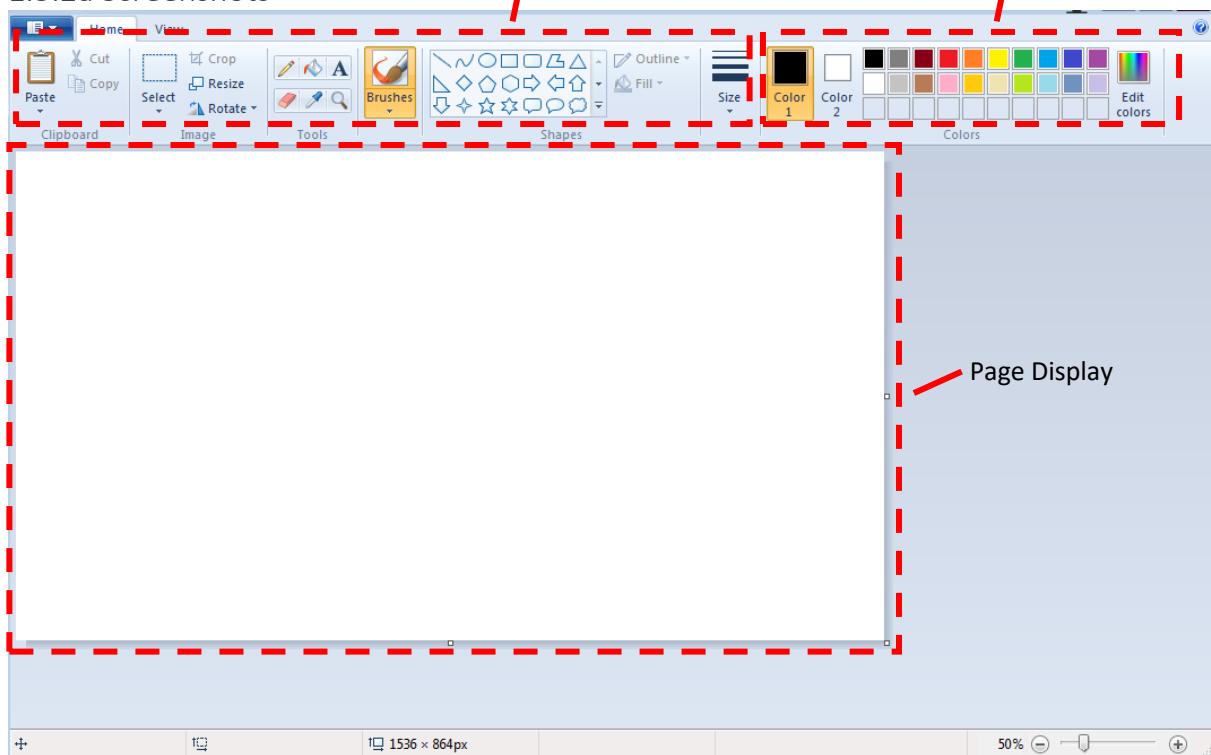
### 1.5.1b Features

Feature	Description
Rectangle Selection Tool	Allows a rectangle to be selected on the image, to be manipulated
Magic Selection Tool	Selects a region based on similar colours, used for quickly selecting an object with a complex border
RGB Selector	RGB Selector for selecting a colour quickly and accurately. A square shows the available shades of a colour whilst a range of colours is shown off to the side
Colour Picker	Allows you to select an existing colour on your image to use again
RGB exact input	Allows you to enter an exact colour using standard notation for example #FF0000
Soft brush	A brush with softer edges for the document
Hard brush	A brush with completely sharp edges filling solid colour
Brush size manipulation	The brush's size can be changed from 1 pixel across to many circles
Layer tools	Layers can be hidden, shown and changed in order. Each layer is a separate drawing
Transparency support	Pixels can have varying levels of opacity.
Supported image formats	Supports nearly all major image formats, with extension support for obscure ones
Zoom	Images can be zoomed very freely, to fill the screen
On-screen ruler	A sense of scale can be gained using an on-screen ruler
History Viewer	No support

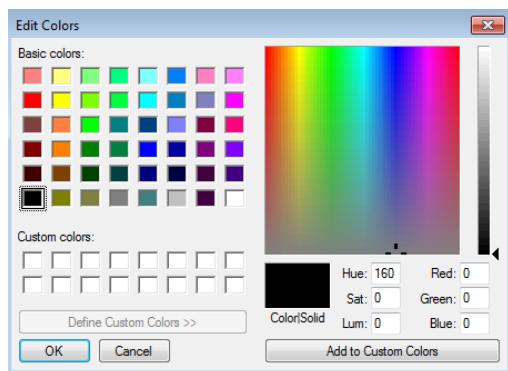


## 1.5.2 Microsoft Paint

### 1.5.2a Screenshots



A screenshot taken showing Paint.



The RGB picker from Paint.

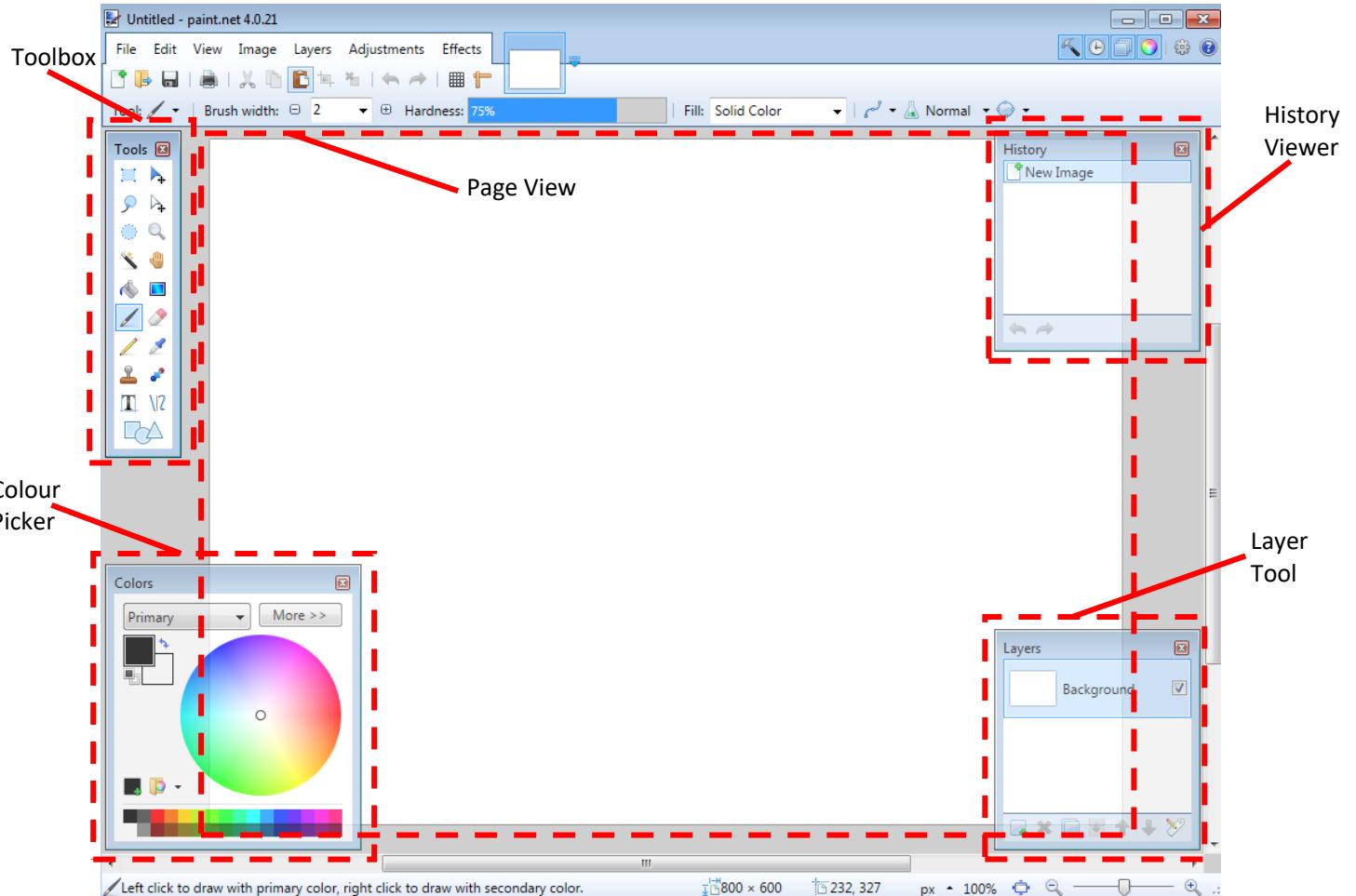
### 1.5.2b Features

Feature	Description
Rectangle Selection Tool	Allows a rectangle to be selected on the image, to be manipulated
Magic Selection Tool	Not available.
RGB Selector	RGB Selector for selecting a colour quickly and accurately. A square shows the available colours whilst a range of shades is shown off to the side
Colour Picker	Available, but not in the colour selection menu
RGB exact input	Available, but is more finicky as involves three smaller number boxes, does not support standard formats
Soft brush	A brush with softer edges for the document
Hard brush	A brush with completely sharp edges filling solid colour
Brush size manipulation	The brush's size can be changed to a number of pre-sets, however only 6 settings are available
Layer tools	No support
Transparency support	No support
Supported image formats	Supports png, bmp, gif, jpeg
Zoom	Images can be zoomed to intervals of 25%, to a max of 200%. Not very free
On-screen ruler	No support
History Viewer	No support



### 1.5.3 Paint.NET

#### 1.5.3a Screenshots



### 1.5.3b Features

Feature	Description
Rectangle Selection Tool	Allows a rectangle to be selected on the image, to be manipulated
Magic Selection Tool	Available
RGB Selector	RGB Selector for selecting a colour quickly and accurately. A square shows the available shades of a colour whilst a range of colours is shown off to the side
Colour Picker	Allows you to select an existing colour on your image to use again
RGB exact input	Allows you to enter an exact colour using standard notation for example #FF0000
Soft brush	A brush with softer edges for the document
Hard brush	A brush with completely sharp edges filling solid colour
Brush size manipulation	The brush's size can be changed from 1 pixel across to many circles
Layer tools	Layers can be hidden, shown and changed in order. Each layer is a separate drawing
Transparency support	Pixels can have varying levels of opacity.
Supported image formats	Supports nearly all major image formats, with extension support for obscure ones
Zoom	Images can be zoomed very freely, to fill the screen
On-screen ruler	Not available
History Viewer	Shows a list of the user's previous actions, allows them to navigate and undo certain actions in a user friendly way

### 1.5.4 Comparison of packages

Feature	GIMP	Paint	Paint.NET	Conclusions & Application
Rectangle Selection Tool	✓	✓	✓	The program should include a rectangular selection tool, as it is simple to program and is included in all researched art creation programs
Magic Selection Tool	✓	✗	✓	The program should include some sort of magic selection tool, as many art creation programs utilise it in some form.
RGB Selector	✓	✓	✓	The RGB selector should include a tray for recently used colours, as all three of the analysed programs had some sort of previous colour feature
Colour Picker	✓	✓	✓	The RGB selector is very important to the program, and the design should be considered. GIMP uses a square of shades, Paint uses a square of colours, Paint.NET uses a circular colour selector with limited support for shades.  The program should make sure that a specific colour can be quickly selected, and its shade changed.
RGB exact input	✓	✗	✓	RGB should be able to be exactly entered in order with the standard format #RRGGBB
Soft brush	✓	✓	✓	A rudimentary form of soft brush is often used in art programs
Hard brush	✓	✓	✓	A hard brush is very simple to create and should be in the program
Brush size manipulation	✓	✗	✓	Whilst Paint does not include a brush width editing feature, the other two packages do and it is more useful to be able to fine-tune the width
Layer tools	✓	✗	✓	This is also a feature missing from the more basic Paint, but is necessary for complex picture creation
Transparency support	✓	✗	✓	This is also a feature missing from the more basic Paint, but is necessary for complex picture creation
Supported image formats	Many	Few	Few	GIMP supports a wider range of image formats than the other packages, however in practical use the fewer formats provided by Paint and Paint.NET are needed.
Zoom	✓	✓	✓	Whilst all packages contain a zoom, they come in varying levels of freedom of zoom. Paint has a very limited degree of freedom, whilst GIMP allows

				almost any level of zoom. This means that there should at least be the same level of zoom as Paint.
On-screen ruler	✓	X	X	This is shown to be a rather niche feature, being only included in GIMP, meaning that it is likely that there is low demand for this feature, so it is not needed.
History Viewer	X	X	✓	This is also shown to be very niche, and the rest of the programs suffice with a simple Undo tool rather than an entire window

### 1.5.5 Conclusions

From this research I can conclude that, as I found from my market research, Paint is the most applicable package to draw inspiration from for the program. It is the most commonly used program by my stakeholders, and as a proprietary Microsoft product packaged with Windows, its features will have been well considered by a large team.

Features that I will utilise from each package:

#### 1.5.5a GIMP

The layer system from GIMP will be utilised, as it is the most commonly used program to utilise layers. Other layer features such as hiding and showing layers will also be included.

The GUI however is quite complex, containing a lot of options & additional features, so the GUI from GIMP would not be appropriate for this program, which will include comparatively fewer features, so does not need a complex GUI

Other features lifted from GIMP will be many of its brush and editing tools, and its support for transparency.

#### 1.5.5b Paint

From Paint, features such as its shape tool will be included, as well as its implementation for inputting text.

A lot of inspiration will be taken from the GUI, as Paint contains relatively fewer tools than the other packages, so it has a good layout for few total options. This helps improve clarity and speed of operations as most features can be seen by the user without having to search menus.

#### 1.5.5c Limitations

The final product will be limited however as it will not be using some features from the research, such as not showing history of operations, making reverting actions more difficult, and it will likely not support many file formats (only JPEG, PNG) due to the large amount of work needed to research and implement other image formats, making it not worth the time.

## 1.6 Software and hardware requirements

---

### 1.6.1 Software

The program will require a Windows computer, to run the software on. Other operating systems will not be able to run the program, as they are largely incompatible and will require a re-write for each system, which is a lot more development.

The program will need up-to-date and standard drivers for all its relevant hardware, so that the hardware can communicate effectively with the program.

### 1.6.2 Hardware

#### 1.6.2a Input

The program will require a mouse or similar pointing device, to enter on-screen locations to draw the image or select points for editing, and for clicking at certain times to perform actions quickly.

The program will require a keyboard, for inputting text in a text drawing tool. As most of the stakeholders interviewed use a desktop computer for editing images, there should also be good key shortcut support in the program for those users who want to access those features quickly.

#### 1.6.2b Processing

The program will need a computer capable of running the Windows operating system, this is because the Windows operating system is a required base for the program

The program will need around 100MB of available RAM. This is so that it can hold the current image data in RAM, which may be quite large depending on the image size. The user will not want to run out of available RAM when using the program, and have to either use virtual memory (making image edits significantly slower) or having the program crash and potentially lose data.

#### 1.6.2c Output

The program will need a standard display output device, likely a monitor or projector, which can connect to the computer using a standard driver & Windows protocol. As long as the Windows operating system can display to the output device, so can the program.

The program will **not** need any sort of audio output device, as there is no point where sound will be necessary for its operation (it edits images). To make the program more accessible to the stakeholders who do not use speakers very often or do not have them, this output device will not be included.

#### 1.6.2d Storage

The computer will need a small amount of storage for the program, and its files. Whilst the executable itself is not likely to be very large, some of the larger image files will require more storage space. To help alleviate this, a small amount of lossless compression will be applied to the images, to help reduce the amount of storage needed.

## 1.7 Justification of features

---

### 1.7.1a Brushes

- Variable brush width
- Hard brushes
- Shape creation tools
- Fill (bucket) tool
- Single pixel pencil
- Rubber

### 1.7.1b Other image editing tools

- Bitmap image editor.
- RGB Colour picker
  - Based on the Microsoft Paint colour picker design
  - Allows direct RGB input
- Layer system
- Rectangle selection tool
- Magic selection tool
- Partial transparency support
  - There will be support for completely blank tiles (empty), for use with layers.
- Zoom in
- Text
- Image effects
- Eyedropper
- Rotating images through 90°
- Clipping masks

### 1.7.1c File system support

- Importing images of supported formats
- Exporting images of supported formats
- Supported image formats
  - PNG
  - JPEG
- Saving a proprietary format containing extra information about layers and other metadata.

## 1.7.2 Limitations

The program will **not** contain these features:

### 1.7.2a Brushes

- Soft brushes
  - Whilst brushes with a hard edge only involve a simple set of pixel colour, soft edged brushes have a much larger level of complexity. This is due to the need for colour mixing, where the edges of the brush will need to 'blend' into the surrounding colour, the calculations for which are out of the scope of the timing of the project.
  - The reasoning for this is covered in more depth at 'Other image editing tools' -> 'Full transparency support'
- Large amount of shapes

- This is to cut down on the amount of redundant code in the project. It may be possible to program an extensive list of shapes into the program, however this would be a lot of redundant coding work for shapes that may not be used often.
- Fast fill
  - The fill function is an example of where the algorithm makes a large impact on the time taken to complete the action. As with sorting, there are many algorithms for filling an area, with varying complexity. To cut down on time & complexity, and as most edited images will likely not be large to begin with, only a simple filling algorithm will be implemented.

### 1.7.2b Other image editing tools

- On-screen ruler
  - From the product research, it became clear than an on-screen ruler was a niche feature, not always necessary to be included in software packages
- History viewer
  - Similar to the On-screen ruler, the history viewer also had a smaller usage in market.
- Full transparency support
  - Whilst there will be support for fully transparent pixels (on layers), there will not be support for partially transparent pixels. (pixels with opacity). This is due to the complexity of the algorithms that determine the resultant colour of pixels, especially with multiple semi-transparent pixels combined together. There must be some degree of transparency in the program, or else the layer system would be useless, so this is a fair compromise between features and complexity.
  - It is for this reason that the soft brush cannot be implemented, as the soft brush would require opacity support.
- Zooming out
  - Whilst it is necessary to provide a 'Zoom in' feature, it is less necessary to provide the feature to 'Zoom out' (e.g. view the image at 50%). In addition to this, the algorithms determining which pixels are shown when the image is reduced in size are more complicated than the algorithms for increasing in size, as decisions must be made as to which pixels are shown.
- Vector image support
  - Vector images are handled completely differently to bitmap images, and so would require a lot of time (and a completely different toolkit) to implement, so is out of the scope of this project.
- Graphics tablet support
  - Will not have any graphics tablet support, as I do not own a graphics tablet.

## 1.8 Success Criteria

The 'code' field denotes a shortening that will be used to refer to that criteria, for example "Criteria A1". *Italics denote stretch goals – optional features added if there is enough time.*

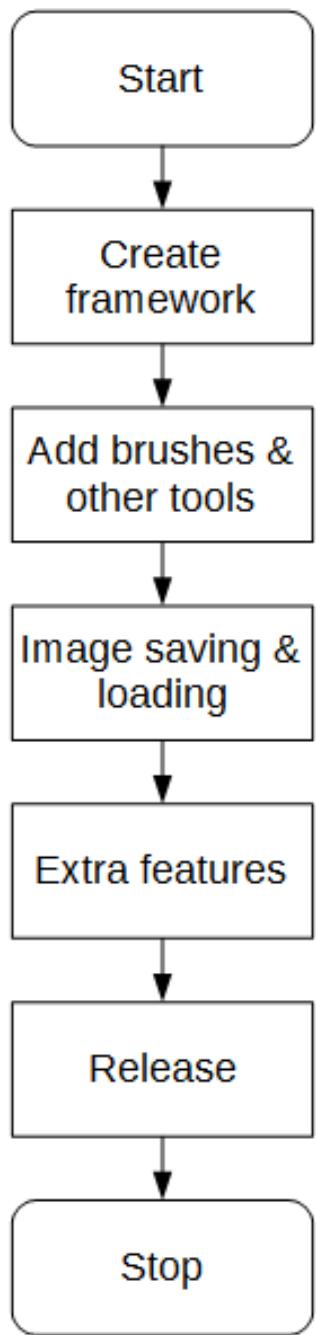
Feature	Proof	Code
<b>Section A - Brushes</b>		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
<b>Section B – Other editing tools</b>		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering "FF0000" (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text "Hello World" on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11
<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13

<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
-----------------------	--	-----

Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

## 1.9 Project Plan

Referencing the main diagram made earlier in the Analysis.



To decompose the project, each component on this diagram will be designed, coded and then tested in sequence, rather than doing all design at the start. This is to make the programming more dynamic, and show how the project builds up.

# Section 2 – Framework

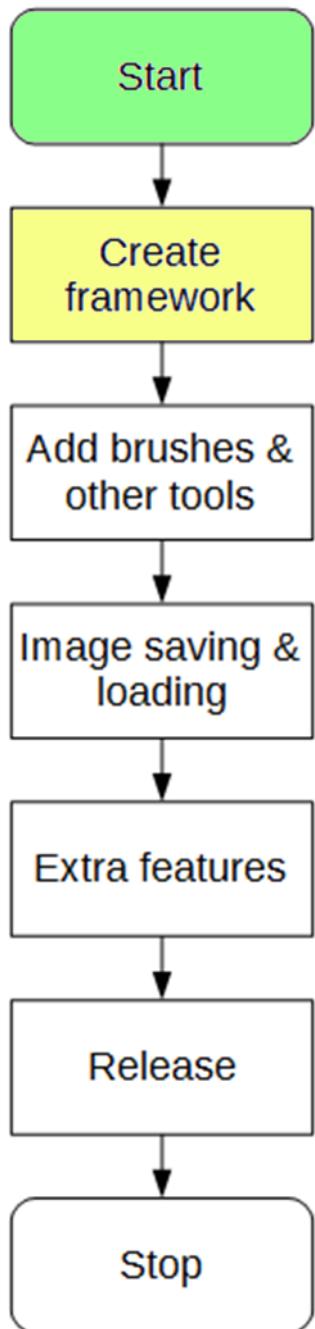
## General Contents

---

- [Decomposition Diagram](#)
- [Usability Diagram](#)
- [IPO Diagrams](#)
- [Class Design #1](#)
- [Algorithm Design](#)
  - Contains smaller Unit Tests
- [Class Design #2](#)
  - This supersedes the previous Class Design
- [Class Relation Diagrams](#)
- [Other key data structures](#)
- [Testing Table](#)
- [Testing Plan](#)

## 2.1 Design

The framework includes a lot of the internal design for the program. Classes, basic GUI and important functions will be designed here. After the framework is complete, the other functions should be able to be easily added on top.



## 2.1.1 Success Criteria Fulfilment Plan

In this section, the following success criteria are planned to be completed:

Not completed

To be done this section

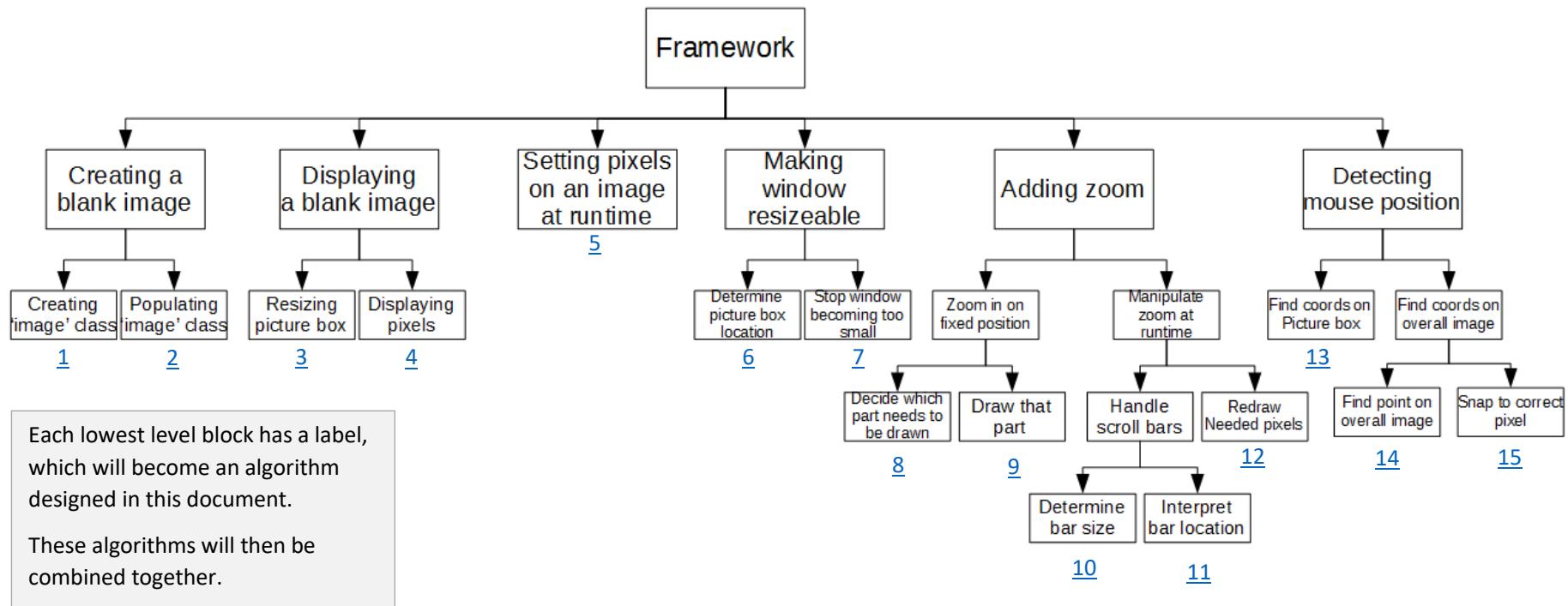
Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11

<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

## 2.1.2 Section Decomposition

### 2.1.2.1 Decomposition diagram



### 2.1.2.2 Diagram Justification

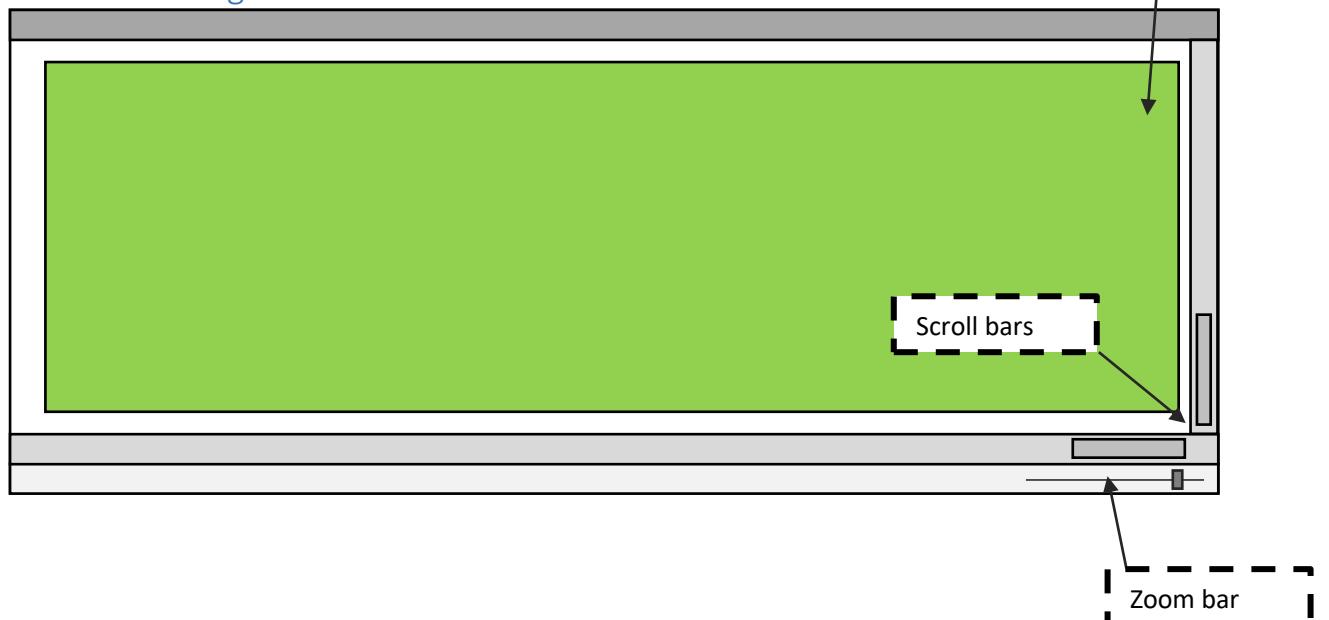
This will detail why each part of the top level of the diagram is needed for this stage in the development

Title	Justification	Fulfils
Creating a blank image	The very first thing that will need to be coded is defining an empty image, which will involve creating a base 'Image' class, and populating the image class with default values, with a flexible constructor.	C1
Displaying a blank image	After the image has been defined, it will need to be displayed in a window. Only a white box will be displayed at this point, and it will not need to move yet.	B1
Setting pixels on an image at runtime	After the image is successfully displayed, with the use of a temporary button, pixels of differing colours will need to set at runtime, showing that the image can change.	A2
Making window resizable	Currently all tests are being performed on a static window of fixed size, now the program will need to appropriately place the picture box in the window. The size will not be changed at this point (as zoom is not added yet) but the picture box should be appropriately placed. The window should also be prevented from becoming smaller than the image.	B1
Adding zoom	Finally in this phase, zooming of the image will need to be added. As this is a very important part of the editor, it <i>must</i> be added at this early stage, as it would be difficult to add later. Zooming contains the decision of which pixels to draw, and at what size.	B9
Detecting mouse position	Almost all brushes and features added later will in some way involve the mouse being clicked on the image. This makes it imperative that detecting mouse clicks is added at this early stage, no brushes can be added until this is complete.	A2

### 2.1.3 Usability

At the end of this design phase, the UI of the program will be very basic, as none of the tools will be added. The stakeholder input is not needed, as this will not reflect the final UI of the project.

#### 2.1.3.1 UI Design



### 2.1.3.2 UI Design Features

#### Canvas

The main canvas will need to be visible to the user. The canvas will display the image in its current state, depending on the zoom and positioning of the image at that time. This will take up the main proportion of the window, as it is what the user will interact with most.

#### Scroll Bars

The scroll bars have been added to allow the user to move around through their image, and there will be two for X and Y movement. The size of the bars will depend on the window size, and current zoom.

Scroll bars are suitable here due to allowing movement through a fixed range

#### Zoom Bar

The zoom bar will allow the user to change his current zoom of the image, and will start at 100% (as Criteria A1 dictates that zooming out is not needed) and will go to a maximum value.

### 2.1.4 Inputs, Processing, Outputs and Storage

Input	Processing	Output	Storage
Mouse click at position on canvas	Detect where the mouse was clicked on the canvas	A pixel of colour is set at that position on the canvas	Image data stored in RAM, the current zoom and viewing location
Change of the value of the zoom bar	Calculate new pixel size, new image position and redraw the desired pixels	The image at a different level of zoom	Image data stored in RAM, the previous zoom and viewing location
Change of the value of the scroll bars	Calculate which portion of the image must now be displayed	A different portion of the image	Image data stored in RAM, the current zoom and previous viewing location

The table is very small at this point in time due to the limited functionality of the program. At later points in development there will be more potential inputs.

## 2.1.5 Class Design

*Note. This class design has been redeveloped. The updated design can be found [here](#)*

### 2.1.5.1 Image



The image class will store the main properties of the image.

#### Properties

Property	Datatype	Justification
colours[,]	2D array of type 'Color'	An array of colours is what will be needed to store the property of each pixel on the grid. The array will make use of the existing 'Color' class in C#, so that I do not reinvent the wheel.
width	Integer	Stores the width of the current image.
height	Integer	Stores the height of the current image.
zoomSetting	<a href="#">ZoomSettings</a>	Stores the current settings describing the zoom of the image.

#### Methods

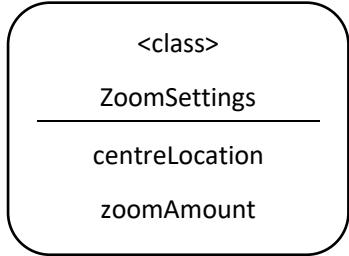
Method	Params	Return type	Justification
GetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get	Colour	Will get an existing colour from a specific point in the image, regardless of its size.
SetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get Colour, Color, colour of pixel to get	None	Will set a new colour onto the grid, similar to GetPixel.

GetDisplay- Image()	None	Image (C#)	Will return the C# base 'Image' class, for use in the displaying picture box.
------------------------	------	------------	--

### Constructor

```
class image {  
    Color[,] pixels  
    integer width  
    integer height  
    ZoomSetting zoomSettings  
    constructor(_width, _height) {  
        width = _width  
        height = _height  
        pixels = new array of Color(width,height)  
        // gives every pixel a default white colour  
        foreach Color in Pixels {  
            Color = White  
        }  
        zoomSettings = new ZoomSetting()  
    }  
}
```

### 2.1.5.2 ZoomSettings



ZoomSettings contains all the properties about the current zoom on the image. This is separate from the image class to enforce proper **encapsulation**, as the level of zoom is not directly tied to the image.

#### Properties

Property	Datatype	Justification
centre-Location	Point	Stores the location of the pixel in the middle of the zoom. The middle location is stored to make the zoom feel more natural when zooming in and out (the middle pixel remains the same)
zoom-Amount	Integer	Stores the amount that the image is currently zoomed in by. 1 = 1x = 100% zoom, 2 = 200% zoom etc.

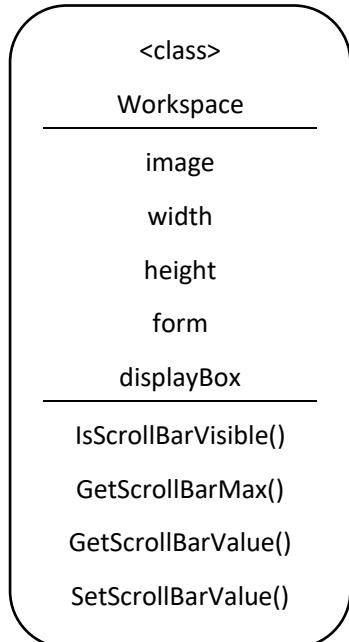
#### Constructor

```
class ZoomSettings {
    Point centreLocation
    Integer zoomAmount
    constructor(_centreLocation) {
        centreLocation = _centreLocation
        zoomAmount = 1
    }
}
```

A red dashed box highlights the line 'zoomAmount = 1'. A callout box with a pink background and black border contains the text: 'The default zoom will be none - 1x.'

### 2.1.5.3 WorkSpace

WorkSpace is a class that encapsulates everything about the current working area. It stores the width and height of the window it needs to display to, holds an image to display, and calculates the display properties for the scroll bar, which it will send to the [ZoomSettings](#) class



#### Properties

Property	Datatype	Justification
image	<a href="#">Image</a>	Stores the image that is currently being edited.
width	Integer	Stores the width of the current working area (not the width of the image).
height	Integer	Stores the height of the current working area.
form	Form	Links the workspace to a Windows Form, to allow for easy interaction
displayBox	PictureBox	Links the workspace to the Picture Box that it will use for displaying

#### Methods

Method	Params	Return type	Justification
IsScrollBar-Visible	axis, <a href="#">Axis</a> , the X or Y bar.	Boolean	This determines whether the selected scroll bar needs to be visible, if the workspace is larger than the image then no scroll bars are needed.
GetScroll-BarMax()	axis, <a href="#">Axis</a> , the X or Y bar.	Integer	Calculates what the maximum for the selected scroll bar needs to be set to.
GetScroll-BarValue()	axis, <a href="#">Axis</a> , the X or Y bar.	Integer	Calculates what the current value for the selected scroll bar needs to be set to.

SetScrollBarValue()	axis, <a href="#">Axis</a> , the X or Y bar. value, Integer, the value to set the scroll bar to.	None	Sets the value of the scroll on the selected X or Y axis. This employs <b>abstraction</b> as the lower classes have hidden the specifics of zooming, and all that is needed to change the zoom location is progress through a bar.
---------------------	---	------	--

### Constructor

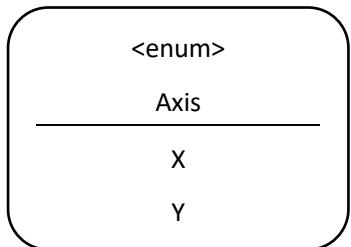
```
class WorkSpace {
    Image image
    Integer width
    Integer height
    Form form
    PictureBox displayBox

    constructor(_image, _form) {
        image = _image
        form = _form
        width = width of form
        height = height of form
        displayBox = new PictureBox
        [add displayBox to form]
    }
}
```

The 'Workspace' object contains the associated PictureBox, which it will add to the form.  
This means that the base form inputted will need to be blank

### 2.1.5.2 Axis

The scrollbar enum is a small set of constants for deciding whether to manipulate the X axis or the Y axis. This enum has no purpose by itself but makes the code more **readable** and **maintainable** by making it clear which axis is being manipulated.



Member	Justification
X	Denotes that the X axis has been selected.
Y	Denotes that the Y axis has been selected.

## 2.1.6 Algorithm Design

---

### Algorithm 2.1 Creating Image Class

As there is no program 'flow' in defining a class, there is no flowchart

Pseudocode

```
class image {  
    Color[,] pixels  
    integer width  
    integer height  
    ZoomSetting zoomSettings  
}
```

## Algorithm 2.2 Populating Image Class

To now populate the image class, a constructor has been added to fill in some values.

Pseudocode

```
class image {  
    Color[,] pixels  
    integer width  
    integer height  
    ZoomSetting zoomSettings  
    constructor(_width, _height) {  
        width = _width  
        height = _height  
        pixels = new array of Color(width,height)  
        // gives every pixel a default white colour  
        foreach Color in Pixels {  
            Color = White  
        }  
        zoomSettings = new ZoomSetting(middle of image)  
    }  
}
```

Unit Testing

	Test Data	Test Type	Expected Output	Description
<code>new Image(10,10)</code>	Valid	An image of size 10fpx, 10fpx	This tests if a normal image can be created	
<code>new Image(10,6)</code>	Valid	A 10fpx, 5fpx image	This tests if a non-rectangular image can be created	
<code>new Image(10,5)</code>	Valid	A 10fpx, 2fpx image	This tests when an odd dimension is entered	
<code>new Image(10,1)</code>	Extreme	A 10fpx, 1fpx image	This tests if a very short image can be created	
	Valid			
<code>new Image(1,10)</code>	Extreme	A 1fpx, 10fpx image	This tests if a very thin image can be created	
	Valid			
<code>new Image(1,1)</code>	Extreme	A 1fpx, 1fpx image	This tests when a very small image is created	
	Valid			
<code>new Image(10,0)</code>	Extreme	The parameters are rejected and no image is created	This tests if an image with no height is rejected	
	Invalid			

<code>new Image(0,10)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width is rejected
<code>new Image(0,0)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width or height is rejected
<code>new Image(10000,10)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a width which is much too large is rejected
<code>new Image(10,10000)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a height which is much too large is rejected
<code>new Image(-1,10)</code>	Invalid	The parameters are rejected and no image is created	This tests if a negative width is rejected.
<code>new Image(10,-1)</code>	Invalid	The parameters are rejected and no image is created	This tests if a negative height is rejected.
<code>new Image(10)</code>	Erraneous	The parameters are rejected and no image is created	This tests if an image missing a height is rejected.
<code>new Image("10","10")</code>	Erraneous	The parameters are rejected and no image is created	This tests if an image with invalid numbers is rejected

### Algorithm 2.3 Resizing Picture Box

As this algorithm is very linear, no flowchart diagram is needed

Pseudocode

```
class Image {  
    [...]  
    ResizePictureBox(box) {  
        // uses the width and height properties to set properties of the  
        // picture box  
        box.width = width  
        box.height = height  
        // this ensures the picture box can hold the pixels  
    }  
}
```

This ellipsis denotes previously designed code, showing that this code is part of the existing 'Image' class

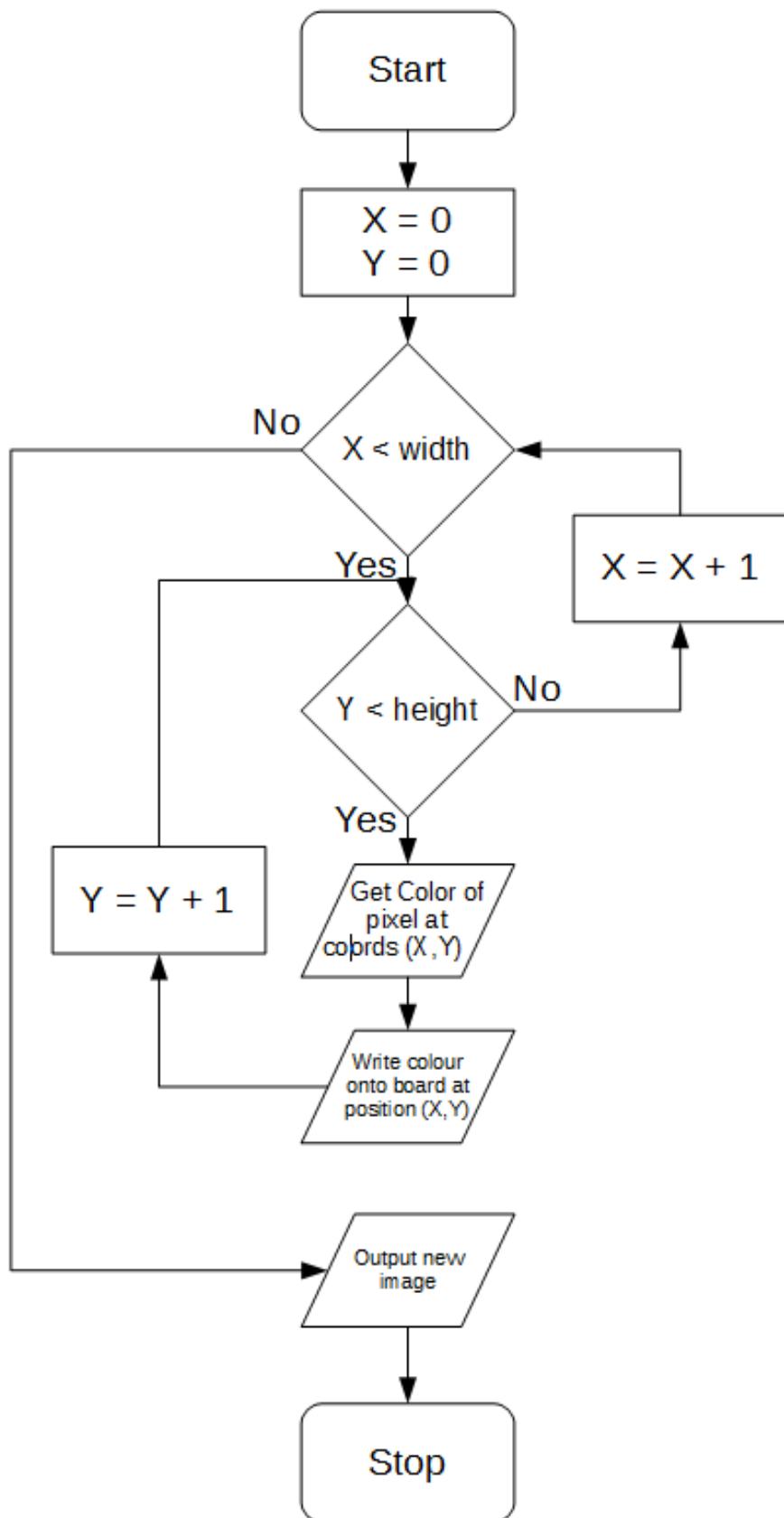
Unit Testing

Test Data	Test Type	Expected Output	Description
<i>Input Box(10,10)</i>	Valid	An image of size 10fpx, 10fpx	This tests if a normal image can be created
<i>Input Box(10,6)</i>	Valid	A 10fpx, 5fpx image	This tests if a non-rectangular image can be created
<i>Input Box(10,5)</i>	Valid	A 10fpx, 2fpx image	This tests when an odd dimension is entered
<i>Input Box(10,1)</i>	Extreme Valid	A 10fpx, 1fpx image	This tests if a very short image can be created
<i>Input Box(1,10)</i>	Extreme Valid	A 1fpx, 10fpx image	This tests if a very thin image can be created
<i>Input Box(1,1)</i>	Extreme Valid	A 1fpx, 1fpx image	This tests when a very small image is created
<i>Input Box(10,0)</i>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no height is rejected
<i>Input Box(0,10)</i>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width is rejected
<i>Input Box(0,0)</i>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width or height is rejected
<i>Input Box(10000,10)</i>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a width which is much too large is rejected
<i>Input Box(10,10000)</i>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a height which is much too large is rejected

<i>Input Box(-1,10)</i>	Invalid	The parameters are rejected and no image is created	This tests if a negative width is rejected.
<i>Input Box(10,-1)</i>	Invalid	The parameters are rejected and no image is created	This tests if a negative height is rejected.
<i>Input Box(10)</i>	Erraneous	The parameters are rejected and no image is created	This tests if an image missing a height is rejected.
<i>Input Box("10","10")</i>	Erraneous	The parameters are rejected and no image is created	This tests if an image with invalid numbers is rejected

## Algorithm 2.4 Displaying Pixels

Flowchart



Pseudocode

For the pseudocode of this algorithm, I have two options for outputting

```

DisplayToPictureBox(box) {
    // creates a 'Graphics' object for
    // drawing directly onto the picture box
    Graphics GFX = box.CreateGrapics()
    for x = 1 to width {
        for y = 1 to height {
            colour = pixels[x,y]
            GFX.SetPixel(x,y,colour)
        }
    }
}

```

```

GetDisplayImage() {
    // creates a C# 'image' object to
    // store the output image
    Drawing.Image image = new
    Drawing.Image(width,height)
    for x = 1 to width {
        for y = 1 to height {
            colour = pixels[x,y]
            Image.SetPixel(x,y,colour)
        }
    }
    RETURN image
}

```

I have decided to use algorithm B for my program, as using an algorithm that outputs a standard object that contains all the necessary info, and is compatible with most workspace objects, is much more desirable than writing directly onto the picture box.

This also means the image data is easier to store and transmit, as it will be outputted fully contained in an image object.

Algorithms 2.3 and 2.4 will be combined into a single combined function, which encompasses the entirety of displaying the image:

```
class Workspace {
    ...
    public Display {
        ResizePictureBox(displayBox)
        displayBox.DisplayImage = GetDisplayImage()
    }
}

private ResizePictureBox(box) {
    // uses the width and height properties to set properties of the
    // picture box
    box.width = width
    box.height = height
    // this ensures the picture box can hold the pixels
}

private GetDisplayImage() {
    // creates a C# 'image' object to store the output image
    Drawing.Image image = new Drawing.Image(width,height)
    for x = 1 to width {
        for y = 1 to height {
            colour = pixels[x,y]
            Image.SetPixel(x,y,colour)
        }
    }
}
```

The only public facing function is 'Display'. The end user will not need to see any other of the functions.

Image displaying will be part of the 'Workspace' class, as it encapsulates both generating the image and placing it in the correct box

## Algorithm 2.5 Setting Pixels at Runtime

### Pseudocode

```
class Image {  
    ...  
    SetPixel(x,y,colour) {  
        pixels[x,y] = colour  
        Display()  
    }  
}
```

All that is needed for this function is to call the existing `Display()` function, which will handle all of the image displaying.

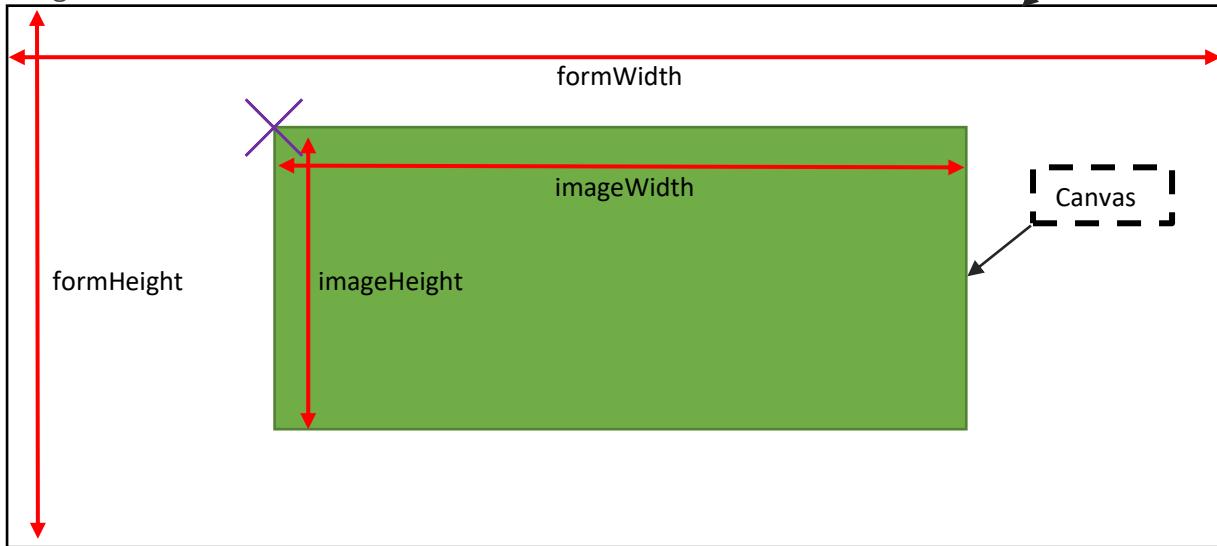
### Unit Testing

This assumes the image has a size of 10px by 10px

<i>Test Data</i>	<i>Test Type</i>	<i>Expected Output</i>	<i>Description</i>
<code>SetPixel(5,5,Black)</code>	Valid	A pixel near the middle of the image is set to black	This tests if a normal pixel can be set
<code>SetPixel(5,5,Green)</code>	Valid	A pixel near the middle of the image is set to yellow	This tests if a pixel can be set to other colours than black
<code>SetPixel(0,0,Black)</code>	Extreme Valid	A pixel in the top-left corner is set to black	This tests if the top-left corner can be set
<code>SetPixel(9,9,Black)</code>	Extreme Valid	A pixel in the bottom-right corner is set to black	This tests if the bottom-right corner can be set
<code>SetPixel(-1,0,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far left is rejected
<code>SetPixel(0,-1,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far up is rejected
<code>SetPixel(10,0,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far right is rejected
<code>SetPixel(0,10,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far down is rejected

## Algorithm 2.6 Determining PictureBox Location

Diagram



The purple **X** denotes what the location of the picture box must be. Looking at the diagram, it becomes clear that the X position of the **X** must be:



So this to find the X, the formula is  $\frac{1}{2}\text{formWidth} - \frac{1}{2}\text{imageWidth}$  or  $\frac{1}{2}(\text{formWidth} - \text{imageWidth})$

Pseudocode

```
class WorkSpace {  
    ...  
    RelocatePictureBox {  
        Integer X = (width - image.width) / 2  
        Integer Y = (height - image.height) / 2  
        displayBox.Location = new Location(X,Y)  
    }  
}
```

## Algorithm 2.7 Stopping window becoming too small

### Pseudocode

```
form.minimumWidth = image.width  
form.minimumHeight = image.height
```

This prevents the form from (at this point) ever being smaller than the image, as zooming is not yet implemented.

### Unit Testing

<i>Test Data</i>	<i>Test Type</i>	<i>Expected Output</i>	<i>Description</i>
<i>Form is started at normal size</i>	Valid	Image Displays in the middle of the form	This tests that the program can start and display the image
<i>Form is maximized</i>	Extreme Valid	Image displays in the middle of the large form	This tests if the image is displayed when the form is very large
<i>Form is minimized</i>	Extreme Valid	No image is displayed (as form is currently invisible)	This tests that the program can be minimized safely
<i>Form is resized to smallest possible</i>	Extreme Valid	The form cannot be made smaller than the image	This tests that the form can never be made smaller than the image
<i>Form is resized to minimum width maximum height</i>	Extreme Valid	A very thin form displays the image	This tests if a form with an extreme height but minimum width is accepted
<i>Form is resized to minimum height maximum width</i>	Extreme Valid	A very short form displays the image	This tests if a form with an extreme width but minimum height is accepted
<i>The form's size is rapidly changed.</i>	Extreme Valid	The image is very quickly moved around but remains centred	This tests that under a stress test the image is displayed correctly

## Algorithm 2.8 Deciding which pixels to draw

### Note

At this point in time, there are two sorts of 'pixel'. There is a pixel on the image file, and a pixel on the user's screen. This becomes a problem when zooming is applied, as 1 file pixel will translate to 4 displayed pixels (at 2x zoom).

To clear up confusion, there will be two terms used to refer to pixels:

- A 'file pixel' (fpx) refers to a pixel in the image file, stored in the original array of colours
- A 'display pixel' (dpx) refers to a pixel being displayed on the user's screen. All pixels up to this point have been display pixels. Note that display pixels are bound to the screen, so a display pixel of 0,0 could be a pixel anywhere on the image

Algorithms to switch between these two sorts of pixel are [here](#).

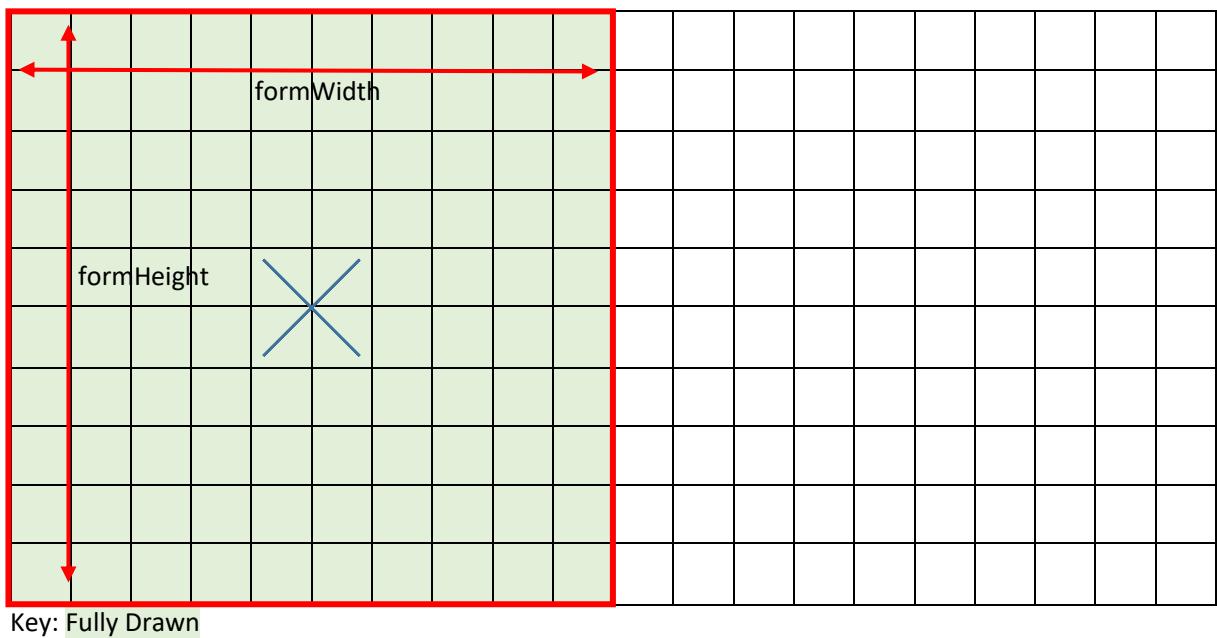
From this point on, algorithms involving zoom and zoom centring will be implemented.

### Algorithm 2.8A Converting between File Pixels & Display Pixels

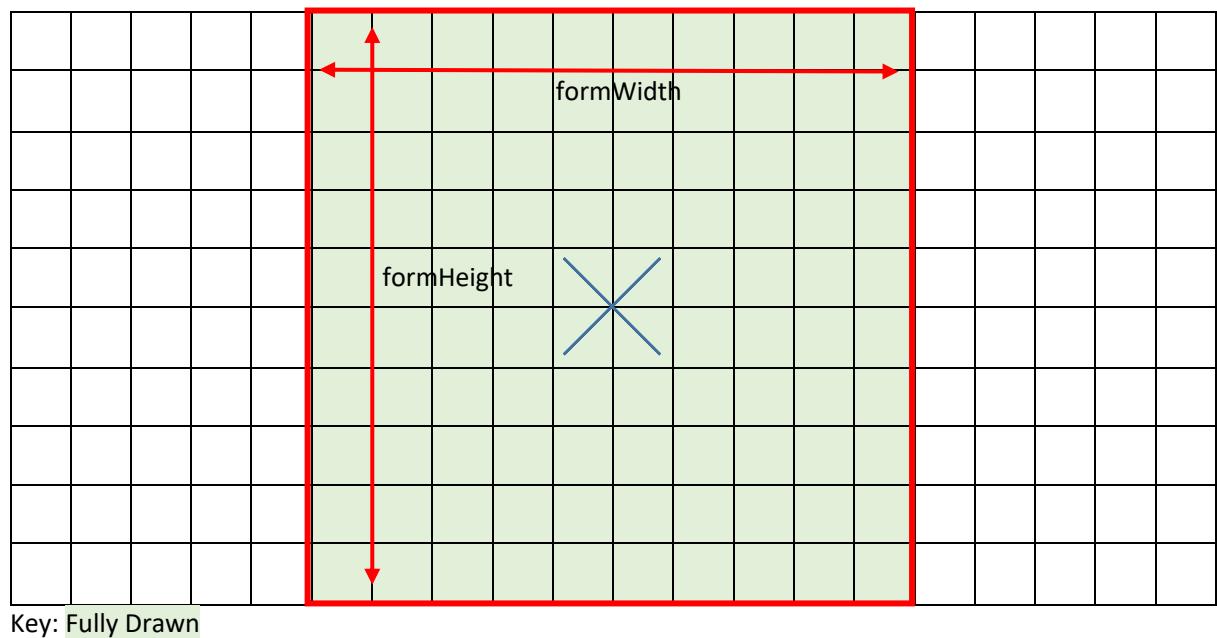
While making these algorithms, two important functions will need to be designed. An algorithm to convert from a File Points (a location in the file), to a display pixel (a pixel on the screen).

Assuming there is an image of 20fpx by 10fpx:

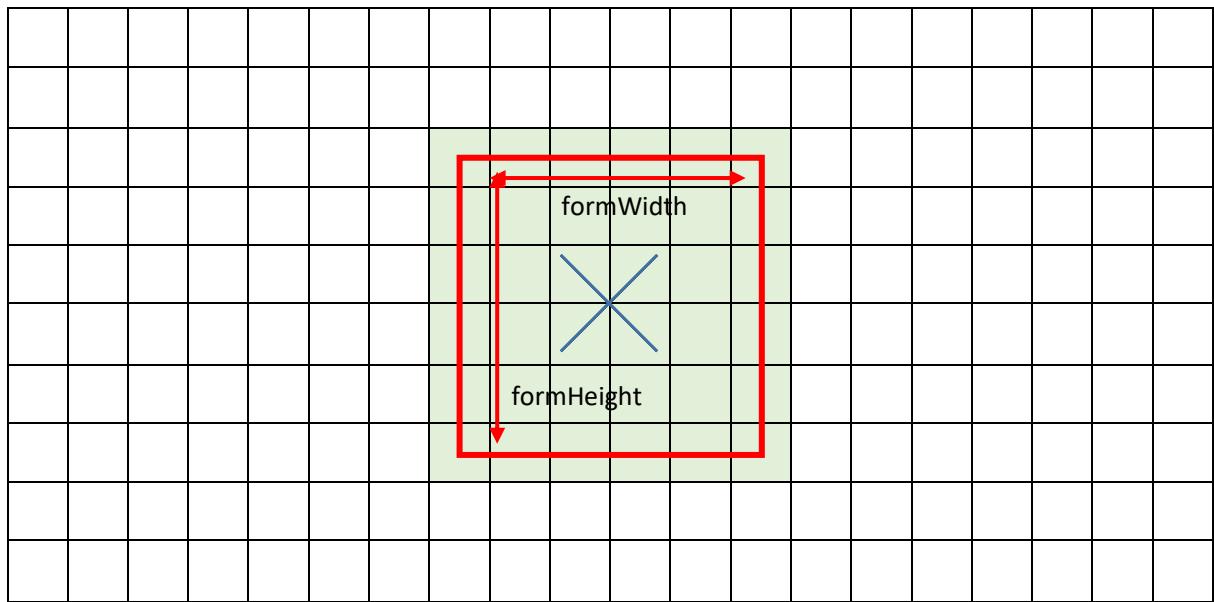
If the screen has a size of 10fpx $\times$ 10fpx, centred on the pixel at 5fpx,5fpx, then the zoomed area will be:



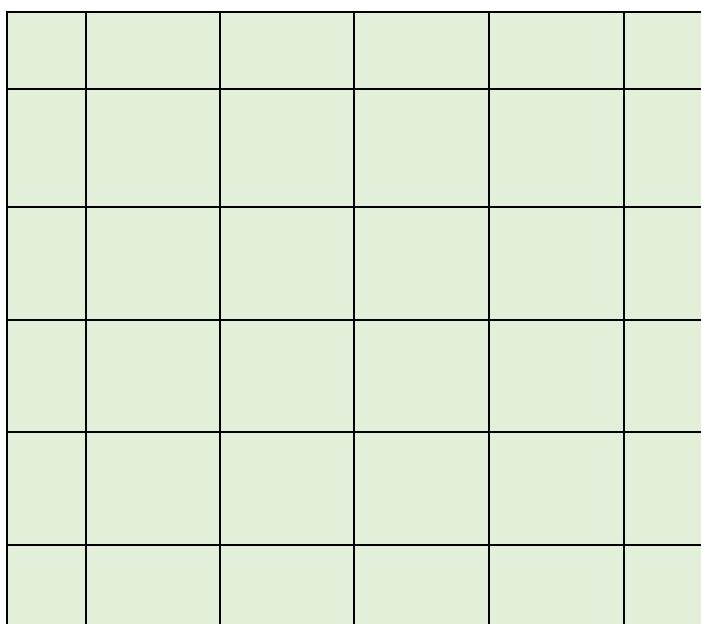
If the centre location is moved to 10,5, the zoomed area will now be:



However, if the zoom is doubled, then the resulting window will be:



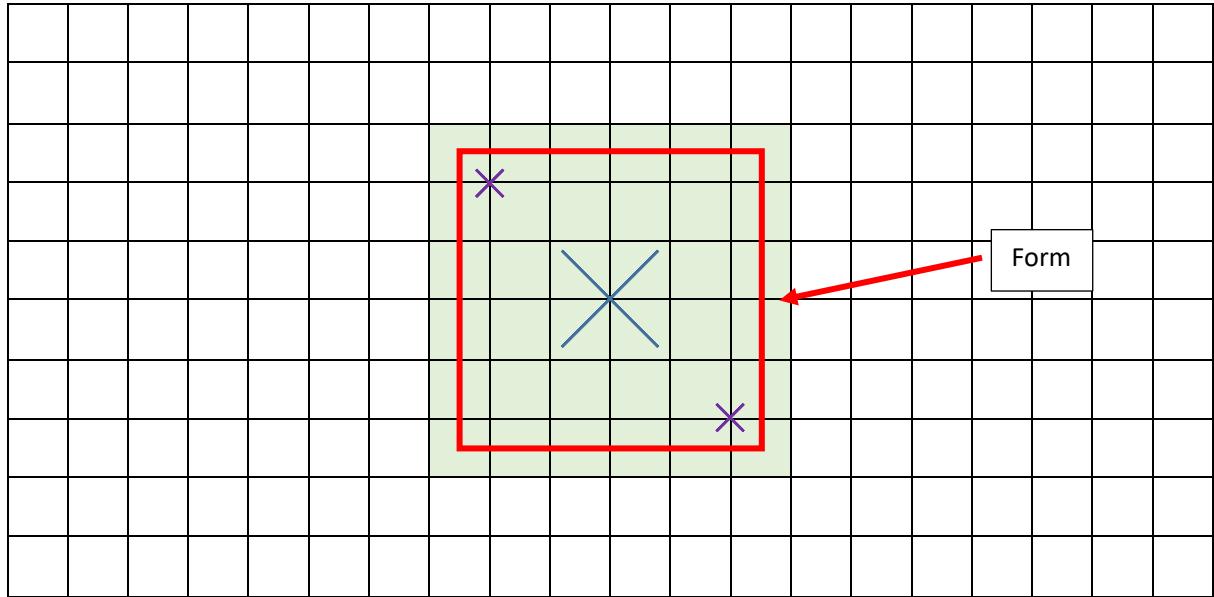
The user will, in this example, see this:



A green pixel represents a pixel that must be drawn to the screen

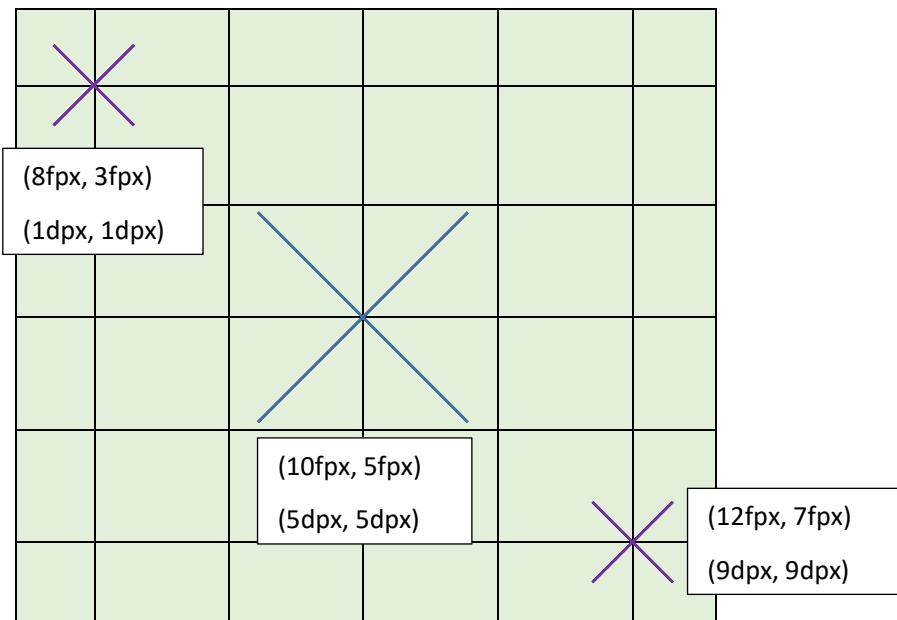
To design these algorithms, the outputs of a form in several positions will be considered.

In this case, the zoom centre is at (10fpx, 5fpx), at 2x zoom.



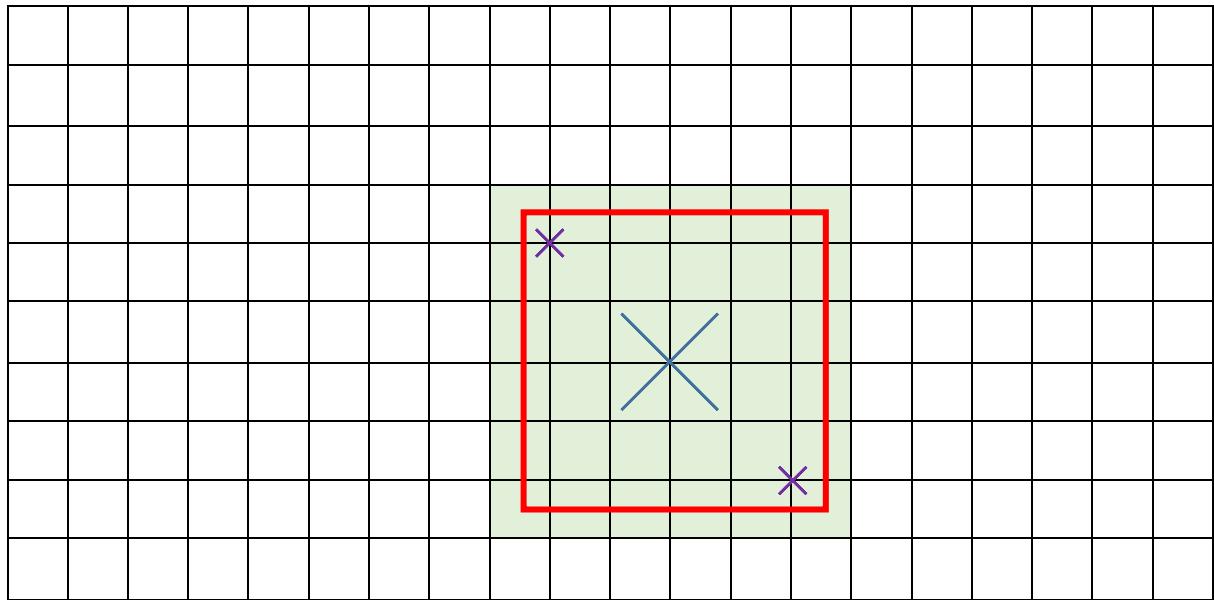
Thus, the form's view will be:

(Each square in this diagram is 2dpx by 2dpx)

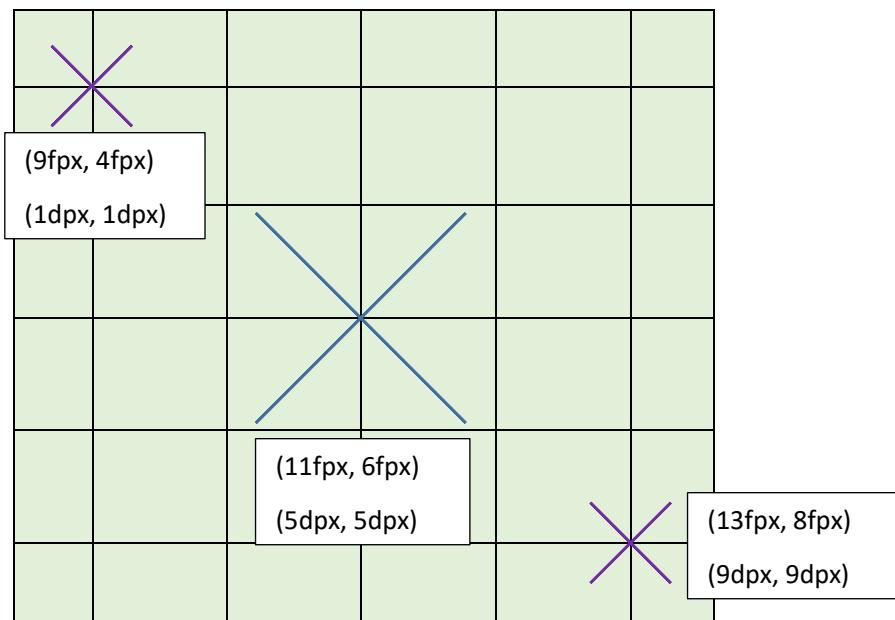


The expected display locations (in dpx) have been labelled, as well as their file positions (in fpx).

In another example, the centre location is moved to file location (11fpx, 6fpx)



Each square in this diagram is 2dpx by 2dpx



The expected display locations (in dpx) have been labelled, as well as their file positions (in fpx).

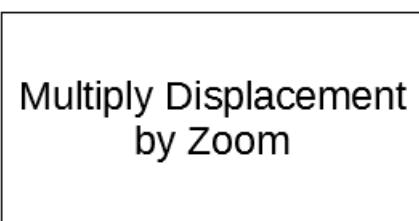
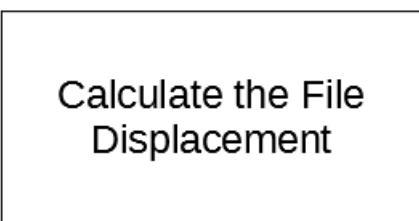
From these diagrams some conclusions can be drawn:

- The centre location X must always stay in the middle of the form.
- Only points close to the centre location are drawn on the diagram
- If the zoom is increased, fewer points are drawn.

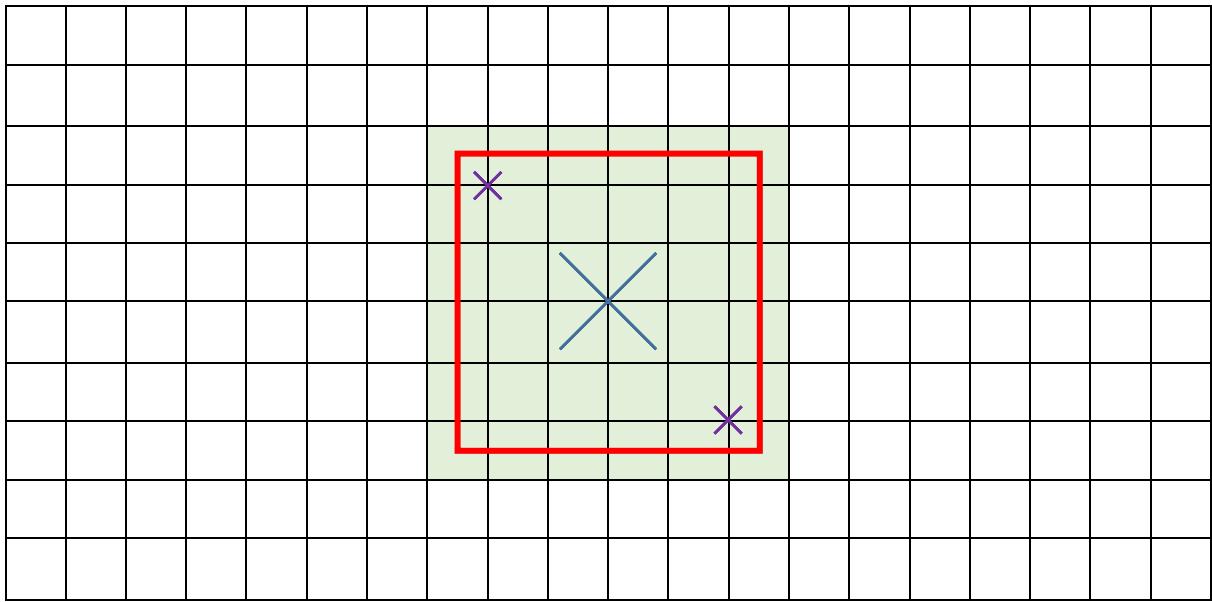
Considering this, algorithms can start to be planned:

File Pixels to Display Pixels

*Flowchart*

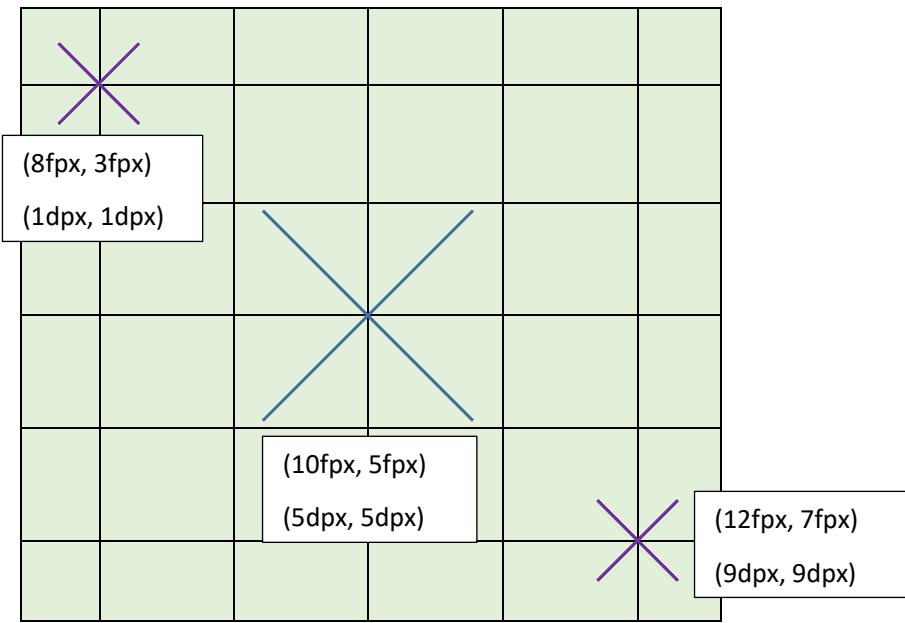


For an example, considering the first diagram again:



Thus, the form's view will be:

(Each square in this diagram is 2dpx by 2dpx)



Considering the top-left X.

File Displacement X = 8 - 10 = -2

File Displacement Y = 3 - 5 = -2

Multiplied by Zoom Displacement X = 2 \* -2 = -4

Multiplied by Zoom Displacement Y = 2 \* -2 = -4

Added to centre location X = 5 + -4 = 1

Added to centre location Y = 5 + -4 = 1 = Display Location of (1dpx, 1dpx)

These results are consistent with the predicted display location.

### *Pseudocode*

From this algorithm, the follow pseudocode can be produced:

```
FilePointToDisplayPoint(filePoint) {  
    displacementX = filePoint.X - centreFilePoint.X  
    displacementY = filePoint.Y - centreFilePoint.Y  
  
    displacementX = displacementX * zoom  
    displacementY = displacementY * zoom  
  
    newX = centreDisplayPoint.X + displacementX  
    newY = centreDisplayPoint.Y + displacementY  
  
    return new Point(newX, newY)  
}
```

### *Unit Testing*

This assumes the image has a size of 20fpx by 10fpx and the zoom centre is at (10fpx, 5fpx) and zoom is at 2x

<i>Test Data</i>	<i>Test Type</i>	<i>Expected Output</i>	<i>Description</i>
<i>DisplayPoint(10,5)</i>	Valid	(5,5)	This tests if a point at the same location as the zoom centre is returned as the centre location of the zoom centre
<i>DisplayPoint(8,3)</i>	Valid	(1,1)	This tests if a point to the top left of the centre location is positioned appropriately in the image
<i>DisplayPoint(12,7)</i>	Valid	(9,9)	This tests if a point to the bottom right of the centre location is positioned appropriately in the image
<i>DisplayPoint(7,2)</i>	Extreme Valid	(-1,-1)	This tests if negative coords will be outputted. Whilst this may seem like a glitch this is necessary for some of the displaying code to display half size pixels.
<i>DisplayPoint(13,8)</i>	Extreme Valid	(11,11)	This tests if coords larger than display form will be outputted. This is also necessary for some of the displaying code to display half size pixels.
<i>DisplayPoint(0,0)</i>	Extreme Valid	(-15,-5)	This tests if a point in the top right corner has a relative display point calculated

<i>DisplayPoint(19,9)</i>	Extreme Valid	(23,13)	This tests if a point in the bottom left corner has a relative display point calculated
<i>DisplayPoint(-1,-1)</i>	Extreme Invalid	Throws out of bounds error	This tests if a point that is out of the bounds of the grid is not accepted
<i>DisplayPoint(20,10)</i>	Extreme Invalid	Throws out of bounds error	This tests if a point that is out of the bounds of the grid is not accepted

## Display Pixels to File Pixels

In order to construct this algorithm, it becomes clear that the inverse of the [File Pixels to Display Pixels](#) algorithm should be employed.

### Flowchart

This is the same steps as the File Pixels to Display Pixels [flowchart](#), but in reverse order.

- Find the displacement between the display location and the centre coord (in terms of X and Y) by subtracting that point's file coords from the coords of the centre location.
- Divide the displacements by the current zoom
- Add the displacements to the file location of the centre coord.

### Pseudocode

The pseudocode follow a similar format as the previous pseudocode:

```
FilePointToDisplayPoint(displayPoint) {  
    displacementX = displayPoint.X - centreDisplayPoint.X  
    displacementY = displayPoint.Y - centreDisplayPoint.Y  
  
    displacementX = displacementX / zoom  
    displacementY = displacementY / zoom  
  
    newX = displacementX + centreFilePoint.X  
    newY = displacementY + centreFilePoint.Y  
  
    return new Point(newX, newY)  
}
```

This will use an integer division, so only whole numbers (rounded down) will be returned from this.

## Unit Testing

Test Data	Test Type	Expected Output	Description
<i>FilePoint(5,5)</i>	Valid	(10,5)	This tests if a point at the same location as the zoom centre is returned as the centre location of the zoom centre
<i>FilePoint(1,1)</i>	Valid	(8,3)	This tests if a location near the top of the display box is returned as above the centre location
<i>FilePoint(12,7)</i>	Valid	(9,9)	This tests if a location near the bottom of the display box is returned as below the centre location
<i>FilePoint(-1,-1)</i>	Extreme Valid	(7,2)	This tests if a file point away from the top left of the image is returned as its appropriate file point
<i>FilePoint(11,11)</i>	Extreme Valid	(13,8)	This tests if a file point away from the bottom right of the image is returned as its appropriate file point
<i>FilePoint(-15,-5)</i>	Extreme Valid	(0,0)	This tests if a point at the top left of the file will be returned as such
<i>FilePoint(23,13)</i>	Extreme Valid	(19,9)	This tests if a point at the bottom right of the file will be returned as such
<i>FilePoint(6,6)</i>	Valid	(10,5)	This tests if a point that is halfway across a pixel is rounded down to its nearest file location
<i>FilePoint(4,4)</i>	Valid	(9,4)	This tests the same as above
<i>FilePoint(-17,-7)</i>	Extreme Invalid	This point is rejected as it would output (-1,-1)	This tests if a Display Point that would output an invalid file point is rejected
<i>FilePoint(25,15)</i>	Extreme Invalid	This point is rejected as it would output (10,5)	This tests if a Display Point that would output an invalid file point is rejected

## Extra Note

These two functions now make a complete loop allowing conversions to and from file pixels and display pixels.

However there will be some information loss, as the [Display Pixel to File Pixel conversion code](#) will round display pixels to the nearest pixel, this is because of this snippet of the pseudocode:

```
displacementX = displacementX / zoom  
displacementY = displacementY / zoom
```

This occurs as **integer division** is used instead of floating point division. The decision to use integer division was made so that File Pixels followed the **simple rule** that File Pixels must always be whole numbers (as well as Display Pixels). This is done to:

- **Reduce confusion.** The concept of 'halfway through' a File Pixel isn't natural. File Pixels are usually understood to be the smallest indivisible component of the image, so there cannot be half a pixel.
- **Reduce code error.** Keeping File Pixels as integers means that all references to the same file pixel are equivalent. This avoids the potential error of checks failing since  $10.0 \neq 10.5$ .
- **Remove unnecessary information.** The knowledge that the user has clicked halfway across a File Pixel isn't useful, as no edits can be made to half pixels, so can be safely ignored.

Why differentiate between pixel types?

White pixels do not need to be drawn, so it is useful to be able to ignore them. When the image is zoomed in, many pixels do not need to be drawn at all. Ignoring them will significantly increase performance, especially with larger images at high zoom.

**Decision:** Which algorithm should be used for determining status of pixels?

In this case, there are two potential sorts of algorithm to use when telling the renderer which pixels are Visible and which are not

Potential Algorithm A: Checking whether a given pixel is Visible

Would be given the location of a pixel and the current zoom settings, and would return whether that pixel is Visible or not. The renderer would iterate over every pixel, making a decision based on the return of this function on each pixel.

If White → Do nothing

If Visible → Draw that pixel

Advantages:

- Good for checking one individual pixel

Disadvantages

- In bulk, may perform the same calculation many times, not very efficient

Potential Algorithm B: Returning a list of all Visible Pixels

Would, from the current zoom settings, return a list of all Visible pixels. A list of white pixels is not necessary (nothing needs to be done with white pixels, they are not visible)

Advantages:

- Means the calculations to determine which pixels are visible is only done **once**. This significantly reduces the overheads that may be caused by doing the same sums repeatedly.
- It is easy to iterate over each set of pixels with a different algorithm once they have been separated.

Disadvantages

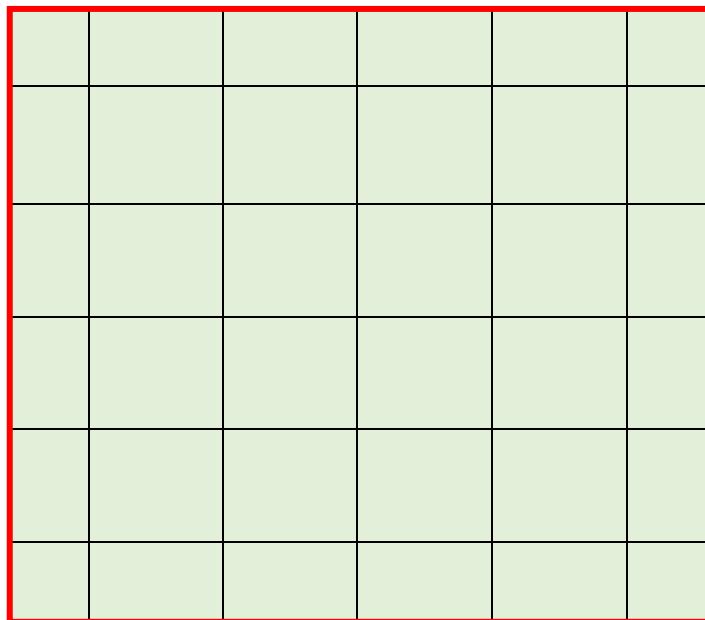
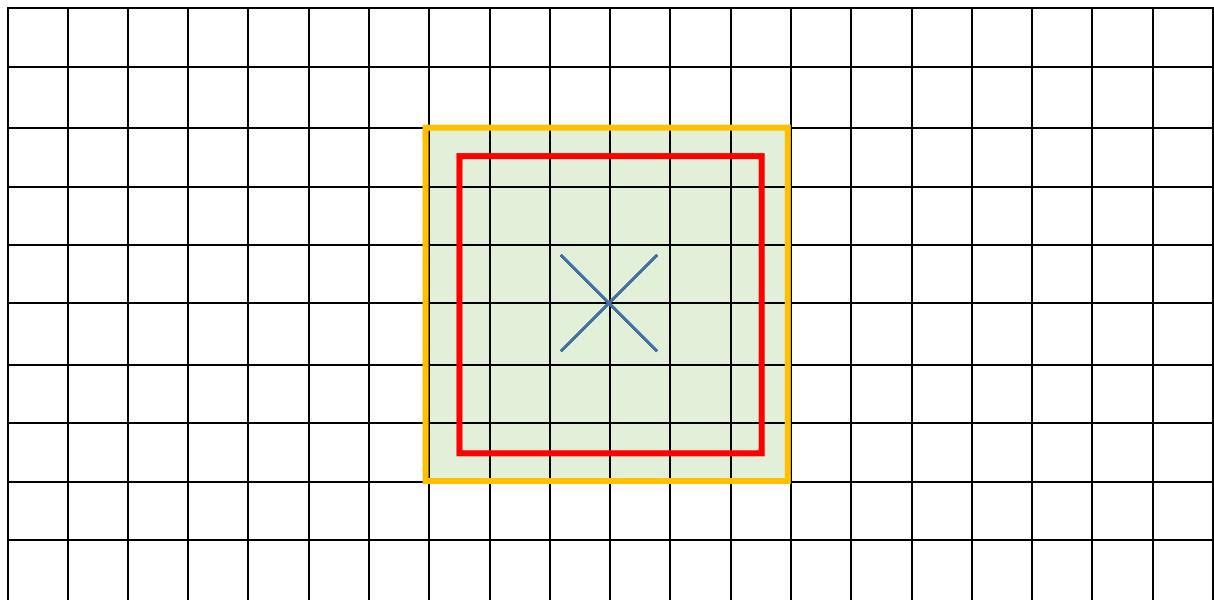
- Takes a long time to calculate whether one individual pixel is Visible or not, as you'd had to check the entire list to see whether it belongs there.

Decision

In this case, I have opted for **algorithm B**, as the image may be very large, the less complex scalability of algorithm B will suit the program much more. It doesn't matter that it is not easy to check an individual pixel as when rendering it is more important to draw as many pixels as possible.

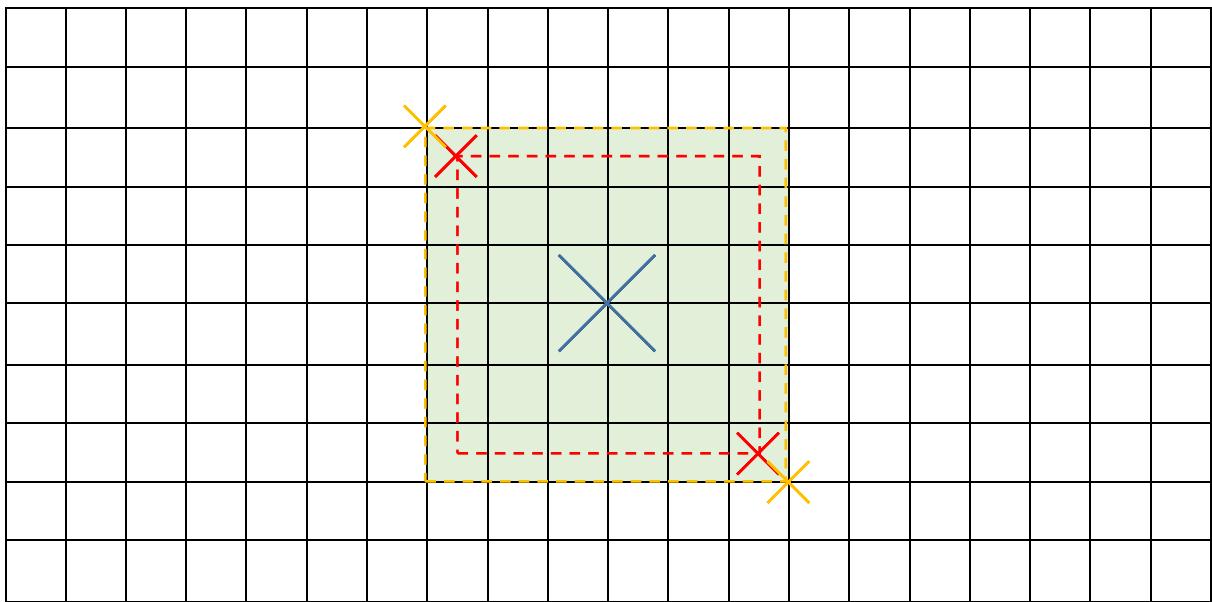
### Algorithm 2.8B Determining borders of pixels

From our illustrations before, two borders can be drawn. A red border to denote the edge of what the user can see, and an orange border to denote the range of pixels that must be drawn.



The Red border denotes where the drawing must stop.

By abstracting the rectangles into two points, this diagram can be drawn:



The advantage of these crosses is because it means that if we want to check whether a pixel needs to be drawn or not, all we need to do is check whether it is inbetween the two orange crosses.

#### Determining Location of Orange Crosses

The centre of this rectangle must be the centre location of the zoom. This implies that calculations must be done relative to the centre location.

In the X Axis firstly, to figure out how many pixels the current form could display in that axis, the program can divide the form size by the display size of one pixel (which is the same as the zoom), and then rounded down. This gives us the amount of file pixels the form can display.

In this example the process would be  $10 / 2$  (display form is 10dpx across and the current zoom is 2x).

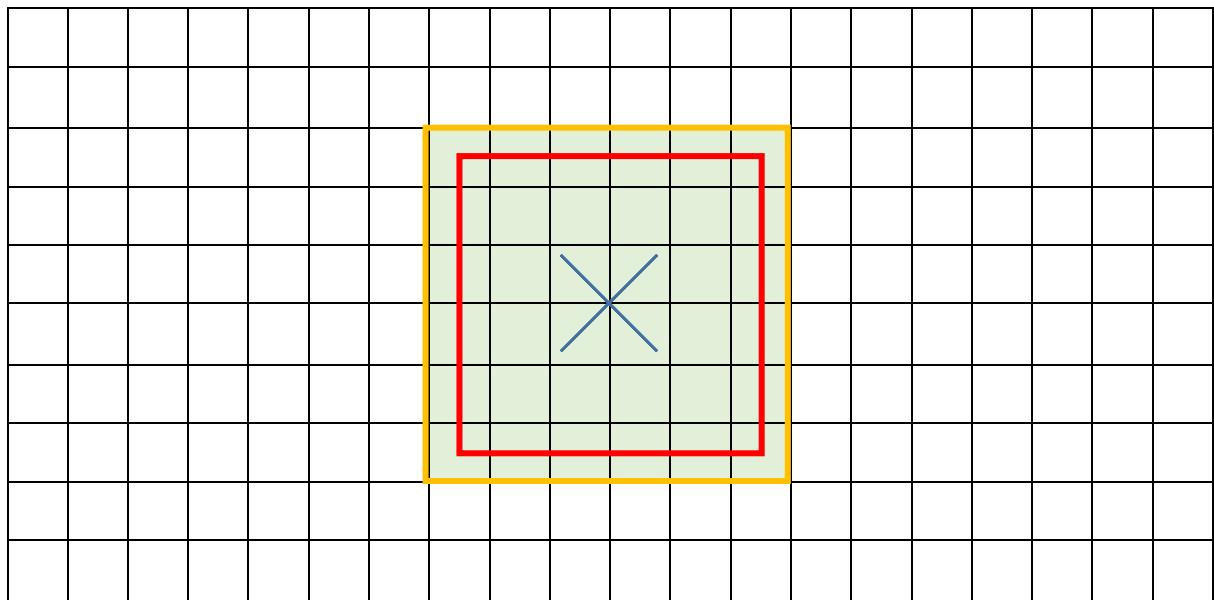
However, if this number is a whole number (for example 5), this can pose a problem if we try to put 5 pixels on each side of the centre location, as 5 cannot be split evenly into two. To find the amount of pixels to display on each side of the centre location, the amount of file pixels must be divided by 2 again, and then rounded **up**. (This is to make sure we draw as many as is needed). In this case the output would be 3, which is the desired result for the example above. (There are 3 visible pixels on each side of the cross).

Thus to find the file coords of Orange Crosses (depending on whether you + or -)

- The X is  $\text{centreloc.X} \pm \text{Ceiling}(\text{Floor}(\text{form.width}/\text{zoom}) / 2)$ 
  - The  $\pm$  depends on whether we are looking for the left or right cross
  - In the above example this would be  $10 + \text{Ceiling}(\text{Floor}(10/2)/2) = 10 + \text{Ceiling}(5)/2 = 10 + \text{Ceiling}(2.5) = 10 + 3 = 13$ .
- The X is  $\text{centreloc.Y} \pm \text{Ceiling}(\text{Floor}(\text{form.height}/\text{zoom}) / 2)$ 
  - The  $\pm$  depends on whether we are looking for the left or right cross
  - In the above example this would be  $5 + \text{Ceiling}(\text{Floor}(10/2)/2) = 5 + \text{Ceiling}(5)/2 = 5 + \text{Ceiling}(2.5) = 5 + 3 = 8$ .

### Algorithm 2.8C Finding Green pixels

To find the location of the green pixels, all that is needed is to find the pixels captured in the orange rectangle.



### Pseudocode

```
class Workspace {  
    ...  
    GetGreenPixels {  
        fileTopLeftPoint = orangeRectangle.topLeftCorner  
        fileBottomRightPoint = orangeRectangle.bottomRightCorner  
  
        for x = fileTopLeftPoint.X to fileBottomRightPoint.X  
            for y = fileTopLeftPoint.Y to fileBottomRightPoint.Y  
                DrawGreenPixel(x,y)  
            next y  
        next x  
    }  
}
```

This algorithm was designed above.

## Algorithm 2.9 Draw Zoomed Part

To draw Green Pixels, the operation can be done very quickly using the in-built Draw Rectangle tool in C#, and only Visible squares will need to be drawn.

Pseudocode

```
DrawGreenPixel(x,y) {  
    displayPoint = FilePointToDisplayPoint\(x,y\)  
    DrawRectangle(displayPoint.X,displayPoint.Y,zoom,zoom,colours[x,y])  
}
```

The zoom is used twice here as this is the width and height of the rectangle.

Another advantage of the in-built Draw Rectangle tool is that it is resistant to potentially bad inputs. As some Visible pixels location may start past the boundary of the display window, this could result in a **negative** Display Pixel location being inputted, or a Display Pixel location that is larger than the dimensions of the form. The Draw Rectangle tool handles these and will not draw pixels that it cannot display.

Unit Testing

Test Data	Test Type	Expected Output	Description
<i>Form is started at normal size</i>	Valid	Image Displays in the middle of the form	This tests that the program can start and display the image
<i>Form is maximized</i>	Extreme Valid	Image displays in the middle of the large form	This tests if the image is displayed when the form is very large
<i>Form is minimized</i>	Extreme Valid	No image is displayed (as form is currently invisible)	This tests that the program can be minimized safely
<i>Form is resized to smallest possible</i>	Extreme Valid	The form cannot be made smaller than the image	This tests that the form can never be made smaller than the image
<i>Form is resized to minimum width maximum height</i>	Extreme Valid	A very thin form displays the image	This tests if a form with an extreme height but minimum width is accepted
<i>Form is resized to minimum height maximum width</i>	Extreme Valid	A very short form displays the image	This tests if a form with an extreme width but minimum height is accepted
<i>The form's size is rapidly changed.</i>	Extreme Valid	The image is very quickly moved around but remains centred	This tests that under a stress test the image is displayed correctly

### Algorithm 2.10 Determining Bar Size

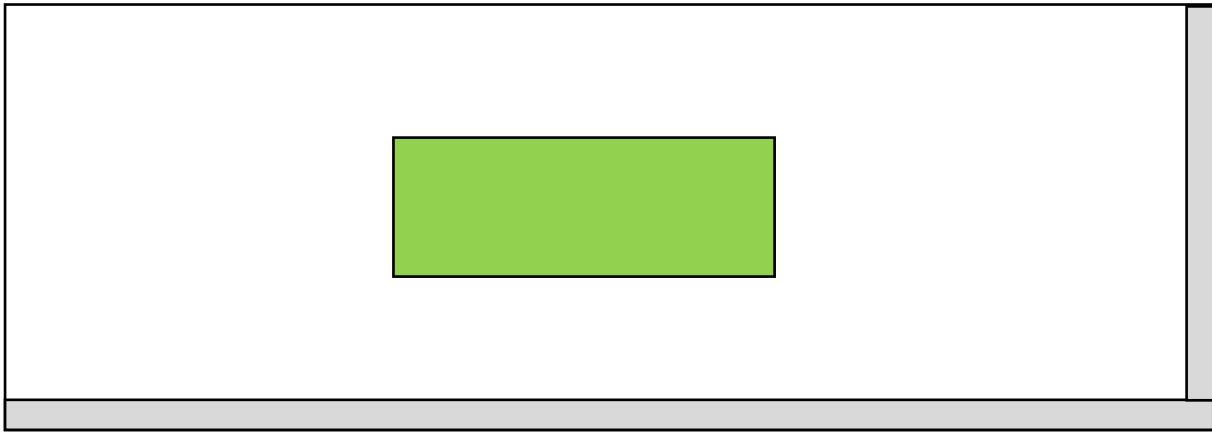
Now that the zooming has been implemented, there now needs to be a way to manipulate the zoom at runtime. As described on the original GUI design, the zoom will be manipulated using two bars at the edges of the screen:



Algorithm 2.10A Determining whether bars are visible or not.

The first consideration must be made to determine whether the scroll bars need to be shown. In examples where the image (with zoom applied) is smaller than the viewing screen, the scroll bars are not needed as the entire image is viewable at once. This means it must be possible to check whether the scroll bars need to be displayed, before attempting to display them.

For example, in this scenario, where the green represents the viewable image, there is obviously no need to put scroll bars in; there is nothing to scroll through.



Conversely, if the image is much more zoomed in, then the scroll bars become needed to help select which part of the image to look at:



The total size of the image can be calculated by multiplying the file size of the image in that axis (X or Y) by the zoom. If this is larger than the display form, the corresponding bar needs to be displayed.

## Pseudocode

```
IsBarVisible(Axis axis) {
    imageSizeInAxis = image.getFileSizeInAxis(axis) * zoom
    IF displayForm.getDisplaySizeInAxis(axis) > imageSizeInAxis {
        RETURN true
    ELSE
        RETURN false
    END IF
}
```

This code made reference to the function `getFileSizeInAxis(axis)`. This is a function that would return the size of the file in that axis (the axis being X or Y).

## Unit Testing

<i>Test Data</i>	<i>Test Type</i>	<i>Expected Output</i>	<i>Description</i>
<i>Form is started</i>	Valid	The form is started as the same size as the image and no bars are visible	This makes sure that form starts in an unzoomed way
<i>Form width is reduced to less than image</i>	Extreme	The horizontal scroll bar becomes visible	This makes sure that the scroll bar becomes visible
<i>Form width is then increased</i>	Valid	The horizontal scroll bar disappears	This makes sure that the scroll bar then becomes invisible
<i>Form height is reduced to less than image</i>	Extreme	The vertical scroll bar becomes visible	This makes sure that the scroll bar becomes visible
<i>Form height is then increased</i>	Valid	The vertical scroll bar disappears	This makes sure that the scroll bar then becomes invisible
<i>Form height and width is both reduced</i>	Extreme	Both scroll bars becomes visible	This makes sure both bars can be visible at the same time
<i>Form height is then increased</i>	Valid	The vertical scroll bar becomes invisible but horizontal scroll bar remains	This tests than one scroll bar can disappear but the other remains
<i>Form width is then increased</i>	Extreme	The horizontal scroll bar then becomes invisible	This tests than one scroll bar can disappear but the other remains

### Algorithm 2.10B Determining Bar Size

The size of the bar is determined by its defined width, and its maximum (and minimum) value. The minimum value will always be 1 (the top of the image).

In this GUI example, if the window was only able to display half of the image, the bars should look like:



Or, in specific numbers, if the image's size was 20dpx by 10dpx, but the display form was only 10dpx by 5dpx (half on each axis), this is what the result from the bars should be.

This means the bar should take up  $\frac{1}{2}$  of its available space. As  $10\text{dpx} / 20\text{dpx} = \frac{1}{2}$

However, an **easier implementation** for this sort of bar would be to (in the X axis) set the Maximum of the bar to the Display Size of the image, and set the Bar's width to the Display Size of the form. This means that the bar's display code will automatically resize the bar to its appropriate width, making the code much simpler.

Thus the pseudocode is:

```
SetupBarSize(progressBar, Axis) {
    progressBar.Maximum = image.getDisplaySizeInAxis(Axis)
    progressBar.BarSize = displayForm.getDisplaySizeInAxis(Axis)
}
```

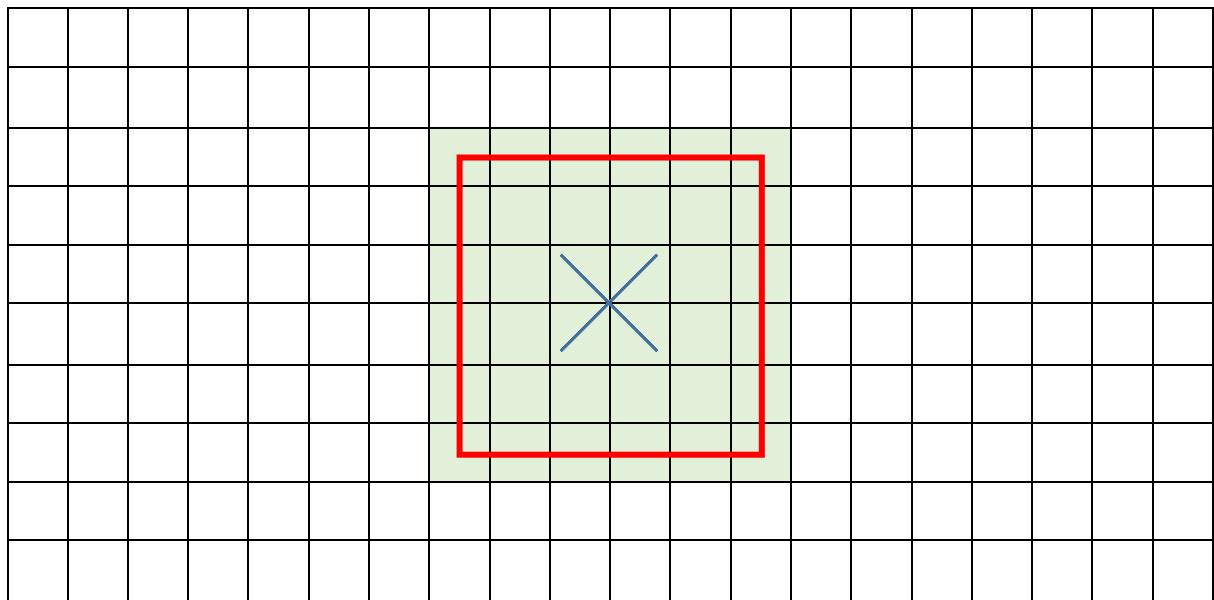
It's important here to set the maximum before the bar's size, or else the bar might be given a large size than the maximum – causing an error

### Algorithm 2.11 Interpreting Bar Location

This algorithm consists of two parts. Since the portion of the image that the user sees is entirely defined by the [centre location](#), there will need to be a way to determine the bar location from the current centre location. First, the total number of possible locations will need to be determined.

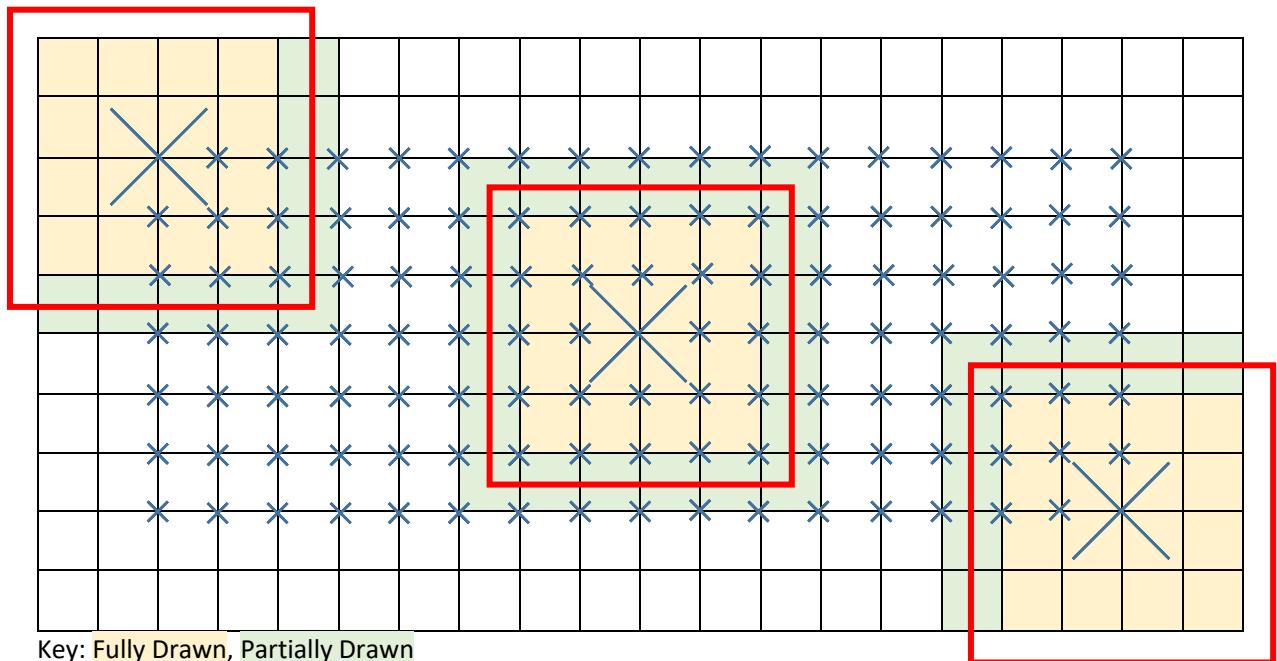
Algorithm 2.11A Determining possible centre locations

Coming back to the prior example of the window:



To recap, the image size is 20dpx by 10dpx, and the display window has dimensions of 10dpx by 5dpx.

In this example, there are many potential places where the centre could be placed, illustrated here:



Key: Fully Drawn, Partially Drawn

In this diagram a second shading has been introduced, yellow. This denotes fully drawn pixels. It is necessary that every pixel is fully viewable, so this yellow rectangle must be able to touch all pixels.

However, it is not necessary or useful to place the centre location any closer to the edge of the image, as it does show the user any more information. Or, to sum up:

**Scrolling in a direction should only be possible if scrolling in that direction reveals pixels that were not visible before.**

Thus, the crosses in the diagram denote all the possible valid locations that the centre locations can be placed.

The amount of crosses in each X axis is 17, whilst the theoretical maximum number of crosses is 21 (one on each vertical line). This means we need to calculate that 4 crosses need to be taken away. It is not necessary to know from where the crosses are taken, as they will always be taken equally from each side (in this example two crosses are taken away from each side). This means that the output of our sum must **always be a multiple of 2**.

The width of the Yellow Rectangle above will always be equal to the amount of crosses taken away, so it then becomes necessary to find the dimensions of the yellow rectangle. The algorithm for this is nearly identical to the [algorithm for finding the green rectangle](#), except it is rounded down at the end rather than rounded up.

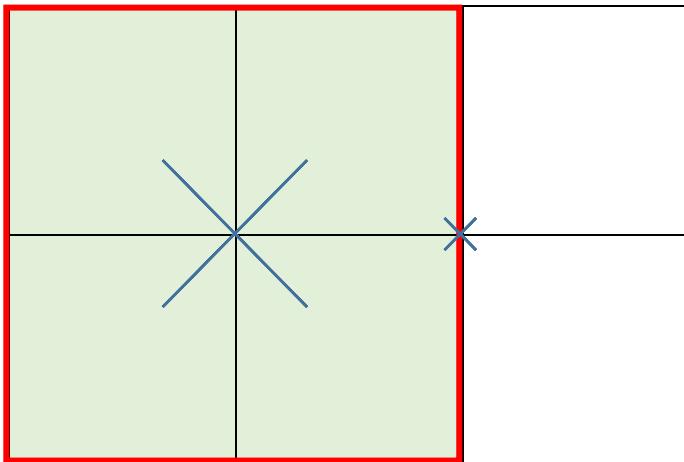
Pseudocode

```
DetermineCrossesInAxis(Axis axis) {  
    MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)*2  
    MaxCrossesInAxis = image.getSizeInAxis(axis)  
    return MaxCrossesInAxis - MissingCrossesInAxis  
}
```

### Algorithm 2.11B Upgrading scroll bar

Now that the potential amount of centre locations has been determined, there is an issue with the scroll bar's size code, as the calculations to determine its size become more complex.

Consider this small 3fpx by 2fpx image, at 1x zoom and a 2dpx by 2dpx view window with a centre location of 1fpx, 1fpx:



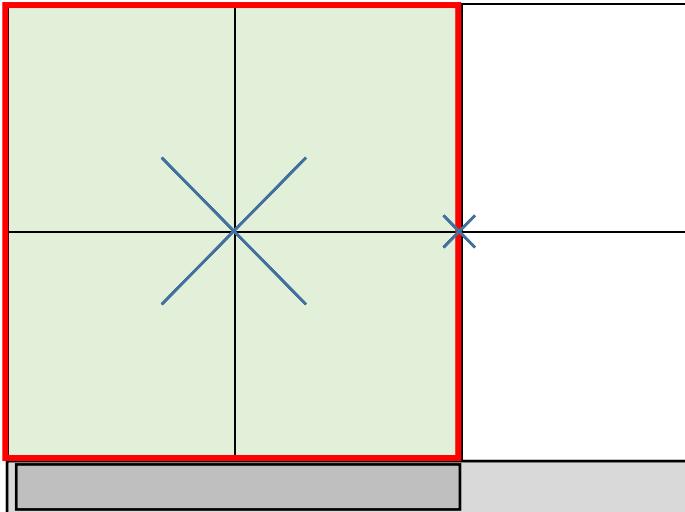
There is one other potential centre location (at 2fpx, 1fpx), and so the horizontal scroll bar should have two potential settings. However, there are multiple potential scroll bars that have two potential settings:

-  & 
-  & 
-  & 

Which one should be selected?

In this case, since the viewer is display  $\frac{2}{3}$  of the image, the bar should take up two thirds of its scroll bar.

To calculate the width, since the scroll bar's size is designed to represent the size of the display form, they can be aligned like so:



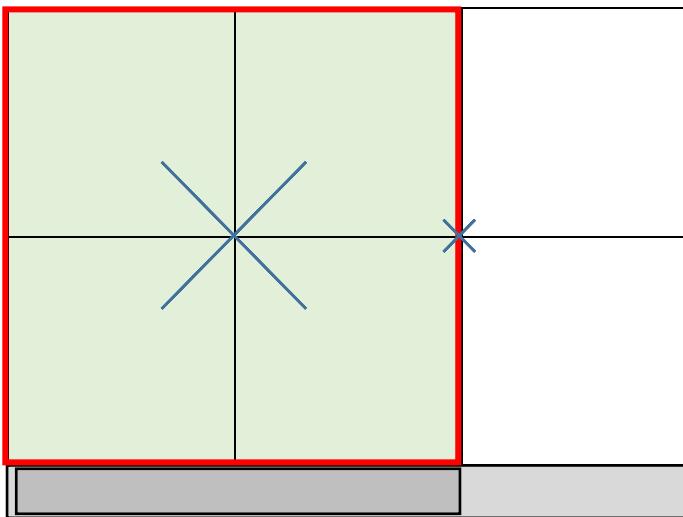
From this it becomes clear that the width of the bar must be equal to the width of the display form (which is equal to the amount of missing crosses), and have 3 potential values (where 1 is taken up by the width of the scroll bar).

Pseudocode

```
UpgradedSetupBarSize(progressBar, Axis) {  
    MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)*2  
    progressBar.barSize = MissingCrossesInAxis  
    progressBar.max = image.getFileWidth(Axis)  
}
```

### Algorithm 2.11C Determining Centre Location from Bar Value

The bar is now limited for how many locations it can be present in. In the above example there are only two potential positions:



Internally the bar has a value of 0 (shown) and 1 (when moved to the right).

So to find the needed centre location in that axis, all that is needed is to add the bar's value to the location of the first cross.

Pseudocode

```
GetCentreLocationFromBar(progressBar bar, Axis axis) {  
    firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)  
    RETURN bar.value + firstCrossPosition  
}
```

## Algorithm 2.11D Determining Bar Value from Centre Location

In some scenarios (such as when originally creating or resizing the display window) it may become necessary to be able to determine what the value of the scroll bar should be from the centre location (and other settings).

To do this, we can use the inverse of the [previous algorithm](#), where after determining the position of the first cross, we subtract it from the current centre location to find out where the bar should be.

### Pseudocode

```
SetBarFromCentreLocation(progressBar bar, Axis axis) {  
    firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)  
    bar.value = centreLocation.getSizeInAxis(axis) - firstCrossPosition  
}
```

## Unit Testing

In this case, image size is 20fpx by 10fpx and zoom centre is at 10fpx, 5fpx

<i>Test Data</i>	<i>Test Type</i>	<i>Expected Output</i>	<i>Description</i>
<i>Form is resized to half the size of image</i>	Valid	Bar is half size of bounds and in its centre position	This tests if the bar can display normally
<i>Form is resized to its smallest position</i>	Extreme Valid	Bar is small but still central.	This tests if the centre location remains in its position when the window size is reduced
<i>Form is resized back to half size of image</i>	Valid	Bar is half size of bounds and in its centre position	This tests if the centre location remains in its position when the window size is increased
<i>Horizontal Scroll bar is moved to far left</i>	Extreme Valid	The far left of the image is displayed, but no more	This tests that the far left of the image can be displayed and also that it stops there
<i>Horizontal Scroll bar is moved to far right</i>	Extreme Valid	The far right of the image is displayed, but no more	This tests that the far right of the image can be displayed and also that it stops there
<i>Vertical Scroll bar is moved to highest</i>	Extreme Valid	The highest of the image is displayed, but no more	This tests that the highest of the image can be displayed and also that it stops there
<i>Vertical Scroll bar is moved to lowest</i>	Extreme Valid	The lowest of the image is displayed, but no more	This tests that the lowest of the image can be displayed and also that it stops there
<i>Horizontal scroll bar is moved far right, then form size increased</i>	Extreme Valid	Centre locations is moved to the left when needed	This tests that the centre location is moved when it no longer becomes valid due to form size increasing

<i>Form is resized to smallest, then maximized</i>	Extreme	The bars disappear and the view is normal	This tests that the program will cope if form size is rapidly increased
	Valid		

### Algorithm 2.12 Redrawing needed pixels

For the moment, all pixels will be redrawn. If this becomes particularly cumbersome (during development) an algorithm may be designed to determine which pixels are new. However as this algorithm may be difficult to design, and computationally expensive to execute (may end up being slower than brute-force), this will only be investigated if necessary.

### Algorithm 2.13 Determining mouse position on picture box

The last task that needs to be implemented now is handling interaction with the mouse. This will be integral for any user interaction with the program.

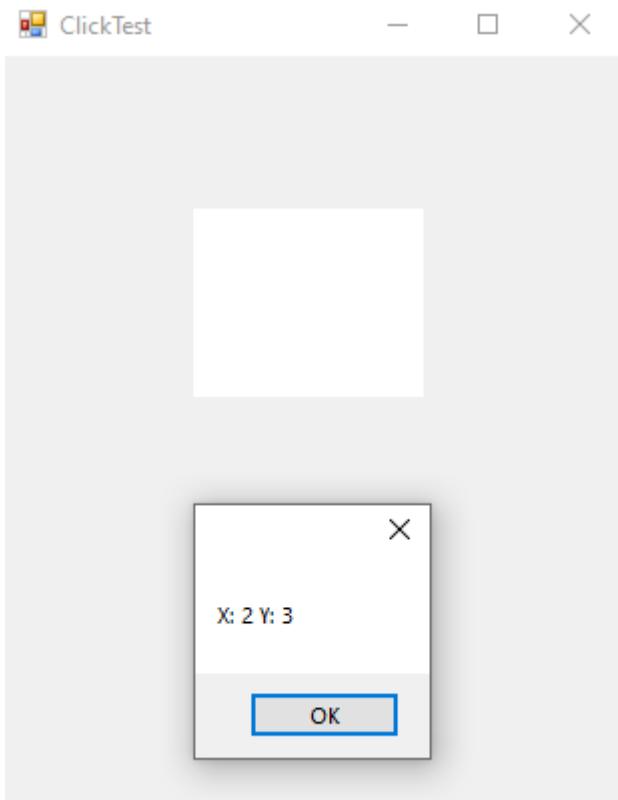
The first task is to determine where the mouse location occurs relative to the picture box, this is because the only information that Windows provides is the mouse's location on the entire screen. It must be necessary to find the location of the display form on the screen.

This can be solved by using C#.NET's in-built Click event. This returns the location of the mouse relative to the picture box.

Demonstration Code

```
void PictureBox1Click(object sender, EventArgs e)
{
    MouseEventArgs me = (MouseEventArgs)e;
    MessageBox.Show(String.Format("X: {0} Y: {1}", me.X, me.Y));
}
```

When clicking at the top left corner of the picture box results in:



Showing that this click event will give the location of the mouse on the picture box, irrespective of where it is on the form.

This happens to also be the display pixel that the mouse resides in on the form, so there is no necessary conversion there.

### Algorithm 2.14 + 2.15 Finding Location on overall image

As the location of the mouse pointer correlates to a display pixel, then finding the corresponding file pixel is easy, as an algorithm to convert from display pixels to file pixels [has already been designed](#), so no extra design is needed here.

## 2.1.7 Class Design Revisited

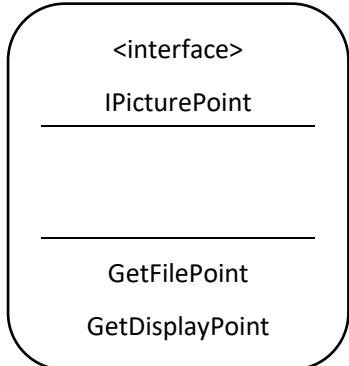
As the algorithms have all been designed, they will need to be allocated to their respective class. Some new classes will also need to be designed as their needs have increased.

Class relation diagrams can be found [here](#).

However, for the underlying framework, two new interfaces will be introduced:

### 2.1.7.1 IPicturePoint

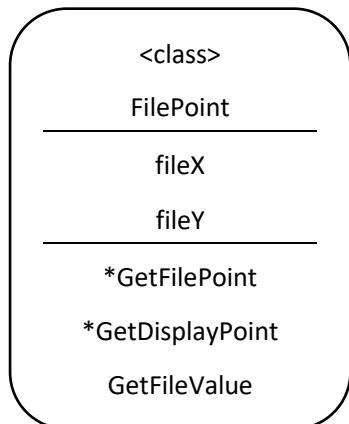
This will be the superclass of all classes that refer to a location on an image. This ensures that any point referred to in code will be able to transfer between file and display points.



#### Methods

Method	Params	Return type	Justification
GetFilePoint	None	FilePoint	Will return the point represented as a File Point
GetDisplay-Point	None	DisplayPoint	Will return the point represented as a Display Point

## 2.1.7.2 FilePoint



This class represents the **entity** of a file point existing on the image. It inherits from IPicturePoint so it must define its own functions for converting itself to a display point, when needed.

### Properties

Property	Datatype	Justification
fileX	(private) Integer	The X position of this point in the file  It is private to enforce getting the coords through GetAxisValue
fileY	(private) Integer	The Y position of this point in the file  It is private to enforce getting the coords through GetAxisValue

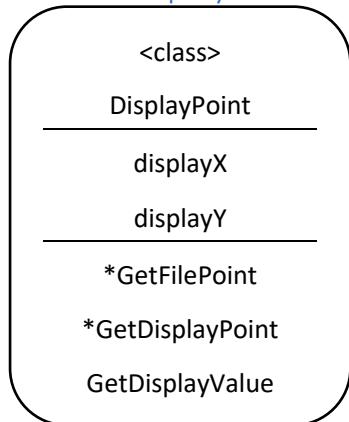
### Methods

Method	Params	Return type	Justification
GetFilePoint	None	<a href="#">FilePoint</a>	Will return itself (as this is a file point)
GetDisplay-Point	None	<a href="#">DisplayPoint</a>	Will convert itself into a display point
GetFile-Value	axis, Axis, the axis to get the coord in	Integer	Will return the value of the coord in the X or Y Axis

### Constructor

```
Constructor (int newFileX, int newFileY) {
    fileX = newFileX
    fileY = newFileY
}
```

### 2.1.7.3 DisplayPoint



This class represents the **entity** of a display point existing on the image. It inherits from **IPicturePoint** so it must define its own functions for converting itself to a file point, when needed.

#### Properties

Property	Datatype	Justification
displayX	(private) Integer	The X position of this point on the display form. It is private to enforce getting the coords through GetDisplayValue
displayY	(private) Integer	The Y position of this point on the display form. It is private to enforce getting the coords through GetDisplayValue

#### Methods

Method	Params	Return type	Justification
GetFilePoint	None	<a href="#">FilePoint</a>	Will convert itself into a file point
GetDisplay-Point	None	<a href="#">DisplayPoint</a>	Will return itself (as this is a display point)
GetDisplay-Value	axis, Axis, the axis to get the coord in	Integer	Will return the value of the coord in the X or Y Axis

#### Constructor

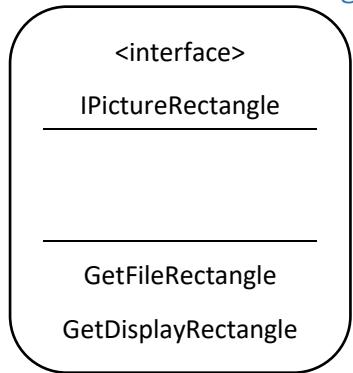
```
Constructor (int newFileX, int newFileY) {  
    fileX = newFileX  
    fileY = newFileY  
}
```

Why have two classes for file and display points?

As a file point and a display point are separate in this design, it makes sense to also separate them in the class design. This increases readability as a function can require or return a file point, and it makes conversions between file and display points much more explicit.



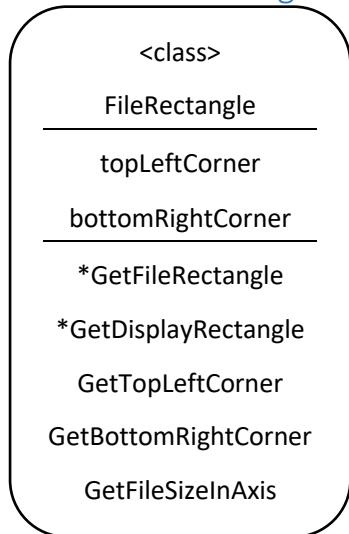
## 2.1.7.4 IPictureRectangle



### Methods

Method	Params	Return type	Justification
GetFile- Rectangle	None	<a href="#">FileRectangle</a>	Will return the rectangle represented as a File Rectangle
GetDisplay- Rectangle	None	<a href="#">DisplayRectangle</a>	Will return the rectangle represented as a Display Rectangle

## 2.1.7.5 FileRectangle



### Properties

Property	Datatype	Justification
topLeftCorner	(private) <a href="#">FilePoint</a>	The location of the top left corner in the file  It is private to enforce getting the location through GetTopLeftCorner
bottomRightCorner	(private) <a href="#">FilePoint</a>	The Y position of this point in the file  It is private to enforce getting the location through GetBottomRightCorner

### Methods

Method	Params	Return type	Justification
*GetFile-Rectangle	None	<a href="#">FileRectangle</a>	Will return itself as this is already a file rectangle
*GetDisplay-Rectangle	None	<a href="#">DisplayRectangle</a>	Will convert itself into a display rectangle
GetTop-LeftCorner	None	<a href="#">FilePoint</a>	Will return the location in the file of the top left corner
GetBottom-RightCorner	None	<a href="#">FilePoint</a>	Will return the location in the file of the bottom right corner
GetFileSizeInAxis	axis, Axis, the axis to get the coord in	Integer	Will return the size (width or height) depending on the inputted axis

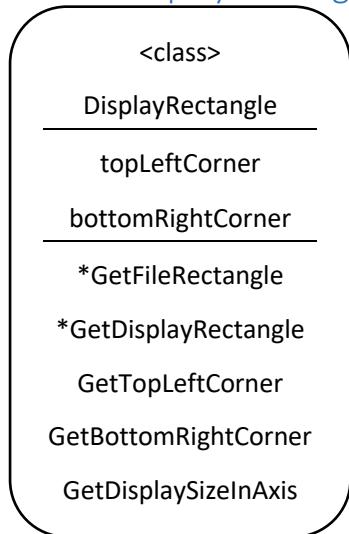
### Constructor

```

Constructor(fileTopLeftX, fileTopLeftY, fileBottomRightX, fileBottomRightY) {
    topLeftCorner = new FilePoint(fileTopLeftX, fileTopLeftY)
  
```

```
bottomRightCorner = new FilePoint(fileBottomRightX, fileBottomRightY)  
}
```

## 2.1.7.5 DisplayRectangle



### Properties

Property	Datatype	Justification
topLeft-Corner	(private) <a href="#">DisplayPoint</a>	The location of the top left corner in the Display It is private to enforce getting the location through GetTopLeftCorner
bottomRight-Corner	(private) <a href="#">DisplayPoint</a>	The Y position of this point in the Display It is private to enforce getting the location through GetBottomRightCorner

### Methods

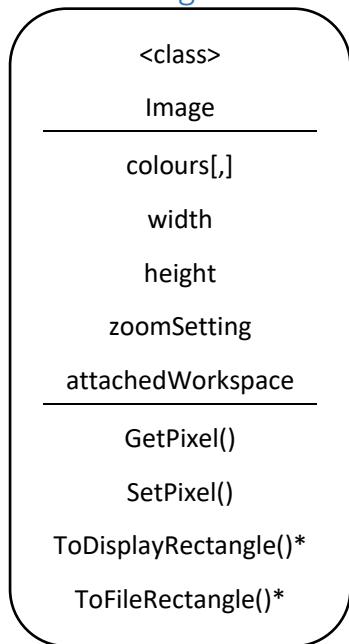
Method	Params	Return type	Justification
*GetFile-Rectangle	None	<a href="#">FileRectangle</a>	Will convert itself into a File Rectangle
*GetDisplay-Rectangle	None	<a href="#">DisplayRectangle</a>	Will return itself as this is already a Display Rectangle
GetTop-LeftCorner	None	<a href="#">DisplayPoint</a>	Will return the location in the Display of the top left corner
GetBottom-RightCorner	None	<a href="#">DisplayPoint</a>	Will return the location in the Display of the bottom right corner
GetDisplay-SizeInAxis	axis, Axis, the axis to get the coord in	Integer	Will return the size (width or height) depending on the inputted axis

### Constructor

```
Constructor(DisplayTopLeftX, DisplayTopLeftY, DisplayBottomRightX,  
DisplayBottomRightY) {
```

```
    topLeftCorner = new DisplayPoint(DisplayTopLeftX, DisplayTopLeftY)
    bottomRightCorner = new DisplayPoint(DisplayBottomRightX,
DisplayBottomRightY)
}
```

## 2.1.7.6 Image



The image class will store the main properties of the image.

### Properties

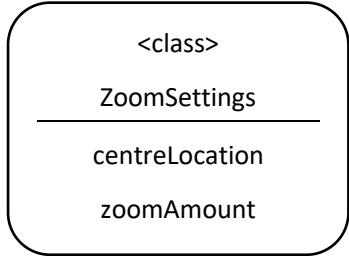
Property	Datatype	Justification
colours[,]	2D array of type 'Color'	An array of colours is what will be needed to store the property of each pixel on the grid. The array will make use of the existing 'Color' class in C#, so that I do not reinvent the wheel.
width	Integer	Stores the width of the current image.
height	Integer	Stores the height of the current image.
zoomSetting	<a href="#">ZoomSettings</a>	Stores the current settings describing the zoom of the image.
attached-Workspace	<a href="#">Workspace</a>	The workspace in which this image exists.

### Methods

Method	Params	Return type	Justification
GetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get	Colour	Will get an existing colour from a specific point in the image, regardless of its size.
SetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get Colour, Color, colour of pixel to set	None	Will set a new colour onto the grid, similar to GetPixel.
ToDisplay-Rectangle()	None	DisplayRectangle	Will return the Display size of the image

ToFile- Rectangle()	None	FileRectangle	Will return the File size of the image
------------------------	------	---------------	--

### 2.1.7.7 ZoomSettings



ZoomSettings contains all the properties about the current zoom on the image. This is separate from the image class to enforce proper **encapsulation**, as the level of zoom is not directly tied to the image.

#### Properties

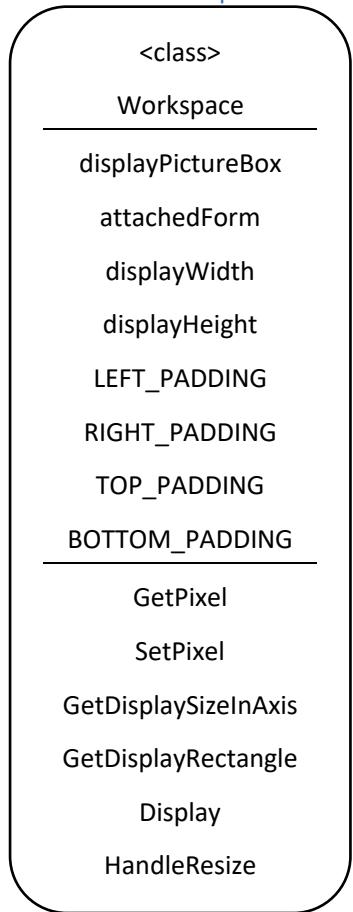
Property	Datatype	Justification
centre-Location	<a href="#">FilePoint</a>	Stores the location of the pixel in the middle of the zoom. The middle location is stored to make the zoom feel more natural when zooming in and out (the middle pixel remains the same)
zoom-Amount	Integer	Stores the amount that the image is currently zoomed in by. 1 = 1x = 100% zoom, 2 = 200% zoom etc.

#### Constructor

```
class ZoomSettings {
    FilePoint centreLocation
    Integer zoomAmount
    constructor(_centreLocation) {
        centreLocation = _centreLocation
        zoomAmount = 1
    }
}
```

The code shows a constructor for the ZoomSettings class. It takes a parameter '\_centreLocation' of type FilePoint and initializes the 'centreLocation' instance variable. It also initializes the 'zoomAmount' instance variable to 1. A callout box with a red dashed border highlights the line 'zoomAmount = 1' with the text: 'The default zoom will be none - 1x.'

## 2.1.7.8 Workspace



### Properties

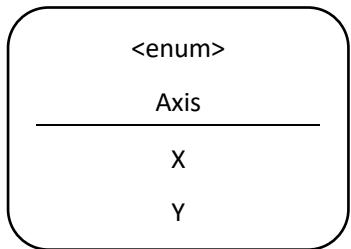
Property	Datatype	Justification
Display-PictureBox	PictureBox	The .NET Picture Box that this workspace will use to display itself into.
attachedForm	Form	The form that this workspace is attached to. This will also provide access to the form controls.
displayWidth	Integer	The current width of pixels that can be used for displaying in. Will be updated each time the form changes size
displayHeight	Integer	The current height of pixels that can be used for displaying in. Will be updated each time the form changes size
LEFT_PADDING	(constant) Integer	The amount of padding on the left hand side. Used for calculations involving where to place the picture box
RIGHT_PADDING	(constant) Integer	The amount of padding on the right hand side
TOP_PADDING	(constant) Integer	The amount of padding on the top side
BOTTOM_PADDING	(constant) Integer	The amount of padding on the bottom side

## Methods

Method	Params	Return type	Justification
GetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get	Colour	A mirror of Image's <a href="#">GetPixel</a> , also included here for convenience
SetPixel()	X, int, X coord of pixel to get Y, int, Y coord of pixel to get Colour, Color, colour of pixel to set	None	A mirror of Image's <a href="#">SetPixel</a> , also included here for convenience
GetDisplay-SizeInAxis	None	Integer	Returns the amount of display pixels in a certain axis
GetDisplay-Rectangle	None	<a href="#">DisplayRectangle</a>	Returns a Display Rectangle that represents the entire display area.
Display	None	None	Calls the code to display the image
Handle-Resize	None	None	Called when the form resizes, move all of the display components into the correct positions.

### 2.1.7.9 Axis

The scrollbar enum is a small set of constants for deciding whether to manipulate the X axis or the Y axis. This enum has no purpose by itself but makes the code more **readable** and **maintainable** by making it clear which axis is being manipulated.



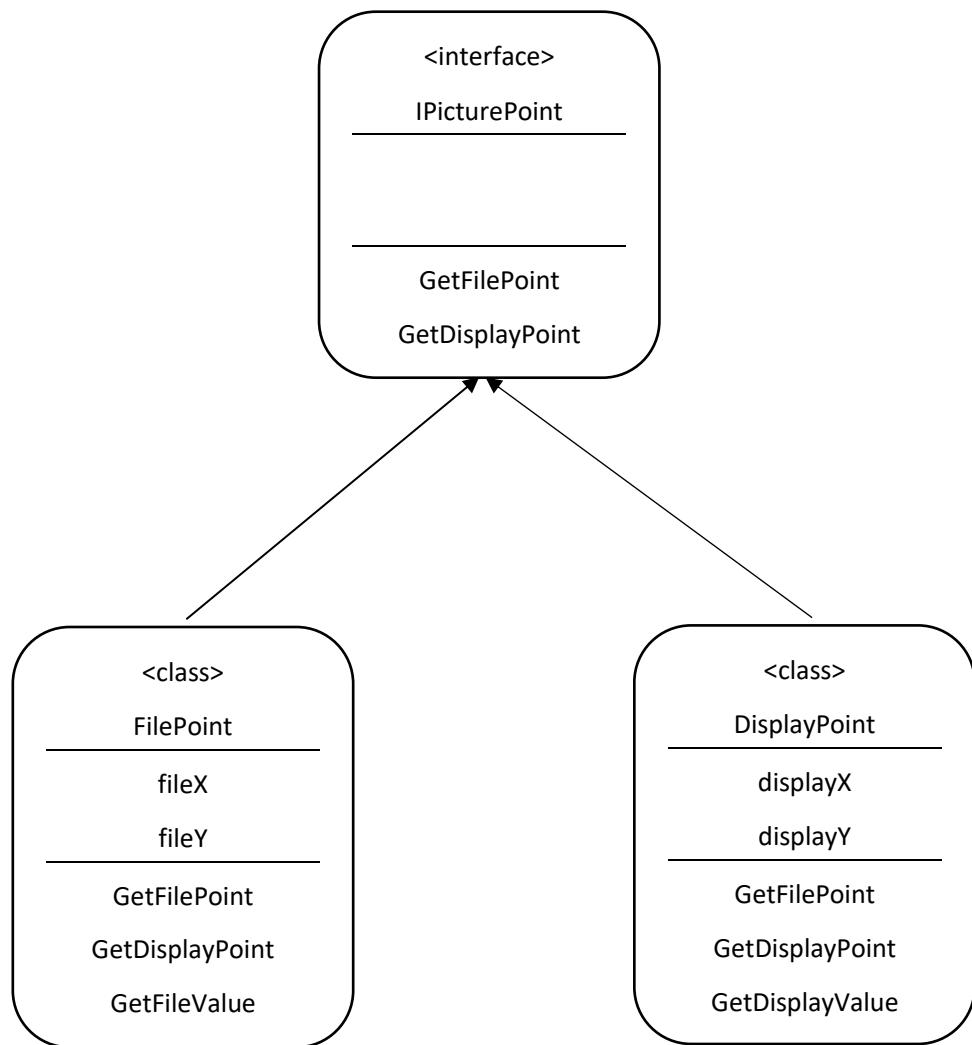
#### Members

Member	Justification
X	Denotes that the X axis has been selected.
Y	Denotes that the Y axis has been selected.

## 2.1.8 Class Relation Diagram

This demonstrates how the classes described in 2.1.7 will interact with each other to provide a full **abstraction stack** to make image editing easy later on.

### 2.1.8.1 Point abstraction stack

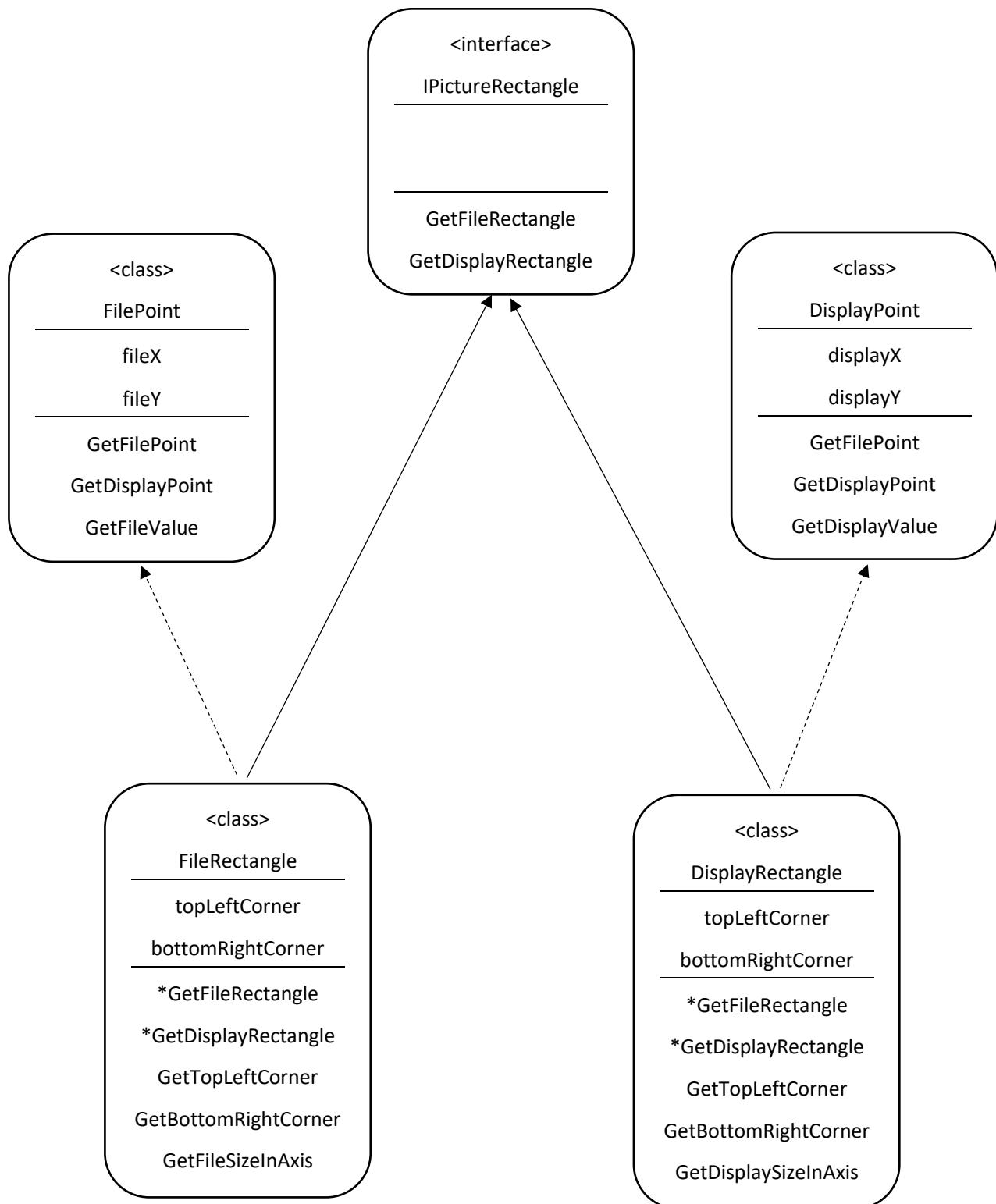


Key:

A —————→ B

A inherits from B

### 2.1.8.2 Rectangle abstraction stack



Key:

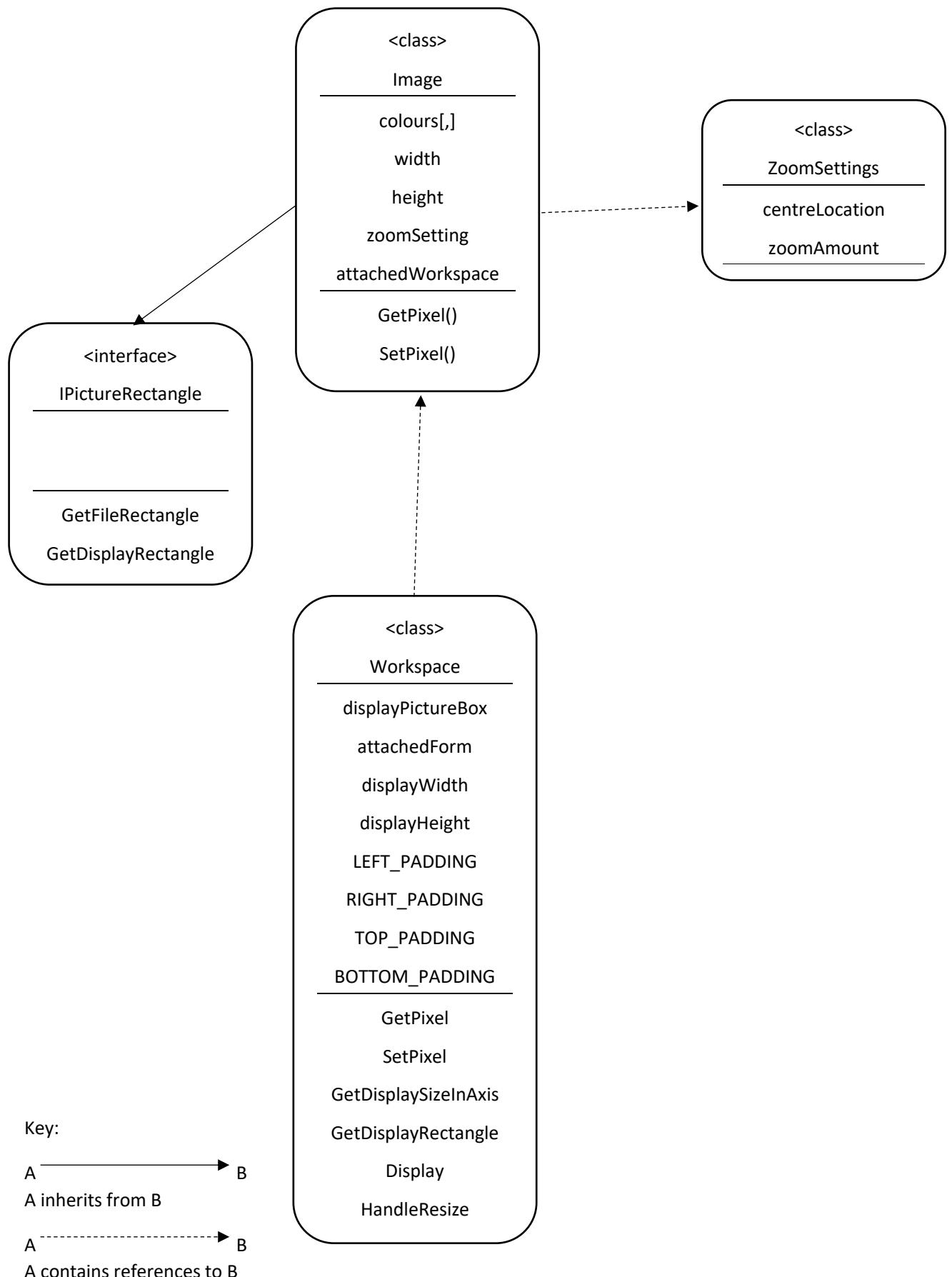
A —————→ B

A inherits from B

A -----→ B

A contains references to B

### 2.1.8.3 Overall stack



## 2.1.9 Key Data Structures

---

The MainForm will contain one data structure. A list of workspaces.

Property	Datatype	Justification
open-Workspaces	List of Workspaces	This stores any current workspaces the program has open.

This means that each instance of the WorkSpace class will **encapsulate** an entire window. This means that there can be multiple distinct instances of a program open at once.

## 2.1.20 Full Section Testing

This is a repeat of the unit tests from before, but tested again to ensure cooperation between the modules

Test Data	Test Type	Expected Output	Description	ID
<code>new Image(10,10)</code>	Valid	An image of size 10fpx, 10fpx	This tests if a normal image can be created	1
<code>new Image(10,6)</code>	Valid	A 10fpx, 5fpx image	This tests if a non-rectangular image can be created	2
<code>new Image(10,5)</code>	Valid	A 10fpx, 2fpx image	This tests when an odd dimension is entered	3
<code>new Image(10,1)</code>	Extreme Valid	A 10fpx, 1fpx image	This tests if a very short image can be created	4
<code>new Image(1,10)</code>	Extreme Valid	A 1fpx, 10fpx image	This tests if a very thin image can be created	5
<code>new Image(1,1)</code>	Extreme Valid	A 1fpx, 1fpx image	This tests when a very small image is created	6
<code>new Image(10,0)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no height is rejected	7
<code>new Image(0,10)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width is rejected	8
<code>new Image(0,0)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if an image with no width or height is rejected	9
<code>new Image(10000,10)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a width which is much too large is rejected	10
<code>new Image(10,10000)</code>	Extreme Invalid	The parameters are rejected and no image is created	This tests if a height which is much too large is rejected	11
<code>new Image(-1,10)</code>	Invalid	The parameters are rejected and no image is created	This tests if a negative width is rejected.	12
<code>new Image(10,-1)</code>	Invalid	The parameters are rejected and no image is created	This tests if a negative height is rejected.	13
<code>new Image(10)</code>	Erraneous	The parameters are rejected and no image is created	This tests if an image missing a height is rejected.	14

<code>new Image("10","10")</code>	Erraneous	The parameters are rejected and no image is created	This tests if an image with invalid numbers is rejected	15
<code>SetPixel(5,5,Black)</code>	Valid	A pixel near the middle of the image is set to black	This tests if a normal pixel can be set	16
<code>SetPixel(5,5,Green)</code>	Valid	A pixel near the middle of the image is set to yellow	This tests if a pixel can be set to other colours than black	17
<code>SetPixel(0,0,Black)</code>	Extreme Valid	A pixel in the top-left corner is set to black	This tests if the top-left corner can be set	18
<code>SetPixel(9,9,Black)</code>	Extreme Valid	A pixel in the bottom-right corner is set to black	This tests if the bottom-right corner can be set	19
<code>SetPixel(-1,0,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far left is rejected	20
<code>SetPixel(0,-1,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far up is rejected	21
<code>SetPixel(10,0,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far right is rejected	22
<code>SetPixel(0,10,Black)</code>	Extreme Invalid	No pixel is set as it is out of bounds	This tests if a pixel too far down is rejected	23
<i>Form is started at normal size</i>	Valid	Image Displays in the middle of the form	This tests that the program can start and display the image	24
<i>Form is maximized</i>	Extreme Valid	Image displays in the middle of the large form	This tests if the image is displayed when the form is very large	25
<i>Form is minimized</i>	Extreme Valid	No image is displayed (as form is currently invisible)	This tests that the program can be minimized safely	26
<i>Form is resized to smallest possible</i>	Extreme Valid	The form cannot be made smaller than the image	This tests that the form can never be made smaller than the image	27
<i>Form is resized to minimum width maximum height</i>	Extreme Valid	A very thin form displays the image	This tests if a form with an extreme height but minimum width is accepted	28
<i>Form is resized to minimum height maximum width</i>	Extreme Valid	A very short form displays the image	This tests if a form with an extreme width but minimum height is accepted	29

<i>The form's size is rapidly changed.</i>	Extreme Valid	The image is very quickly moved around but remains centred	This tests that under a stress test the image is displayed correctly	30
<i>Form is resized to half the size of image</i>	Valid	Bar is half size of bounds and in its centre position	This tests if the bar can display normally	31
<i>Form is resized to its smallest position</i>	Extreme Valid	Bar is small but still central.	This tests if the centre location remains in its position when the window size is reduced	32
<i>Form is resized back to half size of image</i>	Valid	Bar is half size of bounds and in its centre position	This tests if the centre location remains in its position when the window size is increased	33
<i>Horizontal Scroll bar is moved to far left</i>	Extreme Valid	The far left of the image is displayed, but no more	This tests that the far left of the image can be displayed and also that it stops there	34
<i>Horizontal Scroll bar is moved to far right</i>	Extreme Valid	The far right of the image is displayed, but no more	This tests that the far right of the image can be displayed and also that it stops there	35
<i>Vertical Scroll bar is moved to highest</i>	Extreme Valid	The highest of the image is displayed, but no more	This tests that the highest of the image can be displayed and also that it stops there	36
<i>Vertical Scroll bar is moved to lowest</i>	Extreme Valid	The lowest of the image is displayed, but no more	This tests that the lowest of the image can be displayed and also that it stops there	37
<i>Horizontal scroll bar is moved far right, then form size increased</i>	Extreme Valid	Centre locations is moved to the left when needed	This tests that the centre location is moved when it no longer becomes valid due to form size increasing	38
<i>Form is resized to smallest, then maximized</i>	Extreme Valid	The bars disappear and the view is normal	This tests that the program will cope if form size is rapidly increased	39

## 2.1.21 Testing Plan

---

The unit tests will be completed as described by the documentation in the order that they are described. If any test in the unit is failed, then the error(s) will be rectified and then the entire unit will be tested again. This is to ensure no chance of a 'domino effect' of errors, where fixing one may trigger another. This approach of unit testing will be **iteratively applied** until the entire unit is passed, at which point design will start on the next unit.

During development, there will also be the practice of self-testing and diagnosing. I will note changes to code (designed here or not) and the reason for their changing in a **development diary**.

The testing for the program will be recorded in the following table

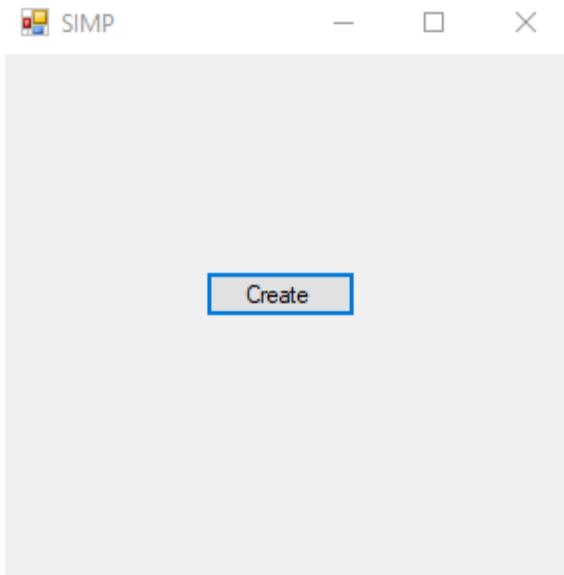
<i>Test</i>	<i>ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<i>The data inputted into the program</i>	0	What the program should output if it is working correctly	What the program outputs, which may differ from Expected	A comment on the performance of the test, ad why it might have failed

## 2.2 Development

18/09/2019 – Design for MainForm and basic Implementation of WorkSpace

---

A basic design for the Main Form has been made:



At the minute, it contains a very simple implementation for the 'Create' button, it creates a new Workspace.

```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();

    }

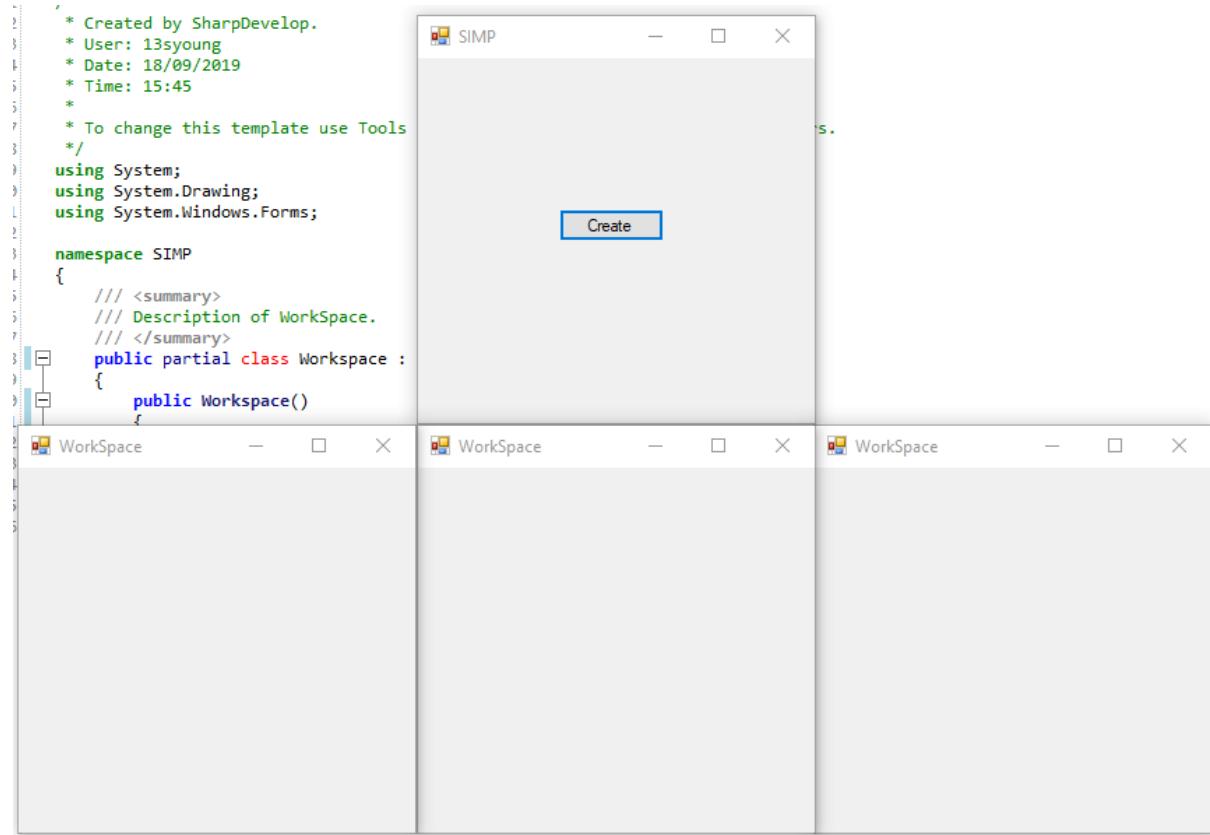
    void BtnCreateClick(object sender, EventArgs e)
    {
        Form newForm = new Workspace();
        newForm.Show();
    }
}
```

The Workspace itself only contains code for creating a form:

```
public partial class Workspace : Form
{
    public Workspace()
    {
        InitializeComponent();
    }
}
```

However defining WorkSpace as a separate class (and form) from the very beginning ensures that any and all Workspaces will be entirely separate, as adding support for multiple Workspaces later becomes more difficult.

As demonstrated here, multiple Workspaces can be created:

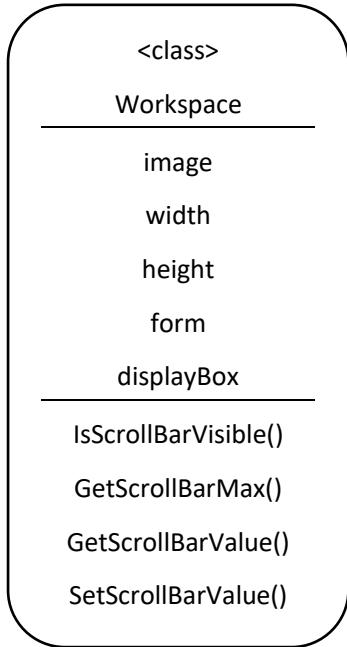


## 19/09/2019 – Remaining Class Implementation

---

### Workspace implementation

The remaining functionality for Workspace has been implemented, from the specification defined in 2.1.4.3:



Properties:

```
public partial class Workspace : Form
{
    public SIMP.Image image;
    public int width;
    public int height;
    public Form form;
    public PictureBox displayBox;
```

The constructor has also been defined:

```

public Workspace()
{
    InitializeComponent();
    Constructor(100,100);
}

private void Constructor(int width, int height) {
    // TODO: Update constructor when Image() is properly defined
    image = new SIMP.Image();

    // Sets the dimensions of the workspace to the dimensions of the form
    this.width = this.Width;
    this.height = this.Height;

    // Stores itself casted as a form
    form = (Form)this;

    // Sets up the dimensions of displayBox
    displayBox.Width = width;
    displayBox.Height = height;
}

```

The necessary functions have also been defined:

```

public bool IsScrollBarVisible(Axis axis) {
    throw new NotImplementedException();
}

public int GetScrollBarMax(Axis axis) {
    throw new NotImplementedException();
}

public int GetScrollBarValue(Axis axis) {
    throw new NotImplementedException();
}

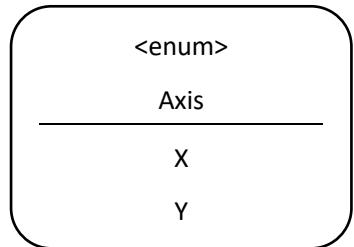
public void SetScrollBarValue(Axis axis, int newValue) {
    throw new NotImplementedException();
}

```

Their implementation has been omitted at this point, as the underlying framework is not implemented.

## Axis Implementation

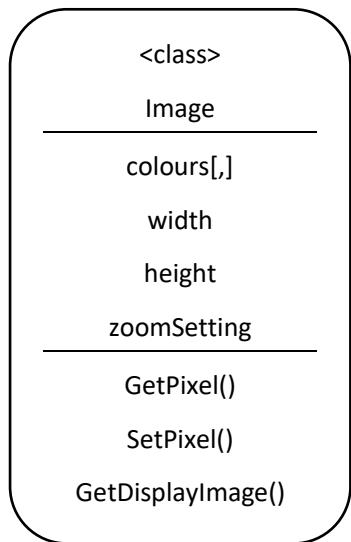
The Axis enum has been implemented in accordance to [2.1.4.2](#):



```
public enum Axis {
    X,
    Y
}
```

## Image Implementation

The image implementation has been completed in accordance to [2.1.4.1](#).



```
public Color[,] pixels;
public int width;
public int height;
public ZoomSettings zoomSettings;
```

The constructor has also been implemented, which is visibly similar to the proposed constructor:

```
class image {  
    constructor(_width, _height) {  
        width = _width  
        height = _height  
        pixels = new array of Color(width,height)  
        // gives every pixel a default white colour  
        foreach Color in Pixels {  
            Color = White  
        }  
        zoomSettings = new ZoomSetting()  
    }  
}
```

```
public Image(int width, int height)  
{  
    this.width = width;  
    this.height = height;  
    pixels = new Color[width,height];  
  
    for (int x = 0; x < width; x++) {  
        for (int y = 0; y < height; y++) {  
            pixels[x,y] = Color.White;  
        }  
    }  
  
    zoomSettings = new ZoomSettings();
```

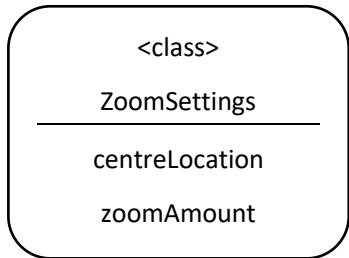
Some of the functions have also been implemented due to their programming simplicity.

```
public Color GetPixel(int x, int y) {  
    return pixels[x,y];  
}  
  
public void SetPixel(int x, int y, Color colour) {  
    pixels[x,y] = colour;  
}  
  
public Image GetDisplayImage() {  
    throw new NotImplementedException();  
}
```

However GetDisplayImage is a more complicated function and will be implemented later.

## ZoomSettings implementation

ZoomSettings has been implemented in accordance to [2.1.4.1](#):



```
public class ZoomSettings
{
    public int zoom;

    public ZoomSettings(int zoom = 1) {
        this.zoom = 1;
    }
}
```

However centreLocation has not been implemented. This is since it will only be relevant when the zooming has been fully implemented.

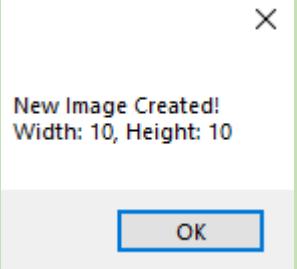
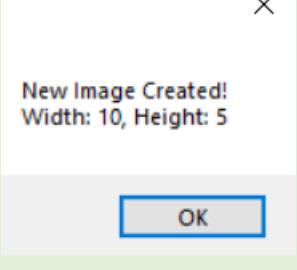
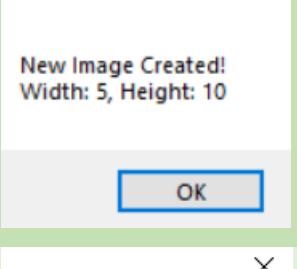
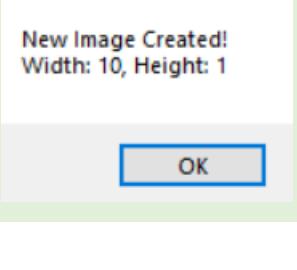
In doing this, [Algorithm 2.1](#) and [Algorithm 2.2](#) have already been implemented. This means the 2.2 Unit test should be run.

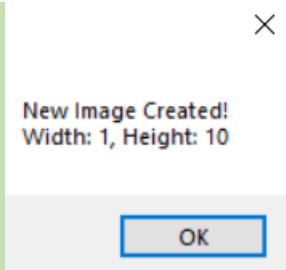
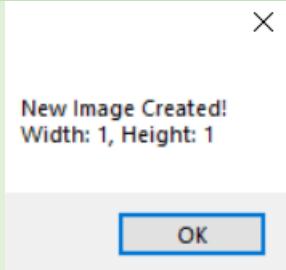
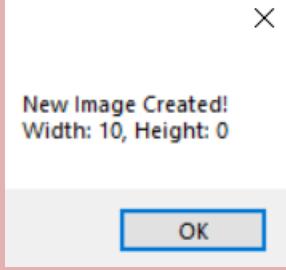
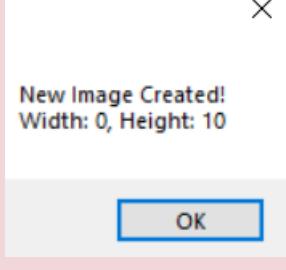
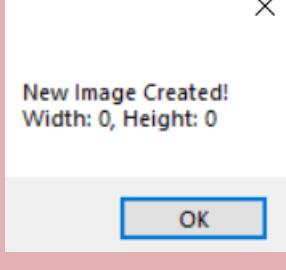
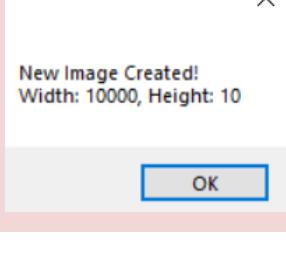
# 20/09/2019 – Unit Testing

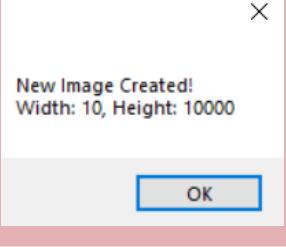
## Unit 2.2 Unit Test

The unit test will be completed using by running the following code snippet:

```
//Create a new SIMP Image
SIMP.Image testImage = new Image(10,10); // ← This is the part that will
//Displays the properties of that image
MessageBox.Show(string.Format(
    "New Image Created! " +
    "\nWidth: {0}, Height: {1}",
    testImage.width,testImage.height));
```

Test	ID	Expected Result	Actual Result	Comment
<code>new Image(10,10)</code>	1	An image of size 10fpx, 10fpx		The valid image is created
<code>new Image(10,5)</code>	2	A 10fpx, 5fpx image		The valid image is created
<code>new Image(5,10)</code>	3	A 5fpx, 10fpx image		The valid image is created
<code>new Image(10,1)</code>	4	A 10fpx, 1fpx image		The valid image is created

<code>new Image(1,10)</code>	5	A 1fpx, 10fpx image	×	The valid image is created
				
<code>new Image(1,1)</code>	6	A 1fpx, 1fpx image	×	The valid image is created
				
<code>new Image(10,0)</code>	7	The parameters are rejected and no image is created	×	The invalid image is not stopped
				
<code>new Image(0,10)</code>	8	The parameters are rejected and no image is created	×	The invalid image is not stopped
				
<code>new Image(0,0)</code>	9	The parameters are rejected and no image is created	×	The invalid image is not stopped
				
<code>new Image(10000,10)</code>	10	The parameters are rejected and no image is created	×	The invalid image is not stopped
				

<code>new Image(10,10000)</code>	11 The parameters are rejected and no image is created		The invalid image is not stopped
<code>new Image(-1,10)</code>	12 The parameters are rejected and no image is created	<code>System.OverflowException: Arithmetic operation resulted in an overflow.</code>	A potentially confusing error is thrown
<code>new Image(10,-1)</code>	13 The parameters are rejected and no image is created	<code>System.OverflowException: Arithmetic operation resulted in an overflow.</code>	A potentially confusing error is thrown
<code>new Image(10)</code>	14 The parameters are rejected and no image is created	'SIMP.Image' does not contain a constructor that takes 1 arguments	The argument is rejected correctly
<code>new Image("10","10")</code>	15 The parameters are rejected and no image is created	Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int'	The argument is rejected correctly

To fix these errors, some extra validation is needed in the Image constructor.

Fixing Error #7 & #9 & #12

To fix these errors, a small check can be implemented:

```
if (width <= 0) {  
    throw new ArgumentException("Width is less than 0", "Width");  
}
```

This throws a descriptive error about why the image was rejected.

Fixing Error #8, #13

To fix these errors, a small check can be implemented:

```
if (height <= 0) {  
    throw new ArgumentException("Height is less than 0", "Height");  
}
```

This throws a descriptive error about why the image was rejected.

Fixing Error #10

To fix this error, a small check can be implemented:

```
if (width > SimpConstants.IMAGE_MAX_WIDTH) {  
    throw new ArgumentException(String.Format("Width is greater than Image Max ({0})", SimpConstants.IMAGE_MAX_WIDTH), "Width");  
}
```

This throws a descriptive error about why the image was rejected.

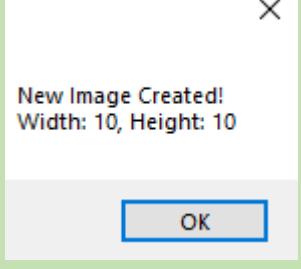
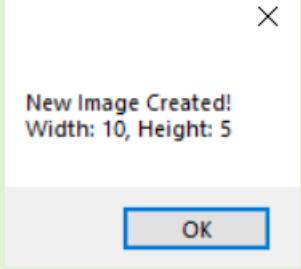
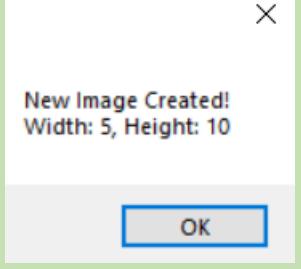
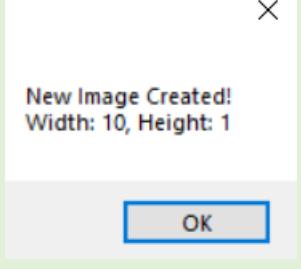
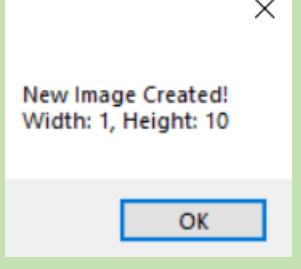
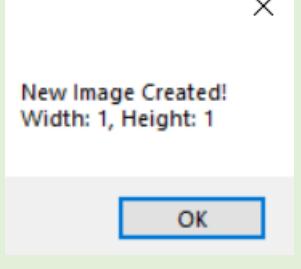
Fixing Error #11

To fix this error, a small check can be implemented:

```
if (height > SimpConstants.IMAGE_MAX_WIDTH) {  
    throw new ArgumentException(String.Format("Height is greater than Image Max ({0})", SimpConstants.IMAGE_MAX_WIDTH), "Height");  
}
```

This throws a descriptive error about why the image was rejected.

## Unit 2.2 Unit Test II

Test	ID	Expected Result	Actual Result	Comment
<code>new Image(10,10)</code>	1	An image of size 10fpx, 10fpx		The valid image is created
<code>new Image(10,5)</code>	2	A 10fpx, 5fpx image		The valid image is created
<code>new Image(5,10)</code>	3	A 5fpx, 10fpx image		The valid image is created
<code>new Image(10,1)</code>	4	A 10fpx, 1fpx image		The valid image is created
<code>new Image(1,10)</code>	5	A 1fpx, 10fpx image		The valid image is created
<code>new Image(1,1)</code>	6	A 1fpx, 1fpx image		The valid image is created

	7	The parameters are rejected and no image is created	System.ArgumentException: Height is 0 or less	A descriptive error is thrown
	8	The parameters are rejected and no image is created	System.ArgumentException: Width is 0 or less	A descriptive error is thrown
	9	The parameters are rejected and no image is created	System.ArgumentException: Width is 0 or less	A descriptive error is thrown
	10	The parameters are rejected and no image is created	System.ArgumentException: Width is greater than Image Max (9999)	A descriptive error is thrown
	11	The parameters are rejected and no image is created	System.ArgumentException: Height is greater than Image Max (9999)	A descriptive error is thrown
	12	The parameters are rejected and no image is created	System.ArgumentException: Width is 0 or less	A descriptive error is thrown
	13	The parameters are rejected and no image is created	System.ArgumentException: Height is 0 or less	A descriptive error is thrown
	14	The parameters are rejected and no image is created	'SIMP.Image' does not contain a constructor that takes 1 arguments	The argument is rejected correctly
	15	The parameters are rejected and no image is created	Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int'	The argument is rejected correctly

## SimpConstants implementation

SimpConstants is a small class that contains static constants to be used throughout the program. It consists of two constants currently:

```
public static class SimpConstants {  
    public static int IMAGE_MAX_WIDTH = 99999;  
    public static int IMAGE_MAX_HEIGHT = 99999;  
}
```

I discussed with my stakeholders, and they agreed that 99,999 pixels would be the largest that they'd need for an image.

# 21/09/2019 Picture Box setup

---

## Algorithm 2.3 – Resizing Picture Box

The algorithm for resizing the picture box has been implemented, though contains **changes** from the planned pseudocode:

```
class Image {  
    ...  
    ResizePictureBox(box) {  
        // uses the width and height properties to set properties of the  
        // picture box  
        box.width = width  
        box.height = height  
        // this ensures the picture box can hold the pixels  
    }  
}  
  
public void ResizePictureBox() {  
    displayBox.Width = image.width;  
    displayBox.Height = image.height;  
}
```

In the pseudocode, **ResizePictureBox** was included as part of the 'Image' class. However this has been changed to be part of the 'Workspace' class as Workspace contains both the Image and its relevant Picture Box. This enforces better **encapsulation** as Workspace manages the interaction between the Image and its box rather than having an unnecessary dependency.

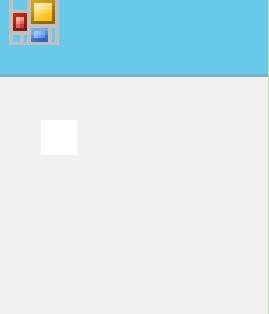
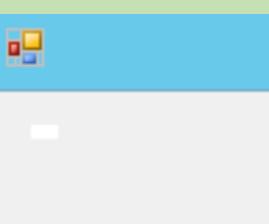
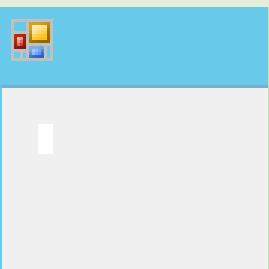
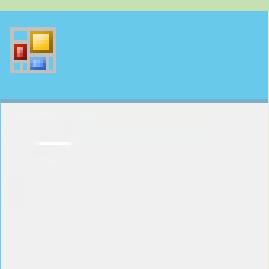
This also means that the 'width' property had to be changed to 'image.width' as 'width' would refer to the width of the Workspace now.

The constructor for the image has also been edited to include default image resolutions, which are taken from the **SamConstants** class.

```
Constructor(  
    SimpConstants.IMAGE_DEFAULT_WIDTH,  
    SimpConstants.IMAGE_DEFAULT_HEIGHT  
)  
  
public static class SimpConstants {  
    public static int IMAGE_MAX_WIDTH = 99999;  
    public static int IMAGE_MAX_HEIGHT = 99999;  
  
    public static int IMAGE_DEFAULT_WIDTH = 540;  
    public static int IMAGE_DEFAULT_HEIGHT = 360;  
}
```

I discussed this with my stakeholders and they agreed that 540 by 360 (a quarter of 1080 by 720) was a good resolution to start with.

## Unit 2.3 Unit Test

Test	ID	Expected Result	Actual Result	Comment
<i>Input Box(10,10)</i>	1	An image of size 10fpx, 10fpx		The Picture Box is correctly resized.
<i>Input Box(10,5)</i>	2	A 10fpx, 5fpx image		The Picture Box is correctly resized.
<i>Input Box(5,10)</i>	3	A 5fpx, 10fpx image		The Picture Box is correctly resized.
<i>Input Box(10,1)</i>	4	A 10fpx, 1fpx image		The Picture Box is correctly resized. (Though is very thin)
<i>Input Box(1,10)</i>	5	A 1fpx, 10fpx image		The Picture Box is correctly resized. (Though is very thin)

<i>Input Box(1,1)</i>	6	A 1fpx, 1fpx image		The Picture Box is correctly resized. (Though is very small)
<i>Input Box(10,0)</i>	7	The parameters are rejected and no image is created	<code>System.ArgumentException: Height is 0 or less</code>	A descriptive error is thrown
<i>Input Box(0,10)</i>	8	The parameters are rejected and no image is created	<code>System.ArgumentException: Width is 0 or less</code>	A descriptive error is thrown
<i>Input Box(0,0)</i>	9	The parameters are rejected and no image is created	<code>System.ArgumentException: Width is 0 or less</code>	A descriptive error is thrown
<i>Input Box(10000,10)</i>	10	The parameters are rejected and no image is created	<code>System.ArgumentException: Width is greater than Image Max (9999)</code>	A descriptive error is thrown
<i>Input Box(10,10000)</i>	11	The parameters are rejected and no image is created	<code>System.ArgumentException: Height is greater than Image Max (9999)</code>	A descriptive error is thrown
<i>Input Box(-1,10)</i>	12	The parameters are rejected and no image is created	<code>System.ArgumentException: Width is 0 or less</code>	A descriptive error is thrown
<i>Input Box(10,-1)</i>	13	The parameters are rejected and no image is created	<code>System.ArgumentException: Height is 0 or less</code>	A descriptive error is thrown
<i>Input Box(10)</i>	14	The parameters are rejected and no image is created	'SIMP.Image' does not contain a constructor that takes 1 arguments	The argument is rejected correctly
<i>Input Box("10","10")</i>	15	The parameters are rejected and no image is created	Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int'	The argument is rejected correctly

As this code is only used from the properties of the Image, this means that it reuses its code from the Image sanitation.

## Algorithm 2.4 – Displaying Image

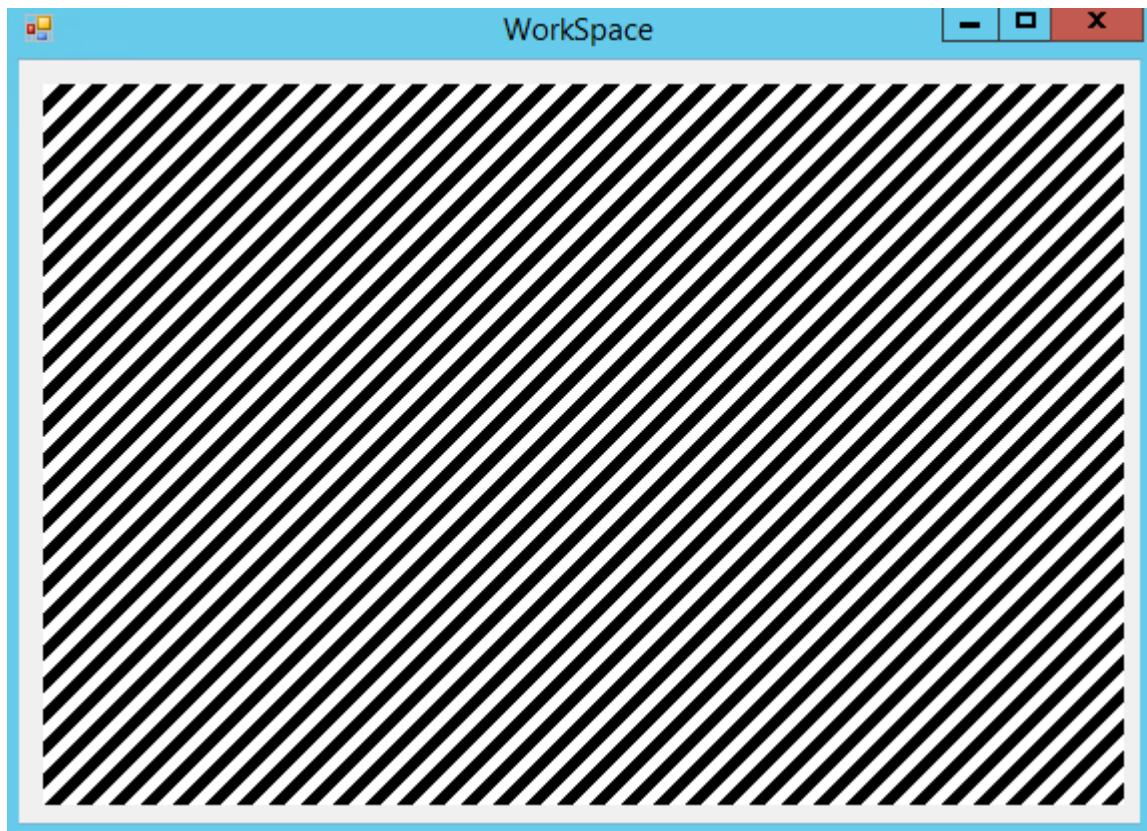
This algorithm has been constructed, according to the following pseudocode:

```
GetDisplayImage() {  
    // creates a C# 'image' object to store the output image  
    Drawing.Image image = new Drawing.Image(width,height)  
    for x = 1 to width {  
        for y = 1 to height {  
            colour = pixels[x,y]  
            Image.SetPixel(x,y,colour)  
        }  
    }  
    RETURN image  
}
```

```
public System.Drawing.Image GetDisplayImage() {  
    Bitmap newImage = new Bitmap(width,height);  
    Color colour;  
  
    for (int x = 0; x < width; x++) {  
        for (int y = 0; y < height; y++) {  
            colour = pixels[x,y];  
            newImage.SetPixel(x,y,colour);  
        }  
    }  
  
    return newImage;  
}
```

To run a small test on the capabilities of the display, a small pattern was created using the following code:

```
//create pattern  
for (int x = 0; x < width; x++) {  
    for (int y = 0; y < height; y++) {  
        if (((x+y)%8) <= 3) {  
            pixels[x,y] = Color.White;  
        } else {  
            pixels[x,y] = Color.Black;  
        }  
    }  
}
```



This demonstrates that the code can be used to display an image, however thorough unit testing will need to be completed.

# 22/09/2019 Setting Pixels

## Algorithm 2.5 – Setting Pixels at runtime

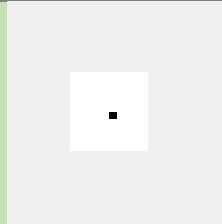
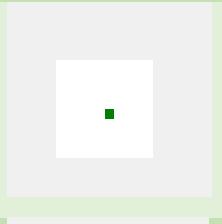
In order to implement this, a small implementation for SetPixel has been added, following the original pseudocode:

```
SetPixel(x,y,colour) {  
    pixels[x,y] = colour  
    Display()  
}
```

```
public void SetPixel(int x, int y, Color colour) {  
    pixels[x,y] = colour;  
}
```

However there has been a change, as the call to Display() has been removed. This makes it so that the image will *not* be redrawn each time a pixel is set, and this helps to optimise large groups of pixel manipulation. This also means that it will be up to the calling function to re-display when necessary

## Unit 2.5 Unit Test

Test	ID	Expected Result	Actual Result	Comment
<code>SetPixel(5,5,Black)</code>	1	A pixel near the middle of the image is set to black		This proves a pixel can be placed on the image.
<code>SetPixel(5,5,Green)</code>	2	A pixel near the middle of the image is set to yellow		This proves multiple colours of pixel can be set
<code>SetPixel(0,0,Black)</code>	3	A pixel in the top-left corner is set to black		This proves the top-left corner is properly accessible
<code>SetPixel(9,9,Black)</code>	4	A pixel in the bottom-right corner is set to black		This proves the bottom-right corner is properly accessible. Both extremes

have now been tested				
<i>SetPixel(-1,0,Black)</i>	5	No pixel is set as it is out of bounds	<code>System.IndexOutOfRangeException:</code> Index was outside the bounds of the array.	There is not an in-bounds check for this
<i>SetPixel(0,-1,Black)</i>	6	No pixel is set as it is out of bounds	<code>System.IndexOutOfRangeException:</code> Index was outside the bounds of the array.	There is not an in-bounds check for this
<i>SetPixel(10,0,Black)</i>	7	No pixel is set as it is out of bounds	<code>System.IndexOutOfRangeException:</code> Index was outside the bounds of the array.	There is not an in-bounds check for this
<i>SetPixel(0,10,Black)</i>	8	No pixel is set as it is out of bounds	<code>System.IndexOutOfRangeException:</code> Index was outside the bounds of the array.	There is not an in-bounds check for this

Fixing Error #5 & #6 & #7 & #8

To fix these errors, an extra check can be added onto the SetPixel function:

```
public void SetPixel(int x, int y, Color colour) {  
    try {  
        pixels[x,y] = colour;  
    } catch (IndexOutOfRangeException ie) {  
        // ignore request if it tried to set out of bounds  
    }  
}
```

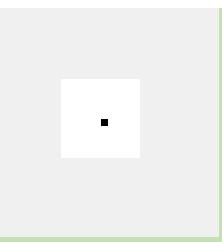
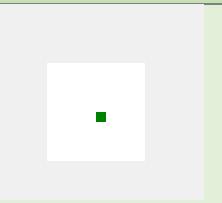
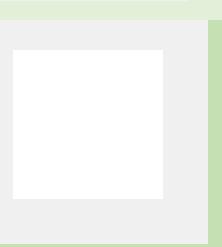
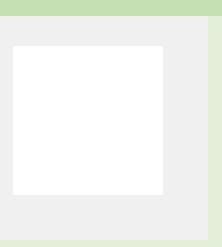
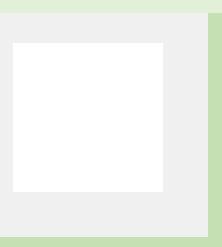
This will cause the code to ignore any out of bounds requests.

It was decided to implement the check this way rather than with a standard range check to **improve performance**. Range checks would add an overhead of four comparisons to every single pixel set, and this is important as many pixels may be set at once.

According to <https://stackoverflow.com/questions/52312/what-is-the-real-overhead-of-try-catch-in-c>, entering a 'try' block incurs almost no penalties, and so will have much less overall overhead.

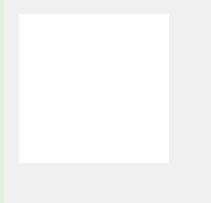
It was decided to ignore out of bounds exceptions rather than throwing an error to **improve stability**. This means that the program will not be forced to stop if any out of bounds pixel is set. The program is able to continue.

## Unit 2.5 Unit Test II

<i>SetPixel(5,5,Black)</i>	1	A pixel near the middle of the image is set to black		This proves a pixel can be placed on the image.
<i>SetPixel(5,5,Green)</i>	2	A pixel near the middle of the image is set to yellow		This proves multiple colours of pixel can be set
<i>SetPixel(0,0,Black)</i>	3	A pixel in the top-left corner is set to black		This proves the top-left corner is properly accessible
<i>SetPixel(9,9,Black)</i>	4	A pixel in the bottom-right corner is set to black		This proves the bottom-right corner is properly accessible. Both extremes have now been tested
<i>SetPixel(-1,0,Black)</i>	5	No pixel is set as it is out of bounds		This proves a too small X is rejected
<i>SetPixel(0,-1,Black)</i>	6	No pixel is set as it is out of bounds		This proves a too large X is rejected
<i>SetPixel(10,0,Black)</i>	7	No pixel is set as it is out of bounds		This proves a too small Y is rejected

*SetPixel(0,10,Black)*

**8**    No pixel is set as it is  
out of bounds



This proves a too  
large Y is rejected

## 23/09/2019 Resizing Picture Box

### Algorithm 2.6 – Resizing Picture Box

The code for resizing the Picture Box has been implemented, in accordance to planned pseudocode 2.6

```
RelocatePictureBox {  
    Integer X = (width - image.width) / 2  
    Integer Y = (height - image.height) / 2  
    displayBox.Location = new Location(X,Y)  
}  
  
private void RelocatePictureBox() {  
    int X = (width - image.width) / 2;  
    int Y = (height - image.height) / 2;  
    displayBox.Location = new Point(X,Y);  
}
```

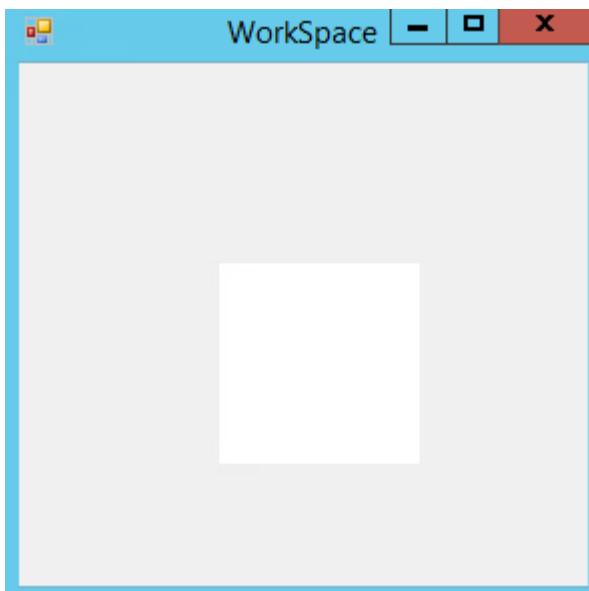
However, some additional code must be added to make the picture box constantly update. To do this a small timer module has been added to the code which calls the following procedure every millisecond:

```
void HeartbeatTick(object sender, EventArgs e)  
{  
    RelocatePictureBox();  
}
```

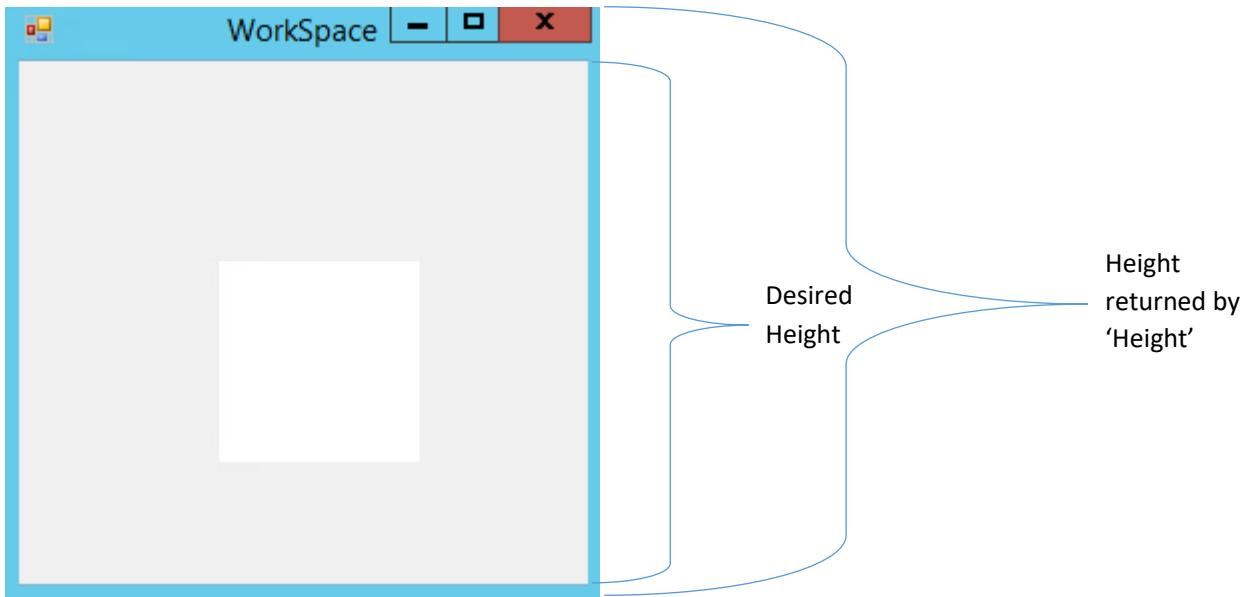
However this is still not enough, as 'width' and 'height' are currently constant and do not update when the form is resized. This can be solved by adding a new event called when the workspace is resized:

```
private void CalculateDimensions() {  
    // Sets the dimensions of the workspace to the dimensions of the form  
    width = Width;  
    height = Height;  
}
```

However, this results in the form not being quite centred:



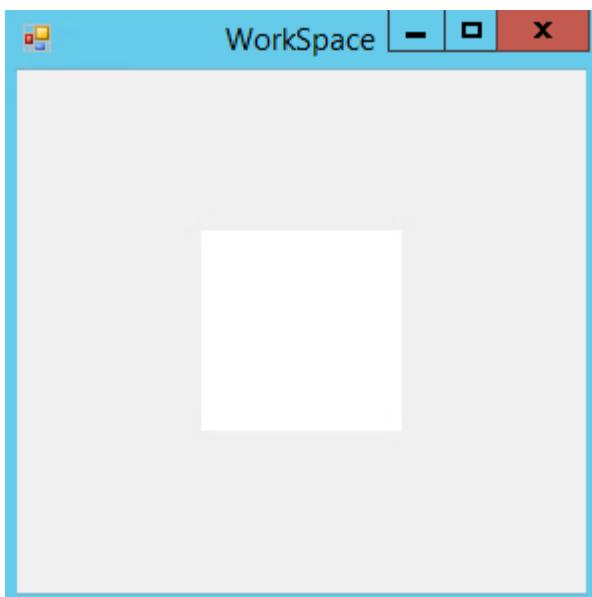
This is caused by the fact that calling the width of the form will return the **entire** width of the form, including its borders. For example:



This can be fixed by instead calling the '[DisplayRectangle](#)' property, which returns a rectangle with the desired dimensions:

```
private void CalculateDimensions() {
    // Sets the dimensions of the workspace to the dimensions of the form
    width = DisplayRectangle.Width;
    height = DisplayRectangle.Height;
}
```

Resulting in:



## 24/09/2019 Minimum Form Size

### Algorithm 2.7 – Minimum Form Size

The algorithm for setting the minimum size has been implemented in accordance to algorithm 2.7:

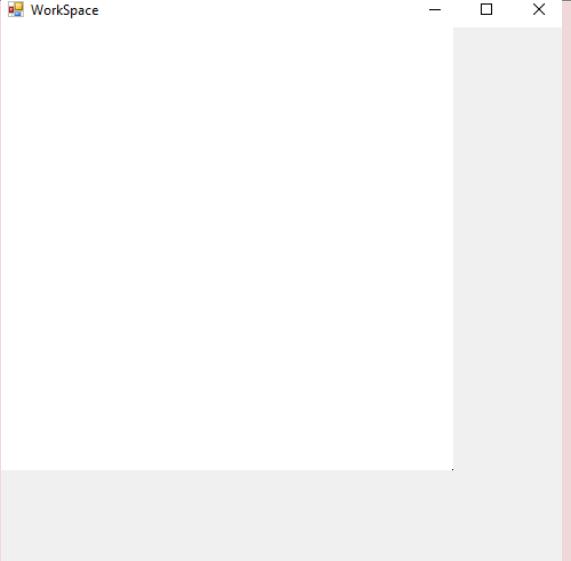
```
form.MinimumWidth = image.Width  
form.MinimumHeight = image.Height  
  
MinimumSize = new Size(image.Width, image.Height);
```

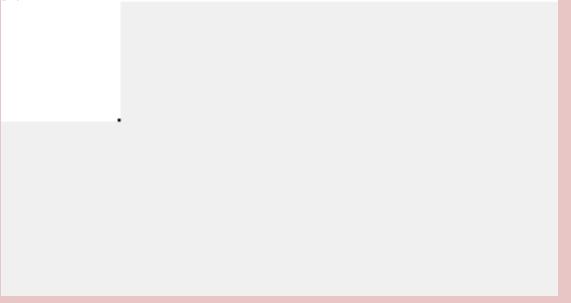
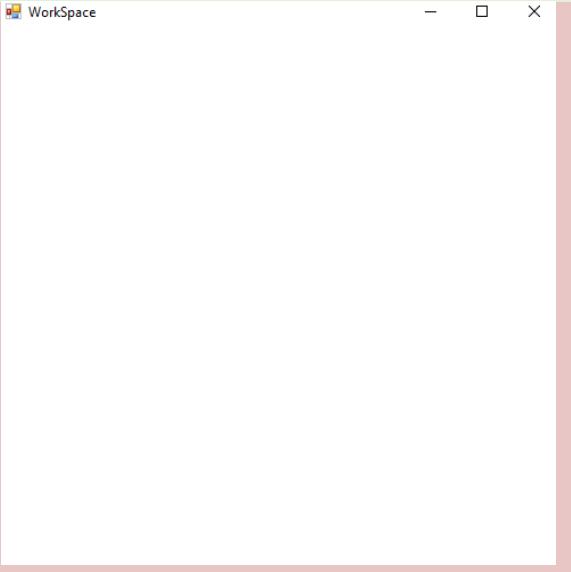
There is a small change in the way that the minimum size is implemented, as it must be added as a size object. Functionally this is the same.

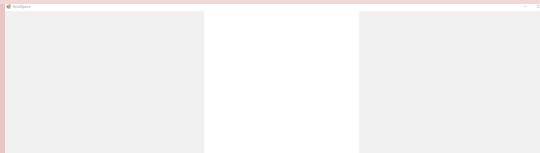
### Unit Test 2.7

In order to test this, two boxes have been added to the bottom left and top right of the image:

```
for (int x = 0; x < 10; x++) {  
    for (int y = 0; y < 10; y++) {  
        image.SetPixel(x,y,Color.Black);  
    }  
}  
  
for (int x = width-10; x < width; x++) {  
    for (int y = height-10; y < height; y++) {  
        image.SetPixel(x,y,Color.Black);  
    }  
}
```

	<i>Test ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<i>Form is started at normal size</i>	1	Image Displays in the middle of the form		The picture box is not correctly resized upon starting

<i>Form is maximized</i>	2	Image displays in the middle of the large form		The picture box is not resized upon maximizing
<i>Form is minimized</i>	3	No image is displayed (as form is currently invisible)	(No screenshot)	The program does not crash upon minimizing
<i>Form is resized to smallest possible</i>	4	The form cannot be made smaller than the image		The two squares in the corners are not displayed, so the minimum size is too small

<p><i>Form is resized to minimum width maximum height</i></p>	<p>5 A very thin form displays the image</p>		<p>The pixels in the corners are still cut off</p>
<p><i>Form is resized to minimum height maximum width</i></p>	<p>6 A very short form displays the image</p>		<p>The pixels in the corners are not displayed</p>
<p><i>The form's size is rapidly changed.</i></p>	<p>7 The image is very quickly moved around but remains centred</p>		<p>The picture box is only moved upon the end of resizing so the picture box aligns correctly eventually</p>

## Fixing Error #1 & #2

Errors #1 and #2 both stem from the Form not being resized when it should be. This is due to the ResizeEnd event that the resizing is currently linked to not triggering at all times. To fix this a function was added to check if the Form's size has changed:

```
private bool HasSizeChanged() {
    // if width has changed
    if (width != DisplayRectangle.Width) {
        return true;
    }

    // if height has changed
    if (height != DisplayRectangle.Height) {
        return true;
    }

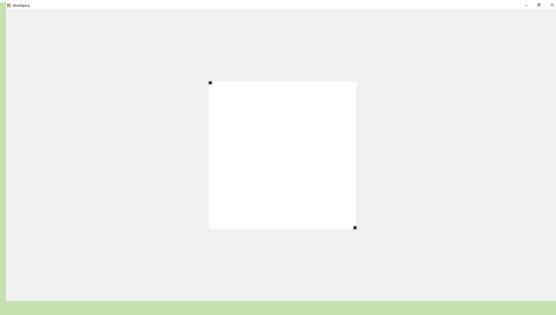
    return false;
}
```

From this, the Picture Box will be resized whenever the Form is resized:

```
private void HeartbeatTick(object sender, EventArgs e)
{
    if (HasSizeChanged()) {
        CalculateDimensions();
        RelocatePictureBox();
    }
}
```

So the tests can now be repeated:

Test	ID	Expected Result	Actual Result	Comment
Form is started at normal size	1	Image Displays in the middle of the form	 WorkSpace	- □ × The picture box is resized but the picture box does not correctly display the entire image.

<p>Form is maximized</p>	<p>2 Image displays in the middle of the large form</p>	 <p>The picture box is correctly placed when the form is resized</p>
--------------------------	---	--

Error #1 however has still not been resolved, due to another existing error.

### Fixing Error #1 & #4 & #5 & #6

These errors all stem from the minimum size of the Form being too small, this means that the image is cut at minimum size.

```
MinimumSize = new Size(image.width,image.height);
```

The error comes from the fact that MinimumSize does not take into account the borders around the Form, meaning elements like the top bar can cut the image off.

To resolve this, the size of the top bar (and other borders) needs to be determined so that they can be factored into the equation.

```
public static int WINDOWS_TOP_BAR_HEIGHT;
public static int WINDOWS_BOTTOM_BAR_HEIGHT;
public static int WINDOWS_LEFT_BAR_WIDTH;
public static int WINDOWS_RIGHT_BAR_WIDTH;
```

They can be calculated using the following code:

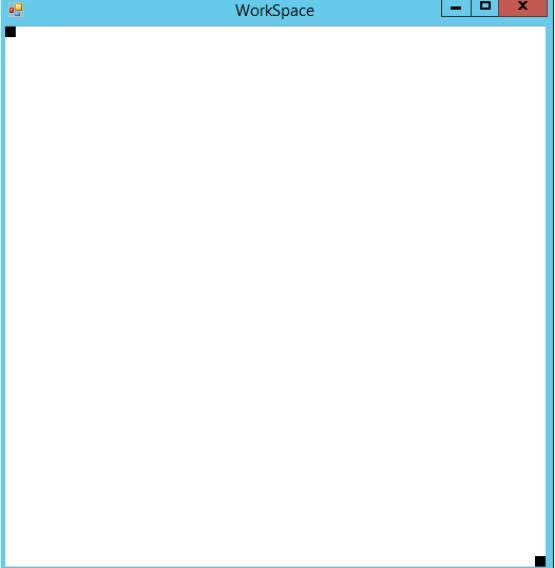
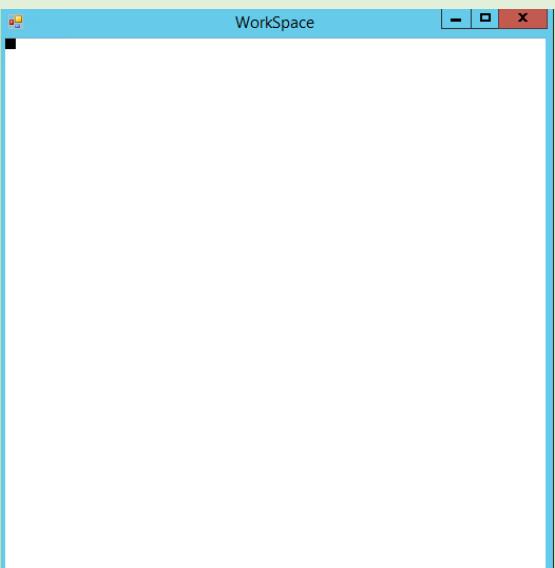
```
static SimpConstants() {
    // defines a test form
    Form testForm = new Form();

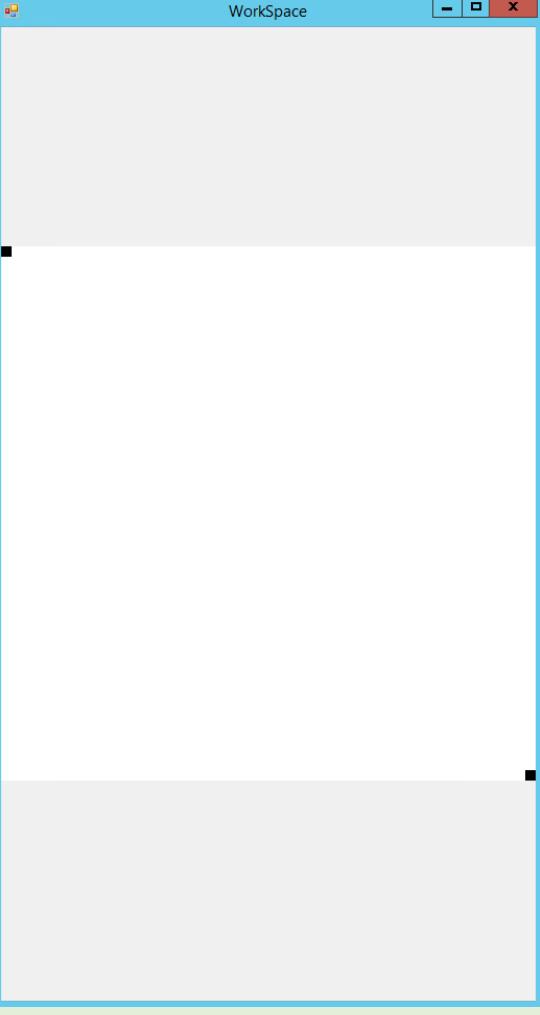
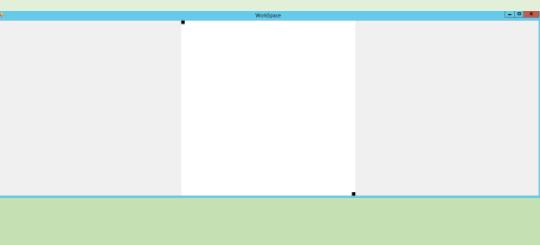
    // determines where the display rectangle appears on the screen
    Rectangle screenRectangle = testForm.RectangleToScreen(testForm.ClientRectangle);

    // determines the bar sizes by comparing that rectangle to the actual location of the form
    WINDOWS_TOP_BAR_HEIGHT = screenRectangle.Top - testForm.Top;
    WINDOWS_BOTTOM_BAR_HEIGHT = screenRectangle.Bottom - testForm.Bottom;
    WINDOWS_LEFT_BAR_WIDTH = screenRectangle.Left - testForm.Left;
    WINDOWS_RIGHT_BAR_WIDTH = screenRectangle.Right - testForm.Right;
}
```

So now they can be included in the size calculations

```
// Sets the minimum dimensions of the form
MinimumSize = new Size(
    width + SimpConstants.WINDOWS_LEFT_BAR_WIDTH + SimpConstants.WINDOWS_RIGHT_BAR_WIDTH,
    height + SimpConstants.WINDOWS_TOP_BAR_HEIGHT + SimpConstants.WINDOWS_BOTTOM_BAR_HEIGHT
);
```

<i>Test</i>	<i>ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<i>Form is started at normal size</i>	1	Image Displays in the middle of the form		The picture box starts at its smallest size, which is still valid.
<i>Form is resized to smallest possible</i>	4	The form cannot be made smaller than the image		The picture box cannot be made any smaller than this size.

<p><i>Form is resized to minimum width maximum height</i></p>	<p>5 A very thin form displays the image</p>		<p>A very tall form still correctly displays the pixels</p>
<p><i>Form is resized to minimum height maximum width</i></p>	<p>6 A very short form displays the image</p>		<p>The pixels in the corners are not displayed</p>

This means that all errors relating to resizing the Form has been resolved.

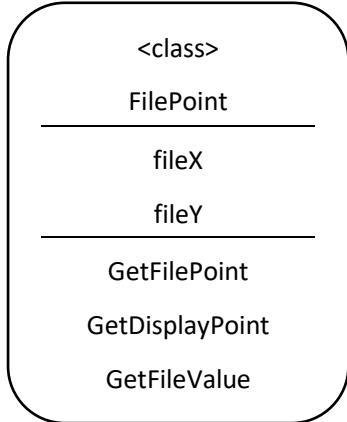
## 25/09/2019 Point Redesign

---

Before completing any further algorithms, some upgrades need to be completed on the base classes, in accordance to [2.1.6](#)

### FilePoint

A basic structure has been implemented for FilePoint, in accordance to [2.1.6.2](#)



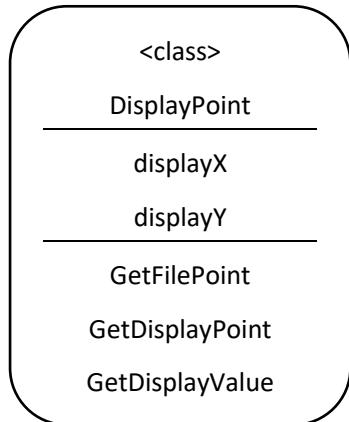
```
public class FilePoint
{
    private int _fileX;
    private int _fileY;

    public FilePoint(int fileX, int fileY)
    {
        _fileX = fileX;
        _fileY = fileY;
    }
}
```

However the methods have not been implemented yet, they will be implemented upon the creation of the IPicturePoint interface.

## DisplayPoint

A basic structure has been implemented for FilePoint, in accordance to [2.1.6.3](#)



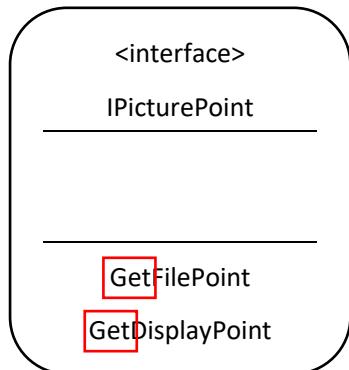
```
public class DisplayPoint
{
    private int _displayX;
    private int _displayY;

    public DisplayPoint(int displayX, int displayY)
    {
        _displayX = displayX;
        _displayY = displayY;
    }
}
```

However the methods have not been implemented yet, they will be implemented upon the creation of the `IPicturePoint` interface.

## IPicturePoint

`IPicturePoint` has been implemented, in accordance to [2.1.6.1](#)



```
public interface IPicturePoint
{
    FilePoint ToFilePoint();
    DisplayPoint ToDisplayPoint();
}
```

Note that there has been a minor name revision, from `GetFilePoint` to `ToFilePoint`. This is to more emphasize that this is a conversion from one type of point to another, similar to the `ToString` function.

## FilePoint functions

The remaining functions for FilePoint have been implemented:

```
public FilePoint ToFilePoint() {
    return new FilePoint(_fileX,_fileY);
}

public DisplayPoint ToDisplayPoint() {
    throw new NotImplementedException();
}

public int GetFileValue(Axis axis) {
    switch (axis) {
        case Axis.X:
            return _fileX;
            break;

        case Axis.Y:
            return _fileY;
            break;

        default:
            throw new Exception("Invalid value for Axis");
    }
}
```

However the code to convert from a FilePoint to DisplayPoint will be coded in [Algorithm 2.8AI](#)

## DisplayPoint functions

The remaining functions for DisplayPoint have been implemented:

```
public FilePoint ToFilePoint() {
    throw new NotImplementedException();
}

public DisplayPoint ToDisplayPoint() {
    return new DisplayPoint(_displayX,_displayY);
}

public int GetDisplayValue(Axis axis) {
    switch (axis) {
        case Axis.X:
            return _displayX;

        case Axis.Y:
            return _displayY;

        default:
            throw new Exception("Invalid value for Axis");
    }
}
```

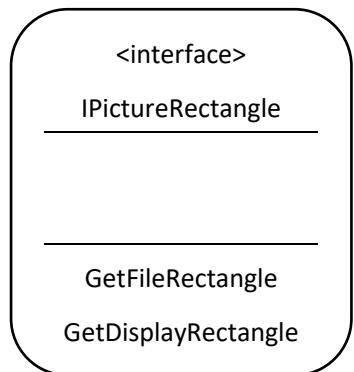
However the code to convert from a DisplayPoint to FilePoint will be coded in [Algorithm 2.8AI](#)

## 26/09/2019 Rectangle Redesign

---

### IPictureRectangle

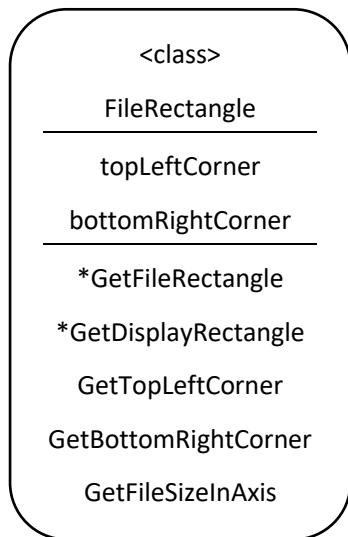
IPictureRectangle has been implemented, in accordance to [2.1.6.4](#)



```
public interface IPictureRectangle
{
    FileRectangle ToFileRectangle();
    DisplayRectangle ToDisplayRectangle();
}
```

## FileRectangle

The FileRectangle class has been implemented in accordance to [2.1.6.5](#)



```
public class FileRectangle : IPictureRectangle
{
    private FilePoint _fileTopLeftCorner;
    private FilePoint _fileBottomRightCorner;

    public FileRectangle(
        int fileTopLeftX, int fileTopLeftY,
        int fileBottomRightX, int fileBottomRightY
    )
    {
        _fileTopLeftCorner = new FilePoint(fileTopLeftX, fileTopLeftY);
        _fileBottomRightCorner = new FilePoint(fileBottomRightX, fileBottomRightY);
    }

    public FileRectangle GetFileRectangle() {
        return new FileRectangle(
            _fileTopLeftCorner.GetFileValue(Axis.X),
            _fileTopLeftCorner.GetFileValue(Axis.Y),
            _fileBottomRightCorner.GetFileValue(Axis.X),
            _fileBottomRightCorner.GetFileValue(Axis.Y)
        );
    }

    public DisplayRectangle GetDisplayRectangle() {
        throw new NotImplementedException();
    }

    public FilePoint GetTopLeftCorner() {
        return _fileTopLeftCorner;
    }

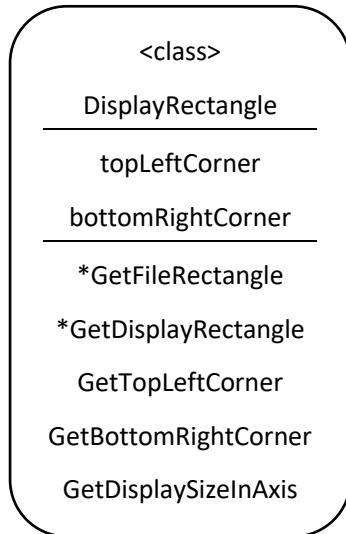
    public FilePoint GetBottomRightCorner() {
        return _fileBottomRightCorner;
    }

    public int GetFileSize(Axis axis) {
        return _fileBottomRightCorner.GetFileValue(axis) - _fileTopLeftCorner.GetFileValue(axis);
    }
}
```

However the conversion to DisplayRectangle has not been implemented yet as the conversion to DisplayPoint has not been implemented.

## DisplayRectangle

The DisplayRectangle class has been implemented in accordance to [2.1.6.6](#)



```
public class DisplayRectangle : IPictureRectangle
{
    private DisplayPoint _displayTopLeftCorner;
    private DisplayPoint _displayBottomRightCorner;

    public DisplayRectangle(
        int displayTopLeftX, int displayTopLeftY,
        int displayBottomRightX, int displayBottomRightY
    )
    {
        _displayTopLeftCorner = new DisplayPoint(displayTopLeftX, displayTopLeftY);
        _displayBottomRightCorner = new DisplayPoint(displayBottomRightX, displayBottomRightY);
    }

    public FileRectangle GetFileRectangle() {
        throw new NotImplementedException();
    }

    public DisplayRectangle GetDisplayRectangle() {
        return new DisplayRectangle(
            _displayTopLeftCorner.GetDisplayValue(Axis.X),
            _displayTopLeftCorner.GetDisplayValue(Axis.Y),
            _displayBottomRightCorner.GetDisplayValue(Axis.X),
            _displayBottomRightCorner.GetDisplayValue(Axis.Y)
        );
    }

    public DisplayPoint GetTopLeftCorner() {
        return _displayTopLeftCorner;
    }

    public DisplayPoint GetBottomRightCorner() {
        return _displayBottomRightCorner;
    }

    public int GetDisplaySize(Axis axis) {
        return _displayBottomRightCorner.GetDisplayValue(axis) - _displayTopLeftCorner.GetDisplayValue(axis);
    }
}
```

However the conversion to FileRectangle has not been implemented yet as the conversion to FilePoint has not been implemented.

## ZoomSettings upgrade

The ZoomSettings class has been upgraded in accordance to [2.1.6.7](#)

```
<class>
ZoomSettings
-----
centreLocation
zoomAmount
```

```
public struct ZoomSettings
{
    public int zoom;
    public FilePoint centreLocation;

    public ZoomSettings(FilePoint centreLocation) {
        this.zoom = 1;
        this.centreLocation = centreLocation;
    }
}
```

With its constructor being implemented in the way that was designed:

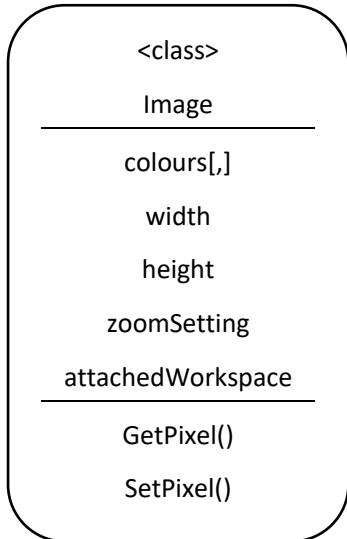
```
constructor(_centreLocation) {
    centreLocation = _centreLocation
    zoomAmount = 1
}
```

zoomAmount has been renamed to **zoom** as the amount part is implicit.

# 27/09/2019 Remaining Class Redesigning

## Image Redesign

The remaining property for Image has been implemented, the attachedWorkspace



```
private Color[,] pixels;
public int width;
public int height;
public ZoomSettings zoomSettings;
public Workspace attachedWorkspace;
```

So thus the constructor has been edited:

```
public Image(int width, int height, Workspace sender)
{
    if (width <= 0) {
        throw new ArgumentException("Width is 0 or less", "Width");
    }
    if (width > SimpConstants.IMAGE_MAX_WIDTH) {
        throw new ArgumentException(String.Format("Width is greater than Image Max ({0})", SimpConstants.IMAGE_MAX_WIDTH), "Width");
    }

    if (height <= 0) {
        throw new ArgumentException("Height is 0 or less", "Height");
    }
    if (height > SimpConstants.IMAGE_MAX_WIDTH) {
        throw new ArgumentException(String.Format("Height is greater than Image Max ({0})", SimpConstants.IMAGE_MAX_WIDTH), "Height");
    }

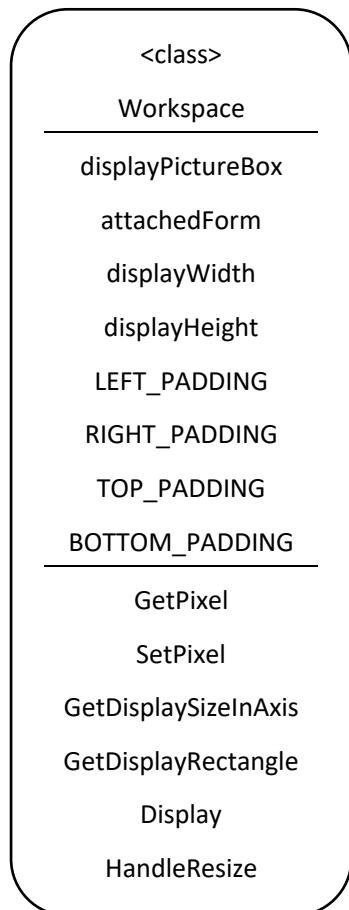
    this.width = width;
    this.height = height;
    this.attachedWorkspace = sender;
    this.pixels = new Color[width, height];
}
```

A displayWidth and displayHeight parameter has been added, which returns the zoomed size of the image:

```
public int displayWidth { get {
    return realWidth * zoomSettings.zoom;
}}
public int displayHeight { get {
    return realHeight * zoomSettings.zoom;
}}
```

## Workspace Redesign

Workspace has been upgraded to include the padding parameters and other functions defined by [2.6.1.8](#)



## Implementing Padding

Padding has now been implemented into the relevant calculations, including ones for defining size of displayBox:

```
private void CalculateDimensions() {
    // Sets the dimensions of the workspace to the dimensions of the form
    width = DisplayRectangle.Width - (leftPadding + rightPadding);
    height = DisplayRectangle.Height - (topPadding + bottomPadding);
}
```

Code for determining locations of displayBox:

```
private void RelocatePictureBox() {
    int X = ((width - image.realWidth) / 2) + leftPadding;
    int Y = ((height - image.realHeight) / 2) + topPadding;

    displayBox.Location = new Point(X,Y);
}
```

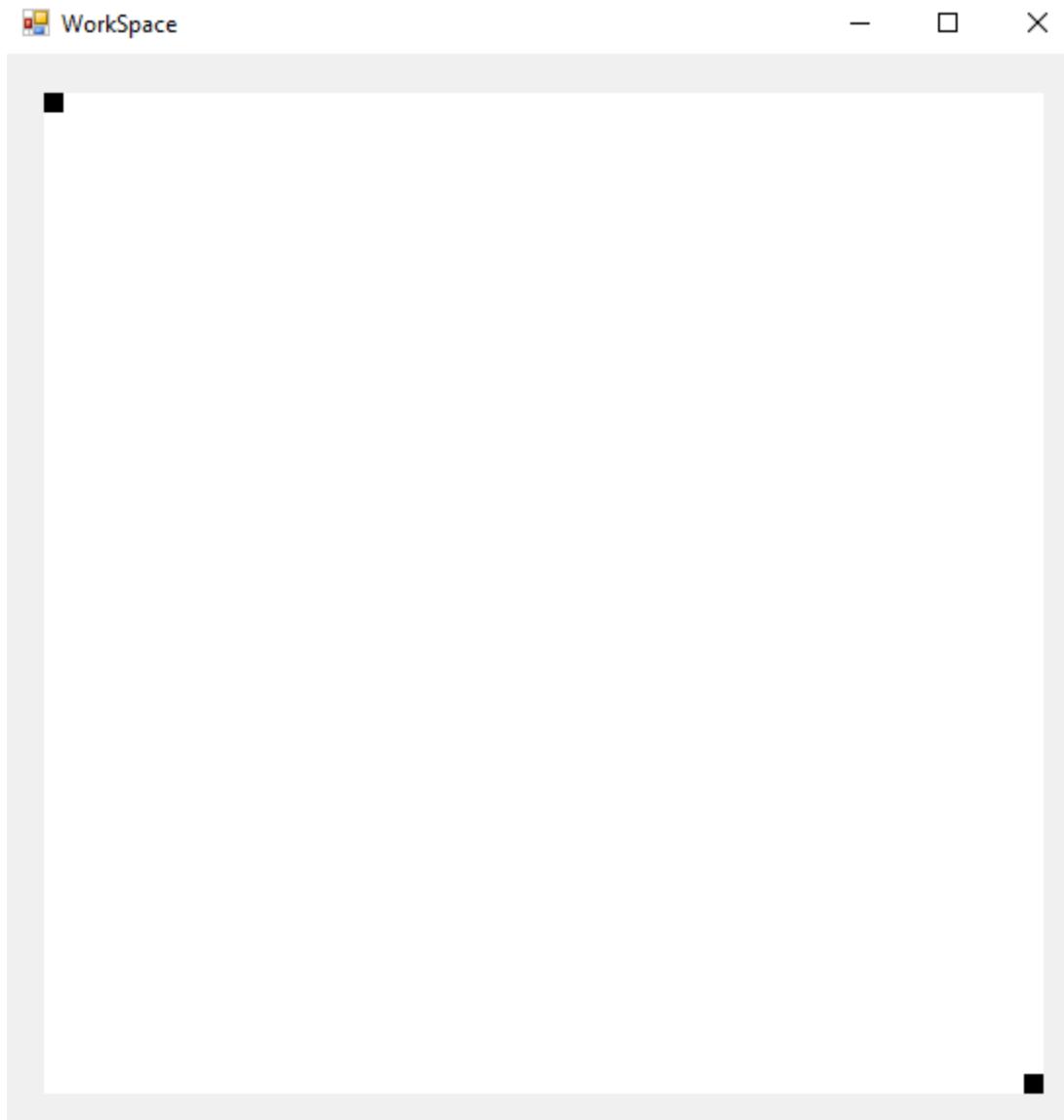
Code for determining minimum size:

```
MinimumSize = new Size(
    width + SimpConstants.WINDOWS_LEFT_BAR_WIDTH + SimpConstants.WINDOWS_RIGHT_BAR_WIDTH+16 + leftPadding + rightPadding,
    height + SimpConstants.WINDOWS_TOP_BAR_HEIGHT + SimpConstants.WINDOWS_BOTTOM_BAR_HEIGHT+16 + topPadding + bottomPadding
);
```

So when the padding is set as part of the constructor:

```
// Defines levels of padding
leftPadding = rightPadding = topPadding = bottomPadding = 20;
```

The resulting program cannot be made smaller than this size:



As there is 20px of padding on each side, to allow space for controls to be placed at the image border.

However, there has been a **change from design**. The padding values are no longer constants, contrary to design:

Property	Datatype	Justification
LEFT_PADDING	(constant) Integer	The amount of padding on the left hand side. Used for calculations involving where to place the picture box
RIGHT_PADDING	(constant) Integer	The amount of padding on the right hand side
TOP_PADDING	(constant) Integer	The amount of padding on the top side
BOTTOM_PADDING	(constant) Integer	The amount of padding on the bottom side

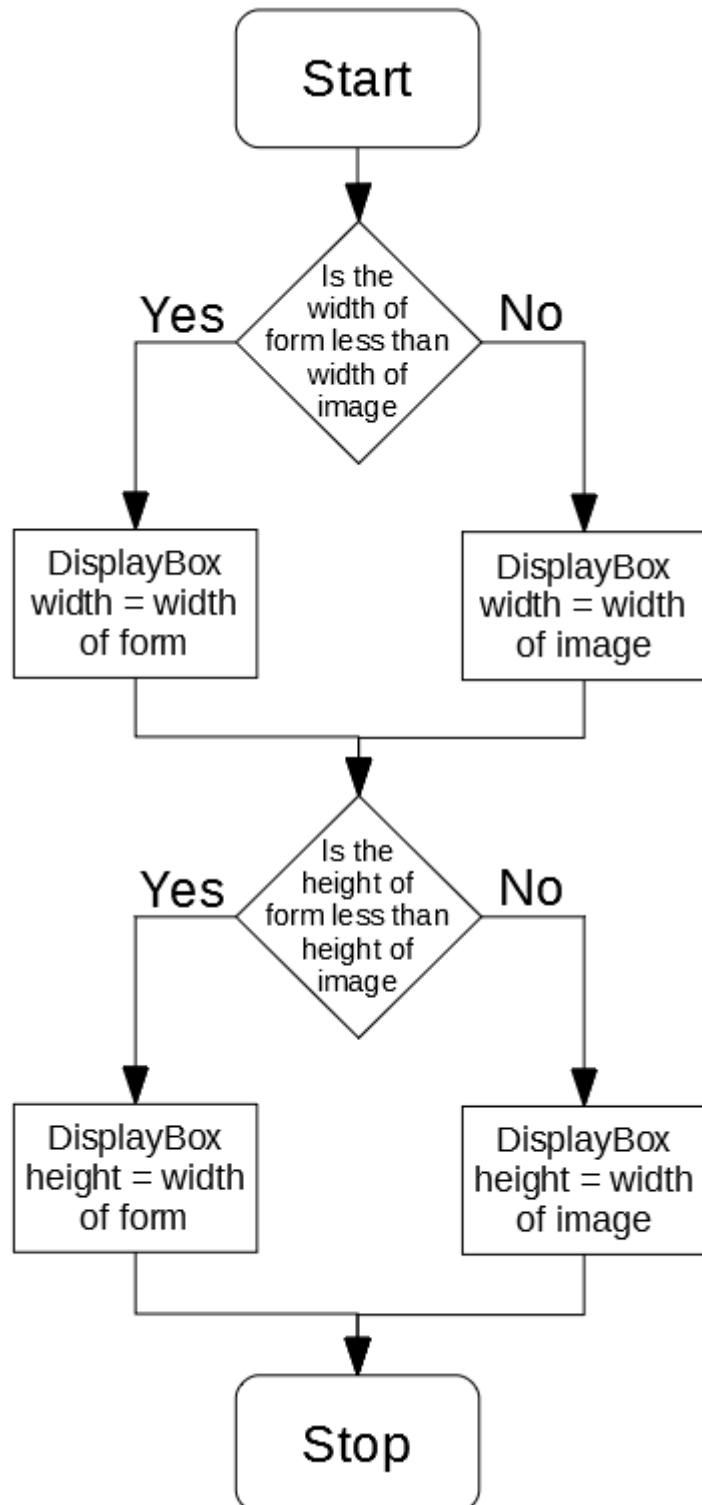
Compared to:

```
public int leftPadding, rightPadding, topPadding, bottomPadding;
```

This means that the amount of padding can be changed at runtime. This adds the possibility to make much more fluid design spaces, where controls can be added or removed from the edge, and the image updates appropriately.

## Relocating displayBox change

As the zoom is now being implemented, it becomes acceptable for the window to be smaller than the pictureBox. This means that the code for changing the displayBox's size must be updated to look like:



## 28/09/2019 Implementing CheckImageSize

---

### EAxisMode

In order to better manage how the image relates to its size in its axis, a new enum has been implemented. It is very simple, containing two options (relating to the decision above)

```
public enum EAxisMode
{
    ImageTooLarge,
    ImageTooSmall
}
```

Using this enum, an algorithm for checking whether the image is too large in a specified axis can be implemented:

### Implementing CheckImageSize

```
private EAxisMode CheckImageSize(EAxis axis) {
    int axisSize;
    int padding;
    switch (axis) {
        case EAxis.X:
            axisSize = DisplayRectangle.Width;
            padding = leftPadding + rightPadding;
            break;
        case EAxis.Y:
            axisSize = DisplayRectangle.Height;
            padding = topPadding + bottomPadding;
            break;
        default:
            throw new Exception("Invalid value for EAxis");
    }

    DisplayRectangle displayImage = image.ToDisplayRectangle();

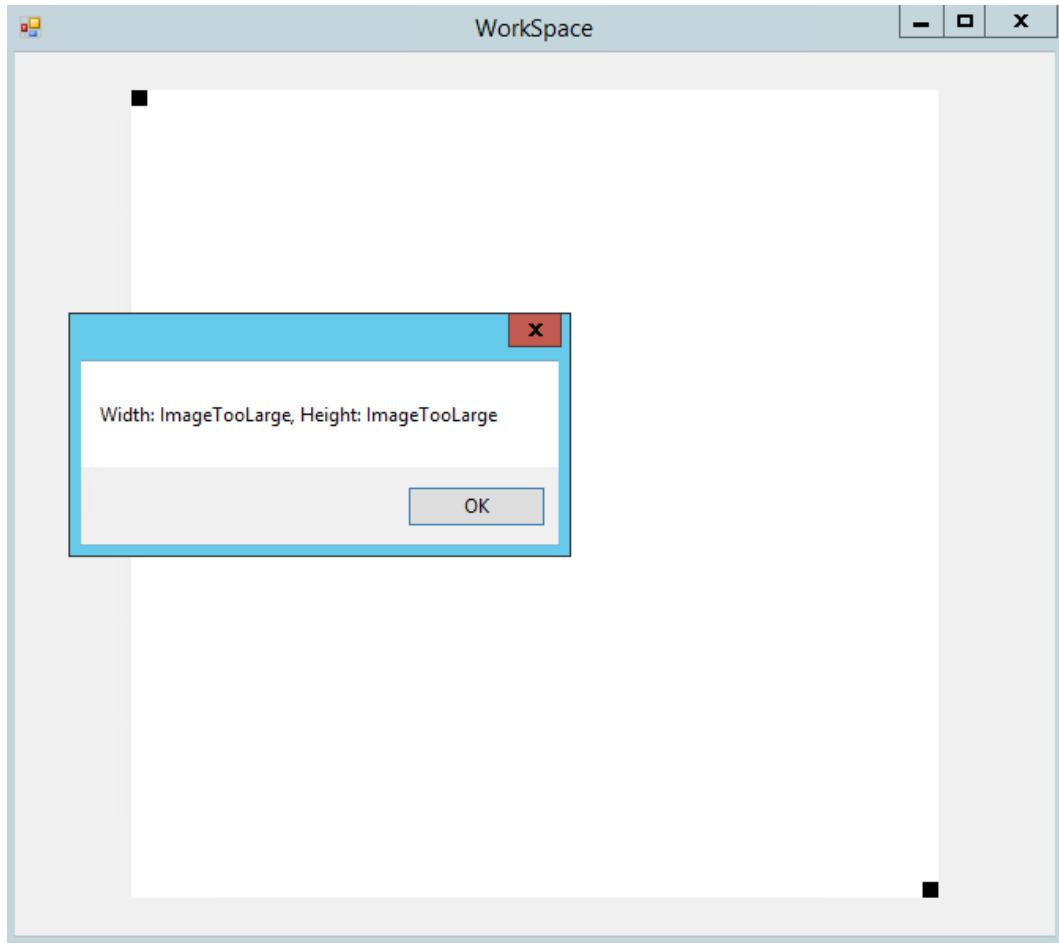
    if (axisSize >= (displayImage.GetDisplaySize(axis) + padding)) {
        return EAxisMode.ImageTooLarge;
    } else {
        return EAxisMode.ImageTooSmall;
    }
}
```

### Alpha Testing CheckImageSize #1

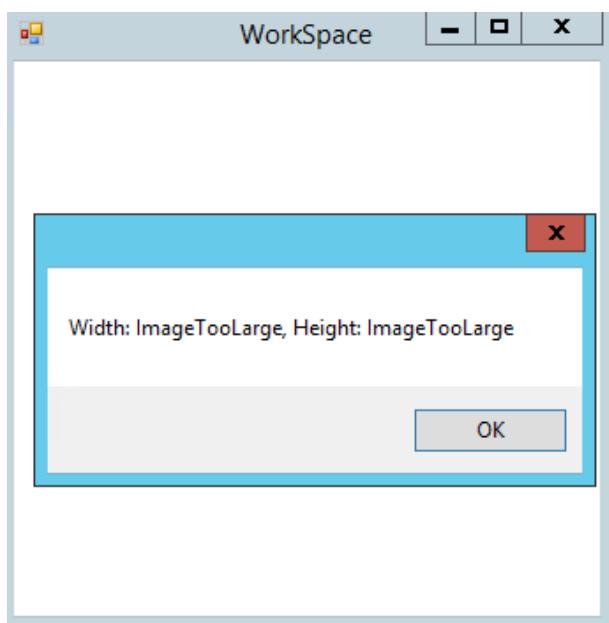
And can be **alpha tested** during development, using the following code:

```
void DisplayBoxClick(object sender, EventArgs e)
{
    MessageBox.Show(string.Format("Width: {0}, Height: {1}", CheckImageSize(EAxis.X), CheckImageSize(EAxis.Y)));
}
```

However, this testing reveals a problem, the code returns ImageTooLarge in situations where it should not:



The correct response should be that the image is too small. Making the form smaller does not change this:



Using a **Break Point** to **debug** the code reveals that:

```
DisplayRectangle displayImage = image.ToDisplayRectangle(  
    int imageAxisSize = displayImage.GetDisplaySize(axis);  
    if (axisSize >= (displayImage.GetDisplaySize(axis) + padd:
```

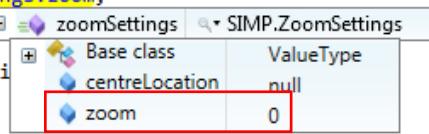
The GetDisplaySize code appears to be returning 0, even when the image should have a width larger than 0.

Through further debugging, it was found that the displayHeight parameter of image (which should return the dimensions of the image in DisplayPixels) was set to 0:

```
rectangle() {  
    0,displayWidth,displayHeight);  
    displayWidth 0
```

Finally the root of the issue was found, the zoom level was erroneously set to 0:

```
public int displayWidth { get {  
    return fileWidth * zoomSettings.zoom;  
}}  
public int displayHeight { get {  
    return fileHeight * zoomSettings.zoom;  
}}  
public Image(int width, int height, Workspace sender)
```



The cause of this was found to be that the constructor for ZoomSettings was made without parameters:

```
this.zoomSettings = new ZoomSettings();
```

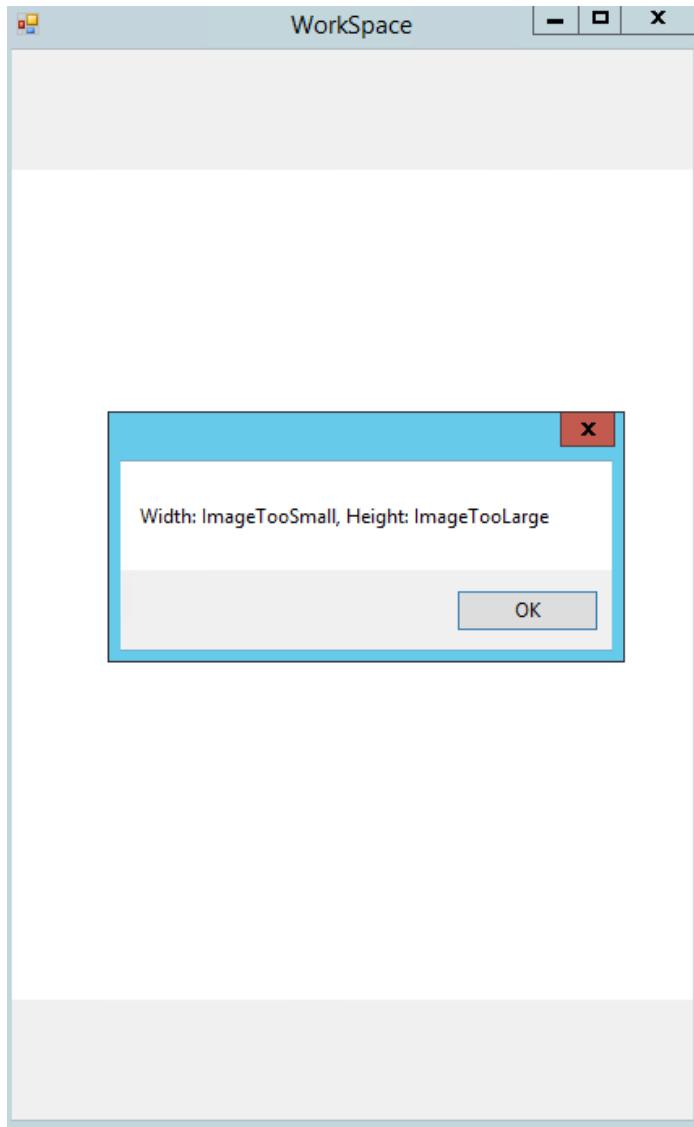
Which causes the class to be populated with null values, which in the case of an integer is 0.

This error can be resolved by making sure that the explicit constructor is called:

```
public ZoomSettings(FilePoint centreLocation) {  
    this.zoom = 1;  
    this.centreLocation = centreLocation;  
    this.zoomSettings = new ZoomSettings(new FilePoint(width/2,height/2));  
}
```

## Alpha Testing CheckImageSize #2

From testing CheckImageSize again, there is a second problem:



The labels for too large and too small are the incorrect way around. A simple switch in the source code fixes this:

```
if (axisSize >= (displayImage.GetDisplaySize(axis) + padding)) {  
    return EAxisMode.ImageTooLarge;  
} else {  
    return EAxisMode.ImageTooSmall;  
}
```



```
if (axisSize >= (displayImage.GetDisplaySize(axis) + padding)) {  
    return EAxisMode.ImageTooSmall;  
} else {  
    return EAxisMode.ImageTooLarge;  
}
```

Thus, by **iterative development and bug fixing, the function has been created.**

## 29/09/2019 Upgrading displayBox functionality

---

### Implementing proper resizing and relocating for displayBox

The code for correctly resizing the displayBox can now be implemented, according to the above flowchart:

```
private void ResizeDisplayBox() {
    switch (CheckImageSize(EAxis.X)) {
        // if the image is larger than form size
        case EAxisMode.ImageTooLarge:
            displayBox.Width = DisplayRectangle.Width - (leftPadding + rightPadding);
            break;
        // if the image is smaller than form size
        case EAxisMode.ImageTooSmall:
            displayBox.Width = image.displayWidth;
            break;
    }

    switch (CheckImageSize(EAxis.Y)) {
        // if the image is larger than form size
        case EAxisMode.ImageTooLarge:
            displayBox.Height = DisplayRectangle.Height - (topPadding + bottomPadding);
            break;
        // if the image is smaller than form size
        case EAxisMode.ImageTooSmall:
            displayBox.Height = image.displayHeight;
            break;
    }
}
```

The switch cases take the role of the Diamond shape in the flowchart.

```
private void RelocateDisplayBox() {
    int X = 0;
    int Y = 0;
    switch (CheckImageSize(EAxis.X)) {
        // if the image is larger than form size
        case EAxisMode.ImageTooLarge:
            X = leftPadding;
            break;
        // if the image is smaller than form size
        case EAxisMode.ImageTooSmall:
            X = ((width - image.fileWidth) / 2) + leftPadding;
            break;
    }

    switch (CheckImageSize(EAxis.Y)) {
        // if the image is larger than form size
        case EAxisMode.ImageTooLarge:
            Y = topPadding;
            break;
        // if the image is smaller than form size
        case EAxisMode.ImageTooSmall:
            Y = ((height - image.fileHeight) / 2) + topPadding;
            break;
    }
    displayBox.Location = new Point(X,Y);
}
```

## Implementing new Image Requests

To help save CPU at this time, a new image will only be requested for the displayBox at the end of a Resize. This is enforced by the use of a Boolean when calling UpdateDisplayBox:

```
public void UpdateDisplayBox(bool redraw) {
    // Sets up the dimensions of displayBox
    ResizeDisplayBox();

    // Relocates the picture box
    RelocateDisplayBox();

    if (redraw) {
        // Displays temporarily to picture box
        displayBox.Image = image.GetDisplayImage();
    }
}
```

So this means that a new image will only be requested when the function is passed 'True'. This does **not** happen on a normal resize, but does happen when the resize ends:

```
private void HeartbeatTick(object sender, EventArgs e)
{
    if (HasSizeChanged()) {
        CalculateDimensions();
        UpdateDisplayBox(false);
    }
}

private void WorkspaceResizeEnd(object sender, EventArgs e)
{
    UpdateDisplayBox(true);
}
```

One change will be made to the GetDisplayImage function, which is to include height and width parameters, to tell the function the size of the image it wants.

```
public System.Drawing.Image GetDisplayImage(int width, int height) {
    Bitmap newImage = new Bitmap(width, height);
    Graphics GFX = Graphics.FromImage(newImage);

    GFX.FillRectangle(new SolidBrush(Color.Red), 0, 0, width, height);

    return newImage;
}
```

Right now a temporary red image is returned, soon image drawing will be implemented.

```
if (redraw) {
    // Displays temporarily to picture box
    displayBox.Image = image.GetDisplayImage(displayBox.Width, displayBox.Height);
}
```

## 1/10/2019 Conversions between pixel types

Before the image display code can be developed, the algorithms for converting between pixel types must be implemented, in accordance to algorithm 2.8A

```
FilePointToDisplayPoint(filePoint) {  
    displacementX = filePoint.X - centreFilePoint.X  
    displacementY = filePoint.Y - centreFilePoint.Y  
  
    displacementX = displacementX * zoom  
    displacementY = displacementY * zoom  
  
    newX = centreDisplayPoint.X + displacementX  
    newY = centreDisplayPoint.Y + displacementY  
  
    return new Point(newX, newY)  
}
```

However there is an issue with this code. This code is being implemented inside of the FilePoint class, which does not have any knowledge of the image. This means that references to the centreLocation aren't possible.

To do this, a new private parameter was added to the image, which contains a reference to the zoom settings of its image:

```
private int _fileX;  
private int _fileY;  
private ZoomSettings _myZoomSettings;
```

However zoomSettings does not contain any knowledge of where the display centre location is. This means that a parameter for this must be added to ZoomSettings:

```
public int zoom;  
public FilePoint fileCentreLocation;  
public DisplayPoint displayCentreLocation;
```

This is updated whenever a new image is requested (whenever the current displayCentreLocation might change).

```
public System.Drawing.Image GetDisplayImage(int width, int height) {  
    Bitmap newImage = new Bitmap(width, height);  
    Graphics GFX = Graphics.FromImage(newImage);  
    zoomSettings.displayCentreLocation = new DisplayPoint(width/2, height/2);  
  
    GFX.FillRectangle(new SolidBrush(Color.Red), 0, 0, width, height);  
  
    return newImage;  
}
```

This means the code for converting from a FilePoint to DisplayPoint can be implemented:

```
public DisplayPoint ToDisplayPoint() {
    int displacementX = _fileX - _myZoomSettings.fileCentreLocation._fileX;
    int displacementY = _fileY - _myZoomSettings.fileCentreLocation._fileY;

    displacementX *= _myZoomSettings.zoom;
    displacementY *= _myZoomSettings.zoom;

    return new DisplayPoint(
        _myZoomSettings.displayCentreLocation.GetDisplayValue(EAxis.X) + displacementX,
        _myZoomSettings.displayCentreLocation.GetDisplayValue(EAxis.X) + displacementY
    )
}
```

The code for converting from DisplayPoint to FilePoint can also be implemented, in accordance to 2.8B.

```
public FilePoint ToFilePoint() {
    int displacementX = _displayX - _myZoomSettings.displayCentreLocation._displayX;
    int displacementY = _displayY - _myZoomSettings.displayCentreLocation._displayY;

    displacementX /= _myZoomSettings.zoom;
    displacementY /= _myZoomSettings.zoom;

    return new FilePoint(
        displacementX + _myZoomSettings.fileCentreLocation.GetFileValue(EAxis.X),
        displacementY + _myZoomSettings.fileCentreLocation.GetFileValue(EAxis.Y)
    );
}
```

## 2/10/2019 Testing Point conversions

### File Point to Display Point testing

From this code, some testing can be completed, where image has a size of 20fpx by 10fpx and the zoom centre is at (10fpx, 5fpx) and zoom is at 2x:

```
public System.Drawing.Image GetDisplayImage(int width, int height) {
    Bitmap newImage = new Bitmap(width, height);
    Graphics GFX = Graphics.FromImage(newImage);
    zoomSettings.displayCentreLocation = new DisplayPoint(width/2, height/2);

    GFX.FillRectangle(new SolidBrush(Color.Red), 0, 0, width, height);

    this.zoomSettings.zoom = 2;
    FilePoint inputPoint = new FilePoint(10, 5);
    DisplayPoint outputPoint = FilePointToDisplayPoint(inputPoint);

    outputPoint = outputPoint;

    return newImage;
```

The data from the break point will be inspected to find the properties of the returned point.

Test	ID	Expected Result	Actual Result	Comment
DisplayPoint(10,5)	1	(5,5)		The location of the centre of the image is returned
DisplayPoint(8,3)	2	(1,1)		The top-left is returned
DisplayPoint(12,7)	3	(9,9)		The bottom-right is returned
DisplayPoint(7,2)	4	(-1,-1)		A location off the edge of the image is returned
DisplayPoint(13,8)	5	(11,11)		A location off the edge of the image is returned
DisplayPoint(0,0)	6	(-15,-5)		The edge of the image is returned
DisplayPoint(19,9)	7	(23,13)		The edge of the image is returned

<i>DisplayPoint(-1,-1)</i>	8	Throws out of bounds error	<table border="1"> <tr> <td>Base class</td><td>Object</td></tr> <tr> <td>displayX</td><td>-12</td></tr> <tr> <td>displayY</td><td>-7</td></tr> </table>	Base class	Object	displayX	-12	displayY	-7	No error is thrown
Base class	Object									
displayX	-12									
displayY	-7									
<i>DisplayPoint(20,10)</i>	9	Throws out of bounds error	<table border="1"> <tr> <td>Base class</td><td>Object</td></tr> <tr> <td>displayX</td><td>30</td></tr> <tr> <td>displayY</td><td>15</td></tr> </table>	Base class	Object	displayX	30	displayY	15	No error is thrown
Base class	Object									
displayX	30									
displayY	15									

Fixing Error #8 & #9

To fix this, a simple range check can be implemented:

```
private DisplayPoint FilePointToDisplayPoint(FilePoint filePoint) {
    if (
        filePoint.fileX < 0 || filePoint.fileX >= fileWidth
        || filePoint.fileY < 0 || filePoint.fileX >= fileHeight
    ) {
        throw new IndexOutOfRangeException();
    }
}
```

	<i>Test</i>	<i>ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<i>DisplayPoint(-1,-1)</i>	8	Throws out of bounds error	System.IndexOutOfRangeException: Index was outside the bounds of the array.	System.IndexOutOfRangeException: Index was outside the bounds of the array.	No error is thrown
<i>DisplayPoint(20,10)</i>	9	Throws out of bounds error	System.IndexOutOfRangeException: Index was outside the bounds of the array.	System.IndexOutOfRangeException: Index was outside the bounds of the array.	No error is thrown

## Display Point to File Point testing

Test	ID	Expected Result	Actual Result	Comment
<i>FilePoint(5,5)</i>	1	(10,5)	 Base class  fileX  fileY	Object 10 5 The centre location is returned
<i>FilePoint(1,1)</i>	2	(8,3)	 Base class  fileX  fileY	Object 8 3 The correct point is returned
<i>FilePoint(12,7)</i>	3	(9,9)	 Base class  fileX  fileY	Object 13 6 The correct point is returned
<i>FilePoint(-1,-1)</i>	4	(7,2)	 Base class  fileX  fileY	Object 7 2 The correct point is returned
<i>FilePoint(11,11)</i>	5	(13,8)	 Base class  fileX  fileY	Object 13 8 The correct point is returned
<i>FilePoint(-15,-5)</i>	6	(0,0)	 Base class  fileX  fileY	Object 0 0 The correct point is returned
<i>FilePoint(23,13)</i>	7	(19,9)	 Base class  fileX  fileY	Object 19 9 The correct point is returned
<i>FilePoint(6,6)</i>	8	(10,5)	 Base class  fileX  fileY	Object 10 5 The point is rounded up
<i>FilePoint(4,4)</i>	9	(9,4)	 Base class  fileX  fileY	Object 10 5 The point is not correctly rounded down
<i>FilePoint(-17,-7)</i>	10	This point is rejected	System.IndexOutOfRangeException: Index was outside the bounds of the array.	The point is correctly rejected
<i>FilePoint(25,15)</i>	11	This point is rejected	System.IndexOutOfRangeException: Index was outside the bounds of the array.	The point is correctly rejected

## Fixing Issue #9

The issue in error #9 occurs because, in this situation, the displacement has a negative value. When this displacement is divided and then rounded, it is rounded **up**, as this is how negative numbers are implemented.

```
displacementX /= zoomSettings.zoom;
displace displacementX -1
```

```
displacementX /= zoomSettings.zoom;
displace displacementX 0
```

As can be seen, when the displacement is seen to be divided by the zoom, it goes from -1 to 0, expected result would be -1 (rounded down) but it is instead rounded up.

In order to fix this, the displacement has been changed to be stored as a float, and then converted back into an integer at the end, after the adding. This makes sure only a positive number is rounded, so the system works.

```
private FilePoint DisplayPointToFilePoint(DisplayPoint displayPoint) {
    float displacementX = displayPoint.displayX - zoomSettings.displayCentreLocation.displayX;
    float displacementY = displayPoint.displayY - zoomSettings.displayCentreLocation.displayY;

    displacementX /= zoomSettings.zoom;
    displacementY /= zoomSettings.zoom;

    return new FilePoint(
        (int)(displacementX + (float)zoomSettings.fileCentreLocation.fileX),
        (int)(displacementY + (float)zoomSettings.fileCentreLocation.fileY)
    );
}
```

Test	ID	Expected Result	Actual Result	Comment						
FilePoint(4,4)	9	(9,4)	<table><tr><td>Base class</td><td>Object</td></tr><tr><td>fileX</td><td>9</td></tr><tr><td>fileY</td><td>4</td></tr></table>	Base class	Object	fileX	9	fileY	4	The point is now correctly rounded down
Base class	Object									
fileX	9									
fileY	4									

## 3/10/2019 Displaying an Image

---

Now the code for displaying pixels in the image can be implemented.

### Algorithm 2.8B – Determining Border Locations

The algorithm for determining the border location has been implemented, in accordance to 2.8B

```
// finds locations of where it needs to draw to and from
FilePoint topLeftOrangeCross = new FilePoint(
    zoomSettings.fileCentreLocation.fileX - (int)Math.Ceiling(Math.Floor((decimal)width/(decimal)zoomSettings.zoom)/2),
    zoomSettings.fileCentreLocation.fileY - (int)Math.Ceiling(Math.Floor((decimal)height/(decimal)zoomSettings.zoom)/2)
);
FilePoint bottomRightOrangeCross = new FilePoint(
    zoomSettings.fileCentreLocation.fileX + (int)Math.Ceiling(Math.Floor((decimal)width/(decimal)zoomSettings.zoom)/2),
    zoomSettings.fileCentreLocation.fileY + (int)Math.Ceiling(Math.Floor((decimal)height/(decimal)zoomSettings.zoom)/2)
);
```

A few extra decimal conversions are needed, but otherwise no major changes needed to be made.

### Algorithm 2.8C – Finding Green Pixels

The algorithm for determining pixels to draw has been implemented, in accordance to 2.8C

```
// loops through all the necessary pixels
for (int x = topLeftPoint.fileX; x < bottomRightPoint.fileX; x++) {
    for (int y = topLeftPoint.fileY; y < bottomRightPoint.fileY; y++) {
        // draw zoomed part
    }
}
```

Where the comment will be replaced with the code necessary to draw the zoomed part

### Algorithm 2.9 – Draw Green Pixels

The above code has been upgraded to include drawing functionality:

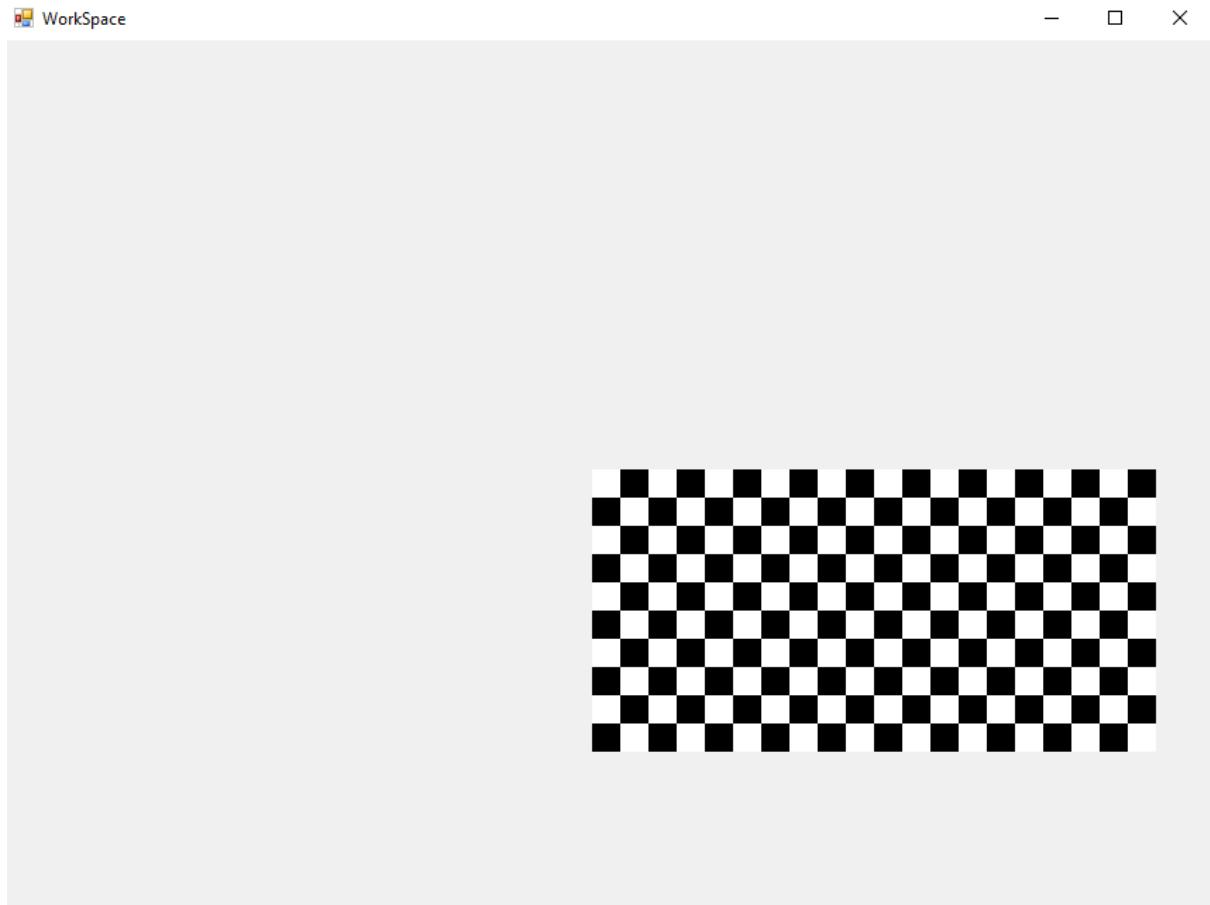
```
// loops through all the necessary pixels
for (int x = topLeftPoint.fileX; x < bottomRightPoint.fileX; x++) {

    // resets the Y (before it is iterated on)
    currentPoint.displayY = topLeftDisplayPoint.displayY;
    for (int y = topLeftPoint.fileY; y < bottomRightPoint.fileY; y++) {
        // Draws a rectangle using colour of current pixel, X and Y of current display location, the width and height is the zoom of the image
        GFX.FillRectangle(new SolidBrush(pixels[x,y]),currentPoint.displayX,currentPoint.displayY,zoomSettings.zoom, zoomSettings.zoom);

        // Moves the current Y along by the size of one pixel
        currentPoint.displayY += zoomSettings.zoom;
    }

    // Moves the current X along for same reason
    currentPoint.displayX += zoomSettings.zoom;
}
```

However there is a problem, the resulting image is not correctly centred:

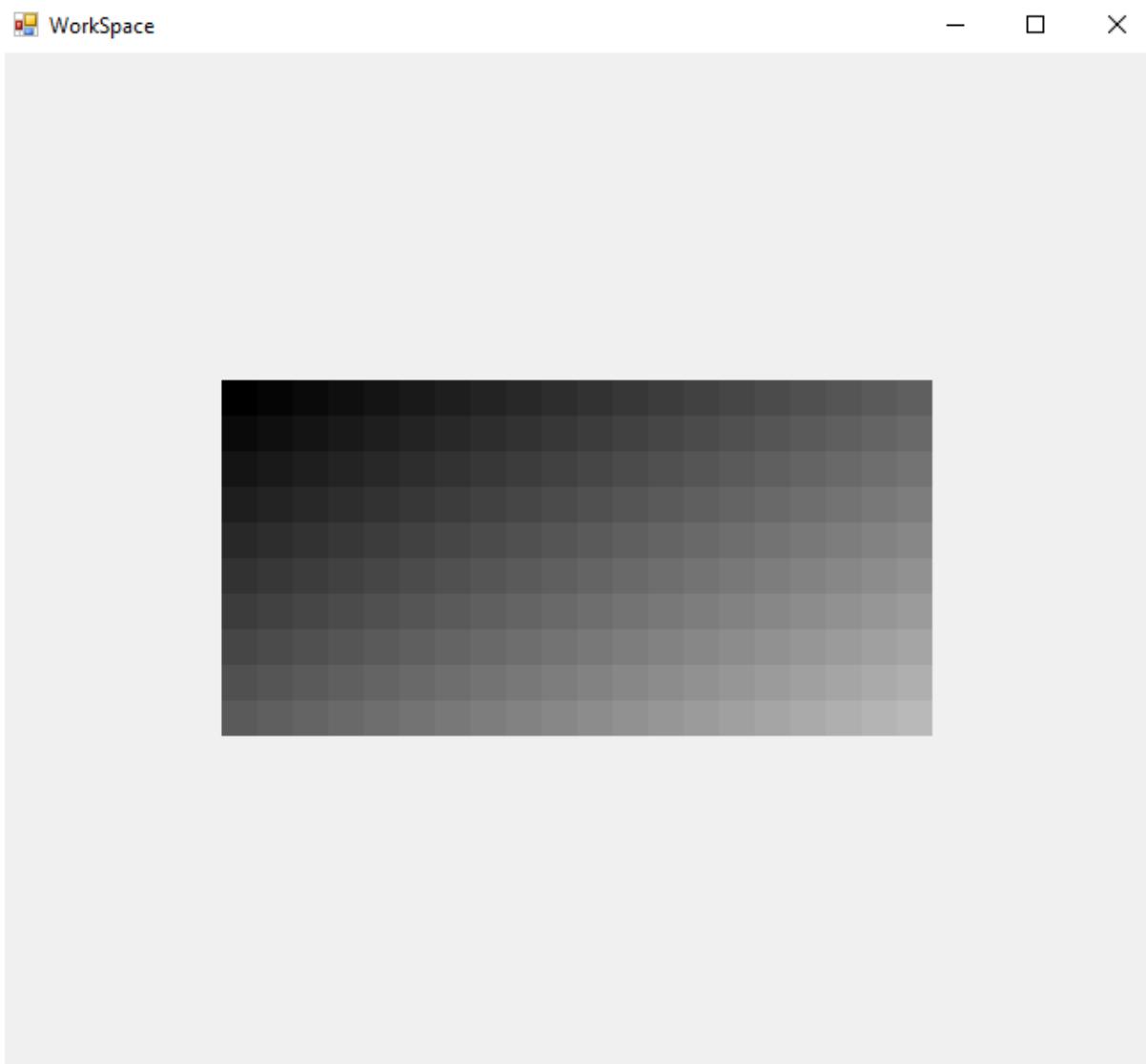


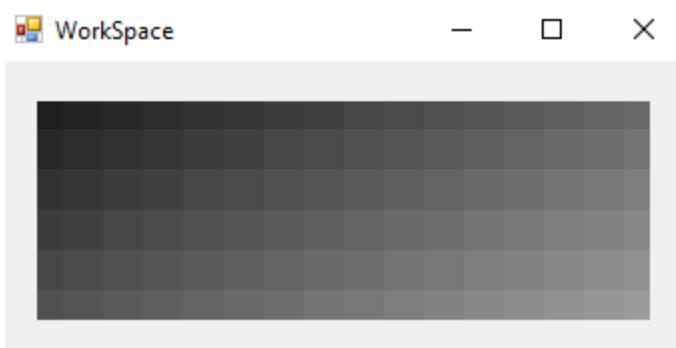
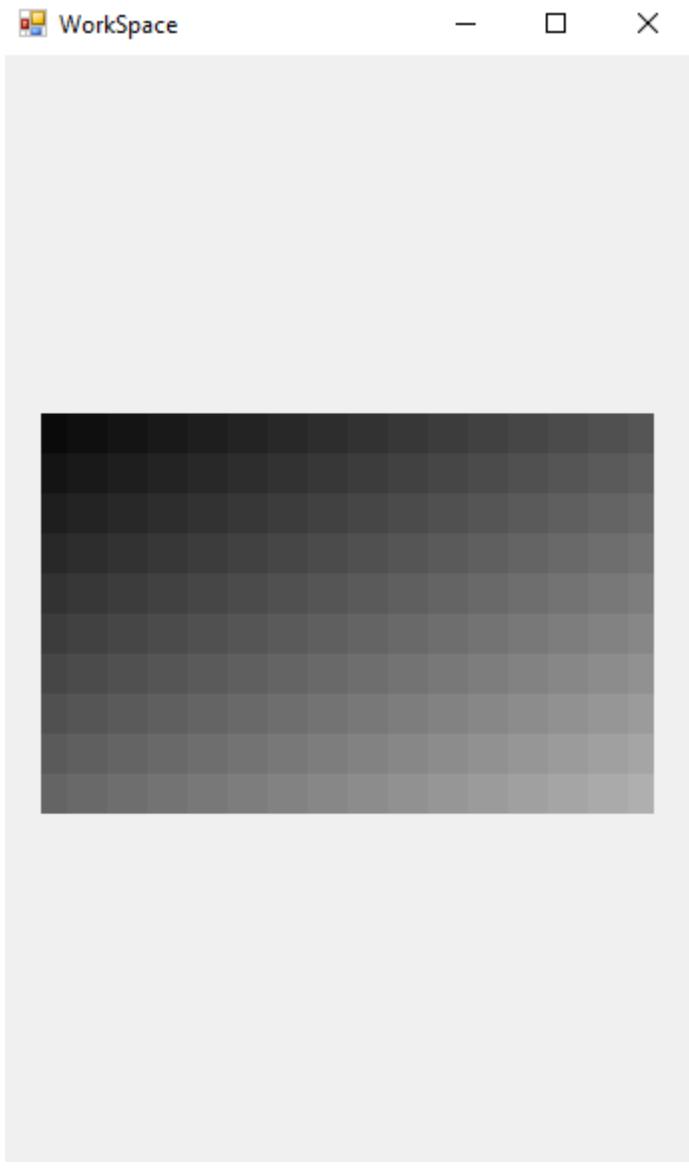
The cause of this was simple, fileWidth was erroneously used instead of displayWidth:

```
private void RelocateDisplayBox() {
    int X = 0;
    int Y = 0;
    switch (CheckImageSize(EAxis.X)) {
        // if the image is larger than form size
        case EAxisMode.ImageTooLarge:
            X = leftPadding;
            break;
        // if the image is smaller than form size
        case EAxisMode.ImageTooSmall:
            X = ((width - image.fileWidth) / 2) + leftPadding;
            break;
    }
}
```

Changing this to displayWidth resolves the error.

SIMP can now display images:

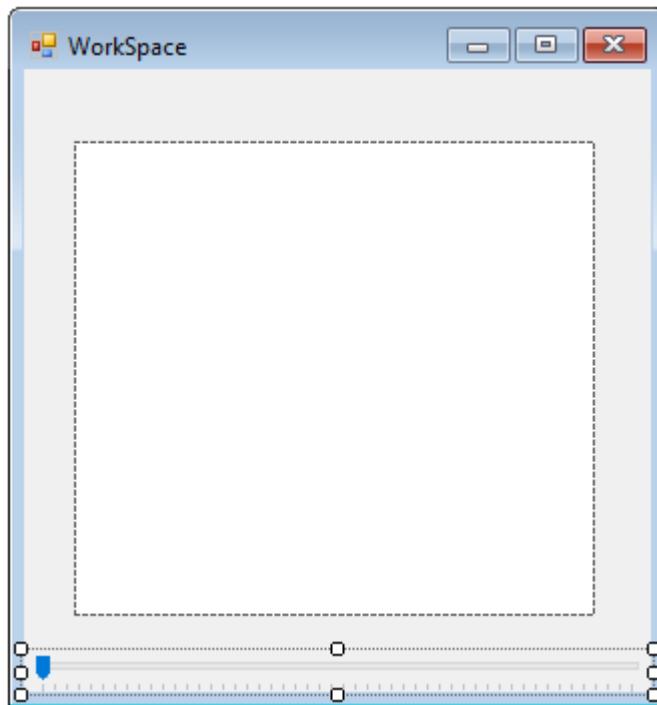




*Some example of SIMP displaying images, even when the display form is much smaller than the image*

## Adding Zoom Bar to GUI

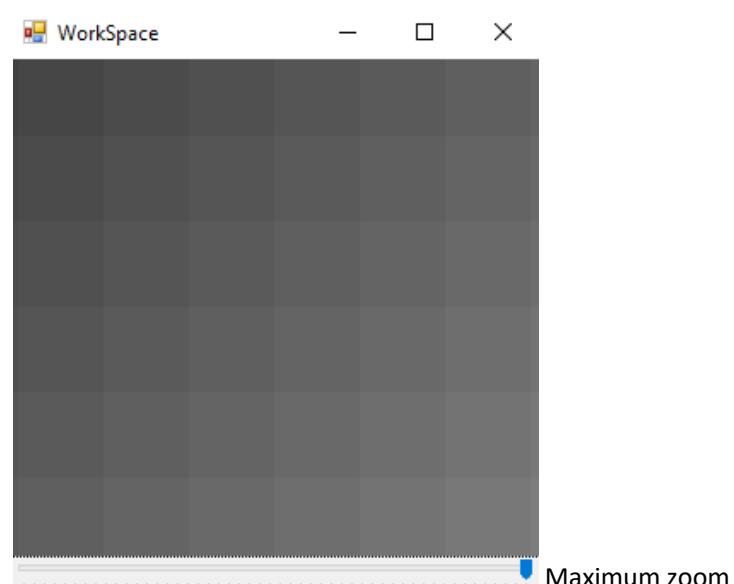
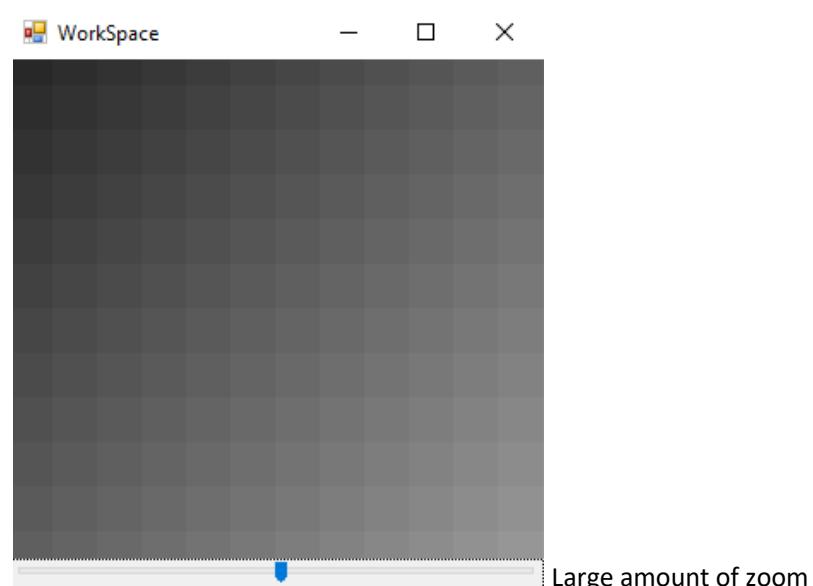
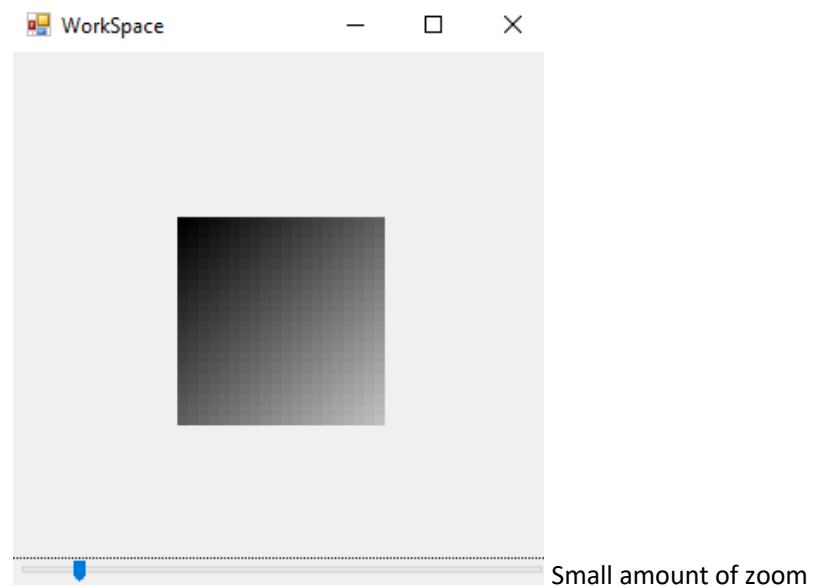
The zoom bar can be added quite simply, filling the border area at the bottom.



Its value can also be interpreted quite simply. Its minimum and maximum has been set to 1 and 50 respectively. When its value is changed, the image's zoom is updated and the box is redrawn.

```
void BarZoomScroll(object sender, EventArgs e)
{
    image.zoomSettings.zoom = barZoom.Value;
    UpdateDisplayBox(true);
}
```

This means the zoom level can be changed using the bar:



# 04/10/2019 Implementing Bars

## Determining whether Bars are visible

Thankfully a function to determine whether the bars should be visible has already been implemented, when [CheckImageSize was implemented](#).

This means the function can be reused due to a **DRY** (Don't Repeat Yourself) programming methodology, as the two functions have the same purpose.

The completed check for this looks like:

```
private void SetBarVisibility(FAxis axis, ScrollBar bar) {
    switch (CheckImageSize(axis)) {
        case FAxisMode.ImageTooLarge:
            bar.Enabled = true;
            break;
        case FAxisMode.ImageTooSmall:
            bar.Enabled = false;
            break;
    }
}
```

## Algorithm 2.11A Determining Crosses in Axis

This code has been implemented, in accordance to [Algorithm 2.11A](#)

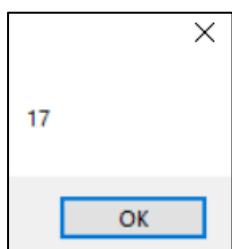
```
DetermineCrossesInAxis(Axis axis) {
    MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)*2
    MaxCrossesInAxis = image.getSizeInAxis(axis)
    return MaxCrossesInAxis - MissingCrossesInAxis
}

private int DetermineCrossesInAxis(int axisSize, int imageSize) {
    int missingCrossesInAxis = (int) Math.Floor(Math.Floor((decimal)axisSize/(decimal)zoomSettings.zoom)/2)*2;
    return (imageSize+1) - missingCrossesInAxis;
}
```

When testing the code with the following inputs:

```
zoomSettings.zoom = 2;
MessageBox.Show(DetermineCrossesInAxis(10,20).ToString());
```

The output is:



Which is what was expected in design.

## Algorithm 2.11B Setting Bar Size

The algorithm to resize the bars has been implemented, in accordance to [Algorithm 2.11B](#):

```
UpgradedSetupBarSize(progressBar, Axis) {  
    MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)*2  
    progressBar.barSize = MissingCrossesInAxis  
    progressBar.max = image.getFileWidth(Axis)  
}  
  
public void SetBarValues(ScrollBar barHorizontal, ScrollBar barVertical, int width, int height) {  
    // determines how many missing centre locations there are  
    int missingCrossexXAxis = MissingCrossesInAxis(width,fileWidth);  
    int missingCrossexYAxis = MissingCrossesInAxis(height,fileHeight);  
  
    // updates horizontal bar  
    barHorizontal.LargeChange = missingCrossexXAxis;  
    barHorizontal.Maximum = fileWidth;  
  
    // updates vertical bar  
    barVertical.LargeChange = missingCrossexYAxis;  
    barVertical.Maximum = fileHeight;  
}
```

However the Axis is not used here, each bar is simply updated at the same time.

### Algorithm 2.11C Determining Centre Location from Bar Value

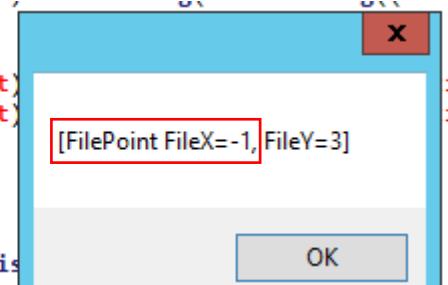
The code for determining a centre location when the bar value has been changed has been implemented, in accordance to [Algorithm 2.11C](#):

```
GetCentreLocationFromBar(progressBar bar, Axis axis) {  
    firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)  
    RETURN bar.value + firstCrossPosition  
}
```

**Insert screenshot here since my laptop is too small to view it properly**

However, when attempting to change the value of a bar, an `IndexOutOfBoundsException` is thrown. The cause of the error is found to be, when rendering, the `FileX` could be taken out of bounds

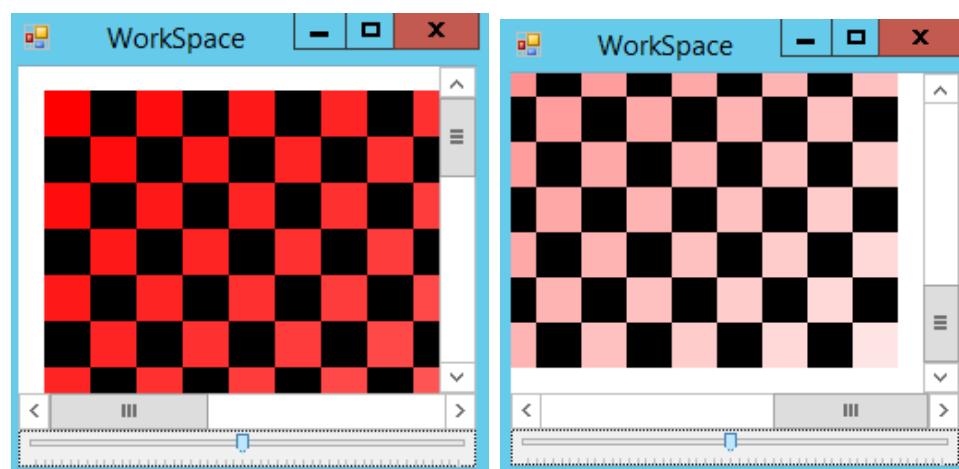
```
);  
FilePoint bottomRightPoint = new FilePoint(  
    zoomSettings.fileCentreLocation.fileX + (int)  
    zoomSettings.fileCentreLocation.fileY + (int)  
);  
  
// the file location of the top right  
MessageBox.Show(topLeftPoint.ToString());  
DisplayPoint topLeftDisplayPoint = FilePointToDis
```



This is expected, as the allowed region for centres is 1 fpixel less than the imagined viewing region. In order to solve this a function for clamping points can be implemented:

```
private FilePoint ClampFilePoint(FilePoint filePoint) {  
    if (filePoint.fileX < 0) {  
        filePoint.fileX = 0;  
    }  
    if (filePoint.fileX >= fileWidth) {  
        filePoint.fileX = fileWidth-1;  
    }  
  
    if (filePoint.fileY < 0) {  
        filePoint.fileY = 0;  
    }  
    if (filePoint.fileY >= fileHeight) {  
        filePoint.fileY = fileHeight-1;  
    }  
}
```

This means that the part of the image that is viewed can now be manipulated:



## 05/10/2019 Remaining Bar code

### Algorithm 2.11D Determining Bar Value from Centre Location

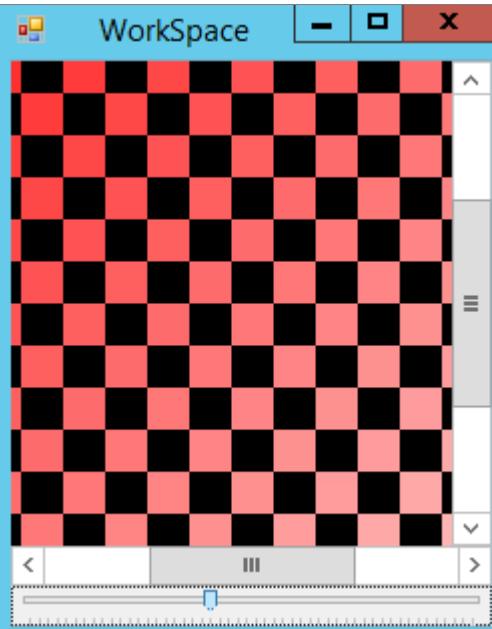
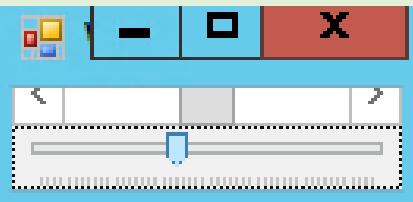
The code for determining the Bar Value from a given Centre Location has been implemented, in accordance to **Algorithm 2.11D**:

```
SetBarFromCentreLocation(progressBar bar, Axis axis) {  
    firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)  
    bar.value = centreLocation.getSizeInAxis(axis) - firstCrossPosition  
}
```

Again I can't add the screenshot, do this later please

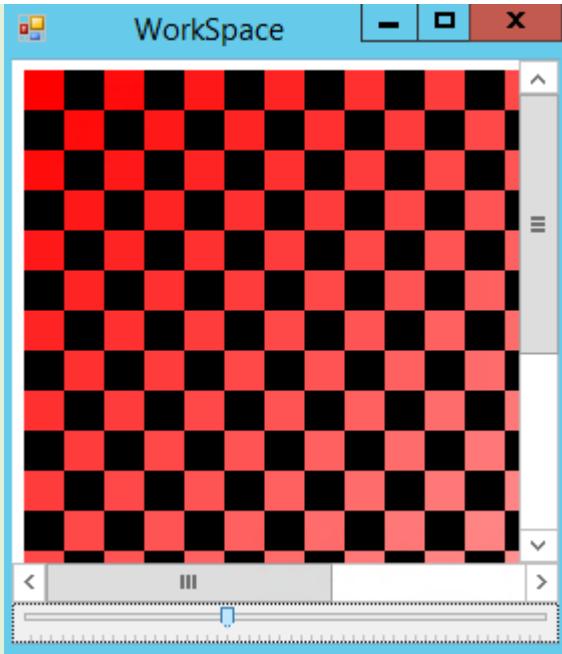
### Unit Test 2.11 #1

Now that the bars have been added, they need to be tested to make sure that they are fully functional.

Test	ID	Expected Result	Actual Result	Comment
Form is resized to half the size of image	1	Bar is half size of bounds and in its centre position		The bars are displaying the correct information
Form is resized to its smallest position	2	Bar is small but still central.	 <p>System.ArgumentException: Parameter is not valid.</p>	An error is thrown as the image becomes too small to display.
Form is resized back to half size of image	3	Bar is half size of bounds and in its centre position	Cannot be tested as relies on success of previous test	

*Horizontal Scroll bar is moved to far left*

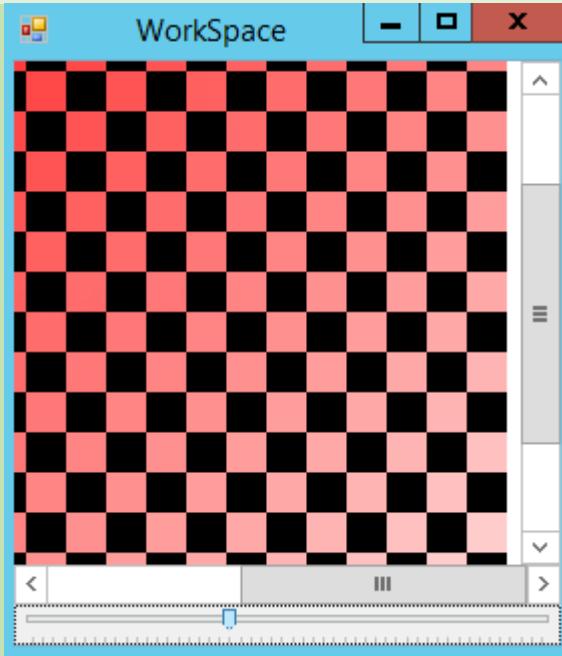
- 4 The far left of the image is displayed, but no more



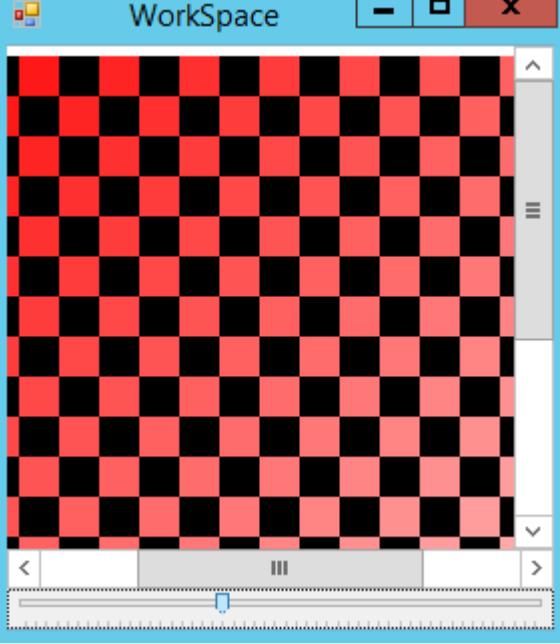
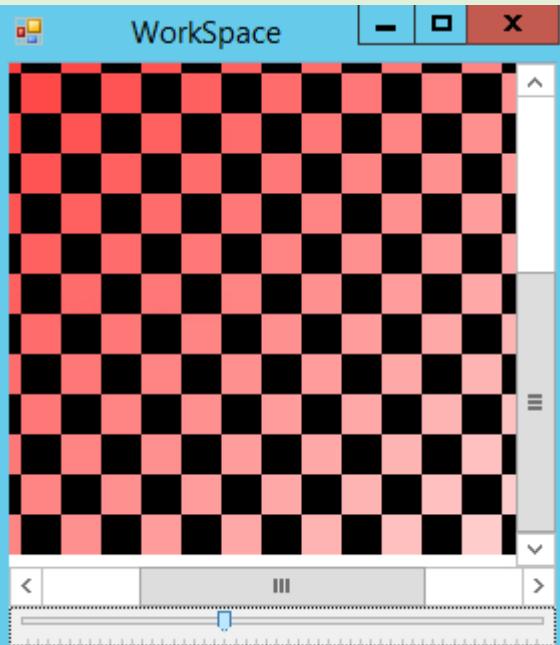
The far left of the image is displayed

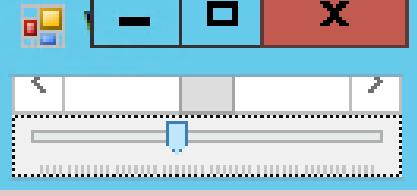
*Horizontal Scroll bar is moved to far right*

- 5 The far right of the image is displayed, but no more



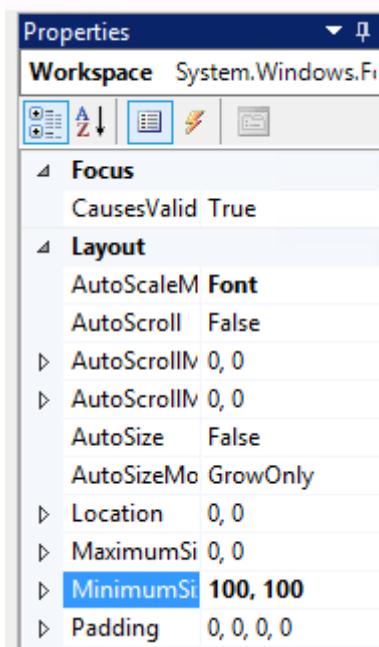
The far right of the image is displayed

<i>Vertical Scroll bar is moved to highest</i>	6    The highest of the image is displayed, but no more		The top of the image is displayed
<i>Vertical Scroll bar is moved to lowest</i>	7    The lowest of the image is displayed, but no more		The bottom of the image is displayed
<i>Horizontal scroll bar is moved far left, then form size increased</i>	8    Centre locations is moved to the left when needed	System.ArgumentOutOfRangeException: Value of '-1' is not valid for 'Value'.	The bar value cannot be properly determined as centre location becomes invalid

<p>Form is resized to smallest, then maximized</p>	<p>9 The bars disappear and the view is normal</p>		<p>An error is thrown as the image becomes too small to display.</p> <p>System.ArgumentException: Parameter is not valid.</p>
--	--	--	---

### Fixing Error #2 & #9

In order to fix the error with the form getting too small, a minimum size can be enforced via a property:



### Fixing Error #8

This error has a more problematic cause. It stems from the fact that when zooming in, changing the centre and then zooming out, the centre location can be placed in invalid places. In order to fix this, when displaying a new image the centre location must be checked to ensure it is still valid:

```
private void ClampCentreLocation(int width, int height) {
    int firstCrossInXAxis = GetFirstCrossPosition(width);
    int firstCrossInYAxis = GetFirstCrossPosition(height);

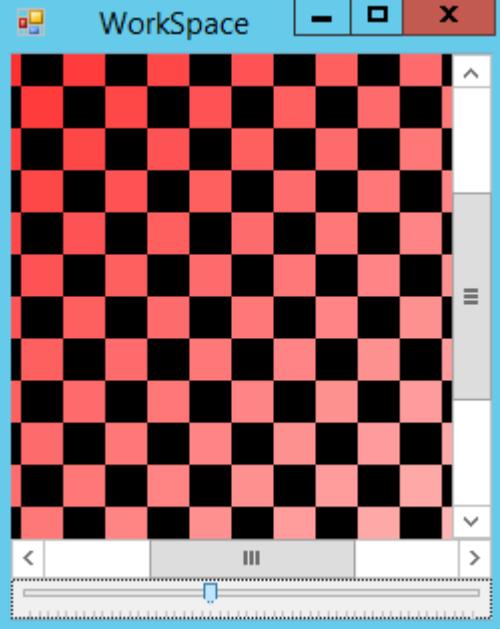
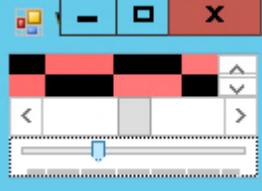
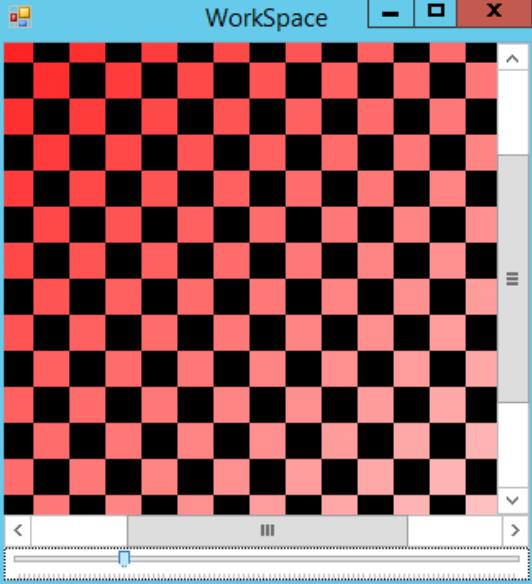
    // whether x is too small
    if (zoomSettings.fileCentreLocation.fileX < firstCrossInXAxis) {
        zoomSettings.fileCentreLocation.fileX = firstCrossInXAxis;
    }

    // whether y is too small
    if (zoomSettings.fileCentreLocation.fileY < firstCrossInYAxis) {
        zoomSettings.fileCentreLocation.fileY = firstCrossInYAxis;
    }

    // whether x is too big
    if (zoomSettings.fileCentreLocation.fileX > (fileWidth - firstCrossInXAxis)) {
        zoomSettings.fileCentreLocation.fileX = (fileWidth - firstCrossInXAxis);
    }

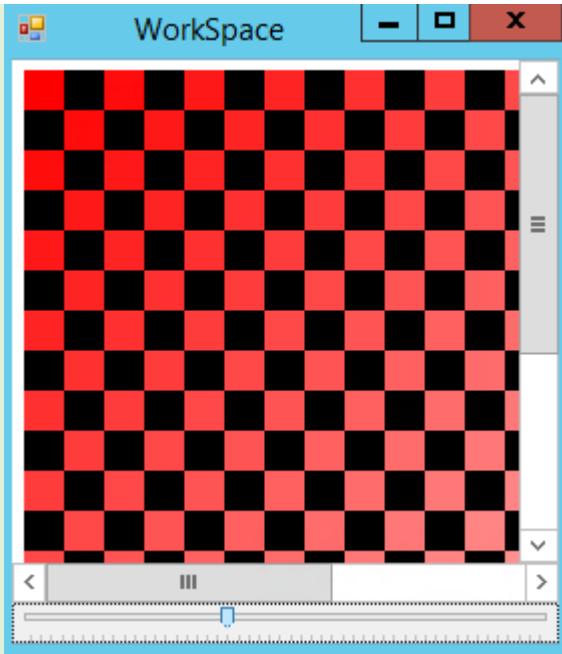
    // whether y is too big
    if (zoomSettings.fileCentreLocation.fileY > (fileHeight - firstCrossInYAxis)) {
        zoomSettings.fileCentreLocation.fileY = (fileHeight - firstCrossInYAxis);
    }
}
```

## Unit Test 2.11 #2

Test ID	Expected Result	Actual Result	Comment
<i>Form is resized to half the size of image</i>	1 Bar is half size of bounds and in its centre position		The bars are displaying the correct information
<i>Form is resized to its smallest position</i>	2 Bar is small but still central.		The minimum form size prevents form from getting too small
<i>Form is resized back to half size of image</i>	3 Bar is half size of bounds and in its centre position		

*Horizontal Scroll bar is moved to far left*

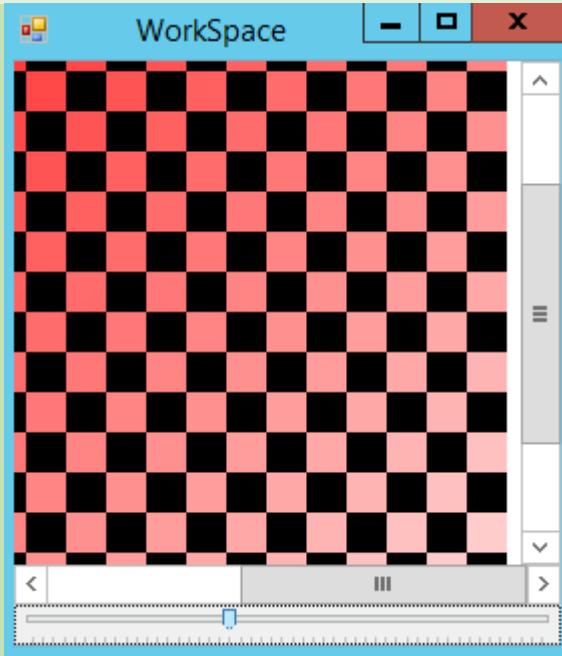
- 4 The far left of the image is displayed, but no more



The far left of the image is displayed

*Horizontal Scroll bar is moved to far right*

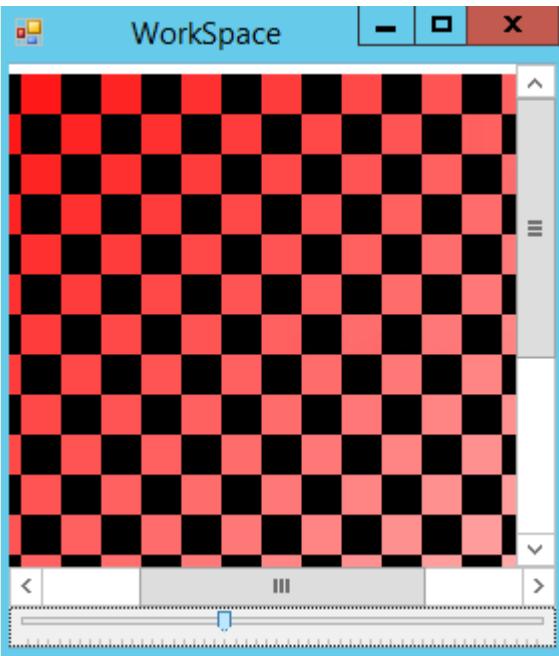
- 5 The far right of the image is displayed, but no more



The far right of the image is displayed

*Vertical Scroll  
bar is moved  
to highest*

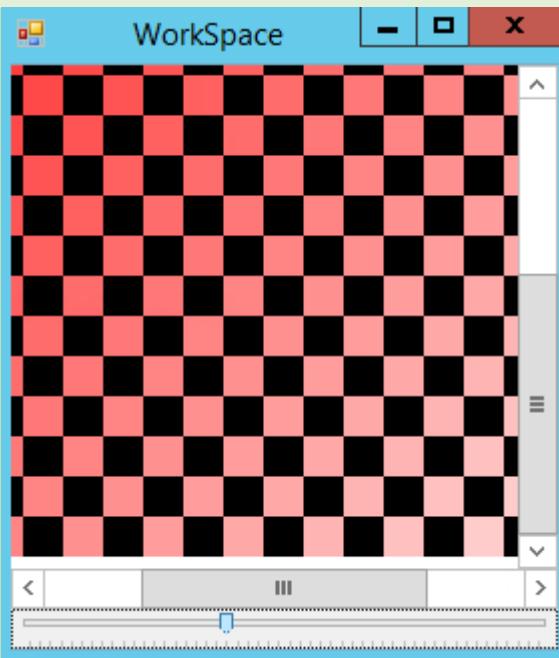
- 6 The highest  
of the image  
is displayed,  
but no more



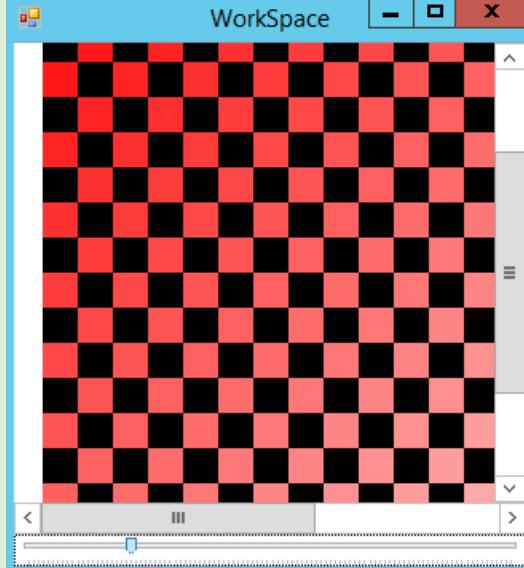
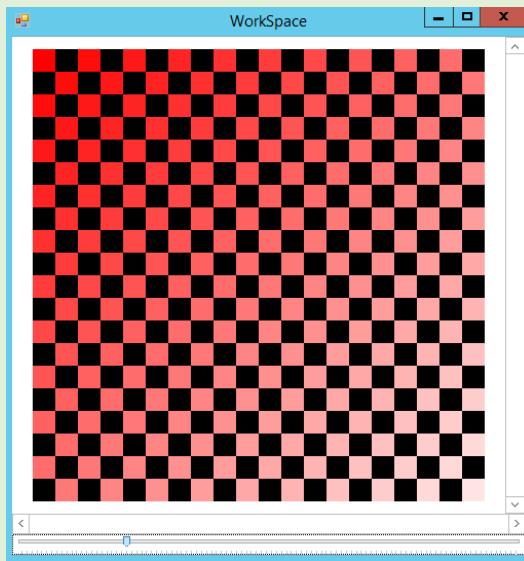
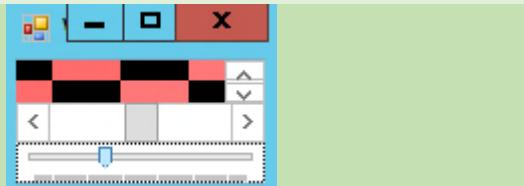
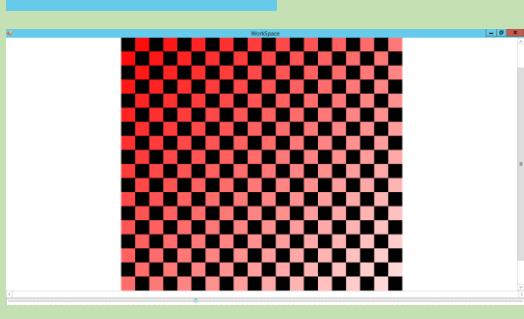
The top of  
the image is  
displayed

*Vertical Scroll  
bar is moved  
to lowest*

- 7 The lowest  
of the image  
is displayed,  
but no more



The bottom  
of the image  
is displayed

<p><i>Horizontal scroll bar is moved far left, then form size increased</i></p>	<p>8   Centre locations is moved to the left when needed</p>	 	<p>The centre location of the image now correctly updates to the centre of the image, resolving the invalid bar value</p>
<p><i>Form is resized to smallest, then maximized</i></p>	<p>9   The bars disappear and the view is normal</p>	 	<p>The system copes with this sudden change of size.</p>

## Algorithm 2.13 Detecting Mouse Clicks

This algorithm has been implemented, in accordance to [Algorithm 2.13](#) demonstration code:

```
void PictureBox1Click(object sender, EventArgs e)
{
    MouseEventArgs me = (MouseEventArgs)e;
    MessageBox.Show(String.Format("X: {0} Y: {1}", me.X, me.Y));
}

void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    if (e.Button = MouseButtons.Left) {
        // set a pixel
    }
}

void DisplayBoxMouseMove(object sender, MouseEventArgs e)
{
    if (e.Button = MouseButtons.Left) {
        // set a pixel
    }
}
```

However this implementation uses two events rather than one, for when the mouse is held down and moved.

## Algorithm 2.14 & Algorithm 2.15

The algorithm for setting pixels on click can be very implemented, by editing the parameters of SetPixel to accept a DisplayPoint or FilePoint:

```
public void SetPixel(FilePoint filePoint, Color colour) {
    try {
        pixels[filePoint.fileX, filePoint.fileY] = colour;
    } catch (IndexOutOfRangeException) {
        // ignore request if it tried to set out of bounds
        // TODO: some sort of logging system to warn about this
    }
}

public void SetPixel(DisplayPoint displayPoint, Color colour) {
    FilePoint filePoint = DisplayPointToFilePoint(displayPoint);
    SetPixel(filePoint, colour);
}
```

And then adding a call from WorkSpace:

```
void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.Black);
        UpdateDisplayBox(true);
    }
}

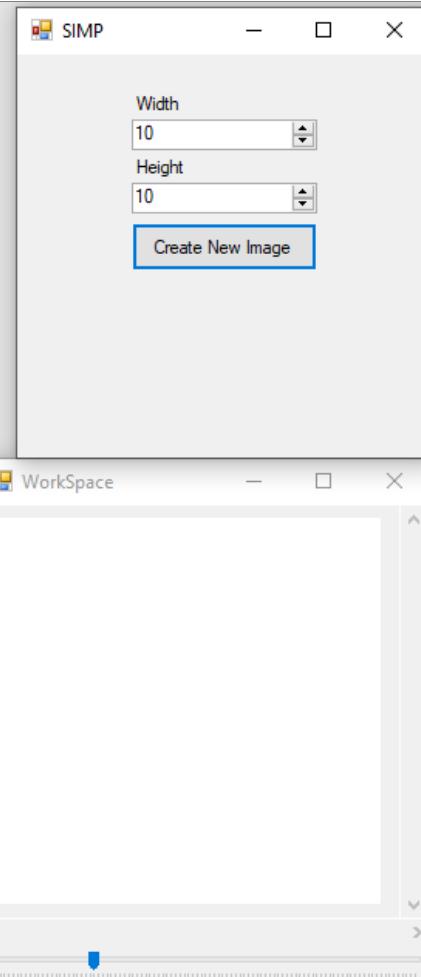
void DisplayBoxMouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.Black);
        UpdateDisplayBox(true);
    }
}
```

This means that the **image can now be changed at runtime**.

# 2.3 Testing

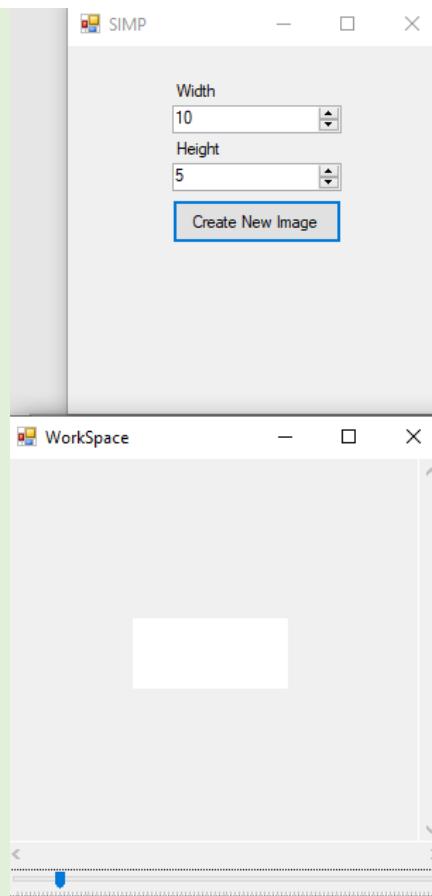
## 2.3.1 Full Testing

### Initial Testing Table

<i>Test</i>	<i>ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<code>new Image(10,10)</code>	1	An image of size 10fpx, 10fpx		A 10px by 10px image is successfully created.

```
new Image(10,5)
```

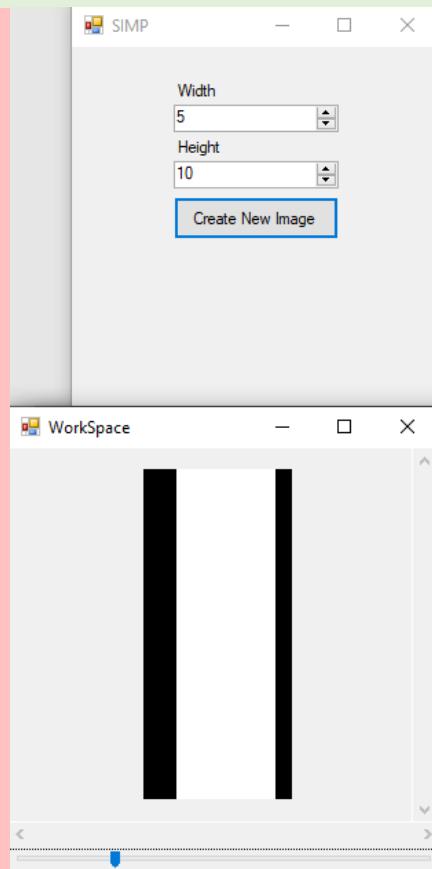
2 A 10fpx,  
5fpx image



The slightly  
thinner image  
is successfully  
created

```
new Image(5,10)
```

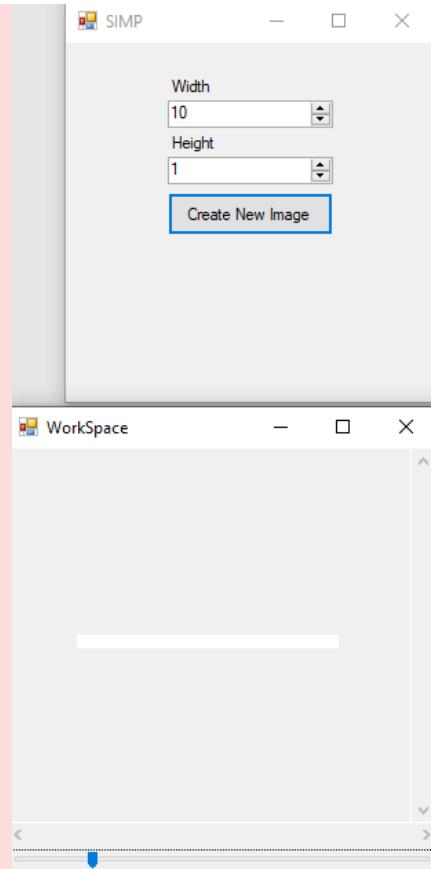
3 A 5fpx,  
10fpx  
image



A tall image is  
successfully  
created.  
However  
there are  
problems  
drawing on  
the odd edge  
of this sort of  
image, as only  
half of the  
edge is  
displayed.

`new Image(10,1)`

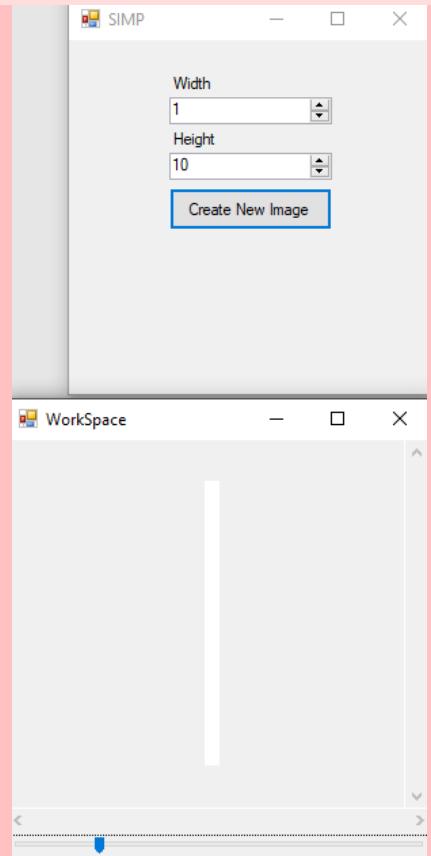
4 A 10fpx,  
1fpx image



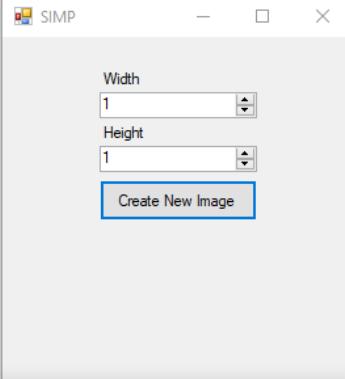
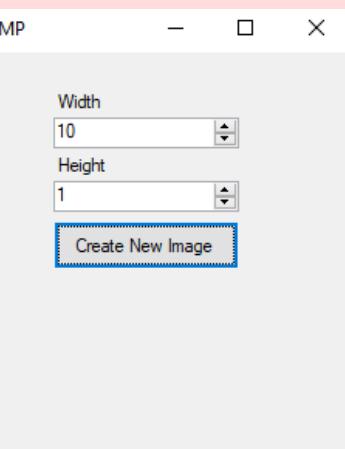
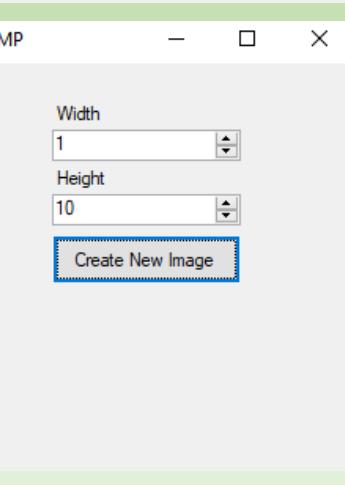
An image is created, but is much too thin (only displaying half of a pixel).

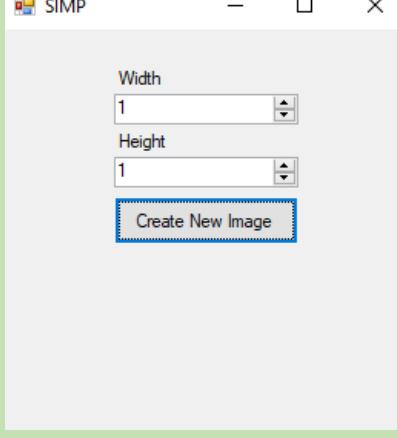
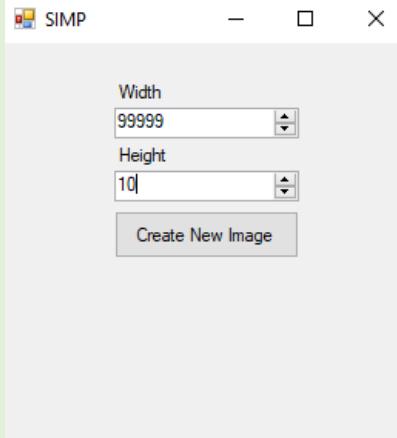
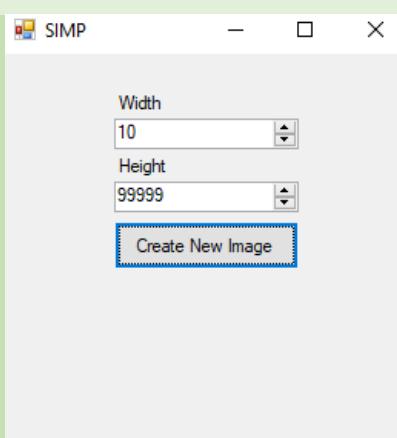
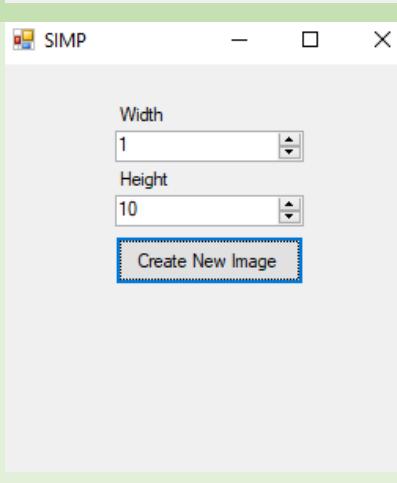
`new Image(1,10)`

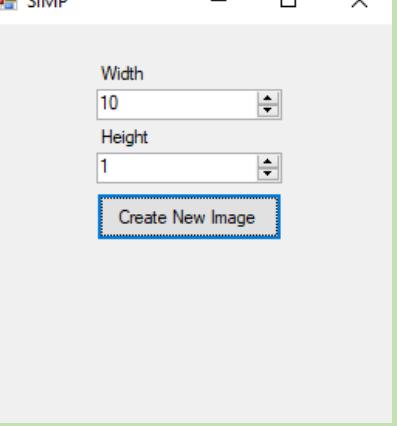
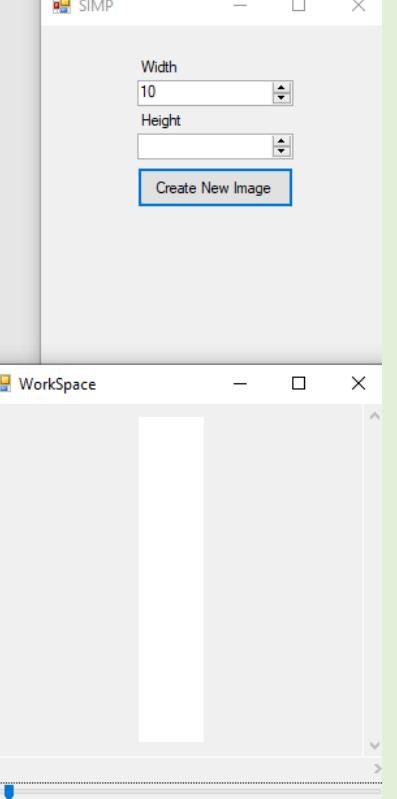
5 A 1fpx,  
10fpx image



An image is created, but is much too thin as only a half pixel is displayed

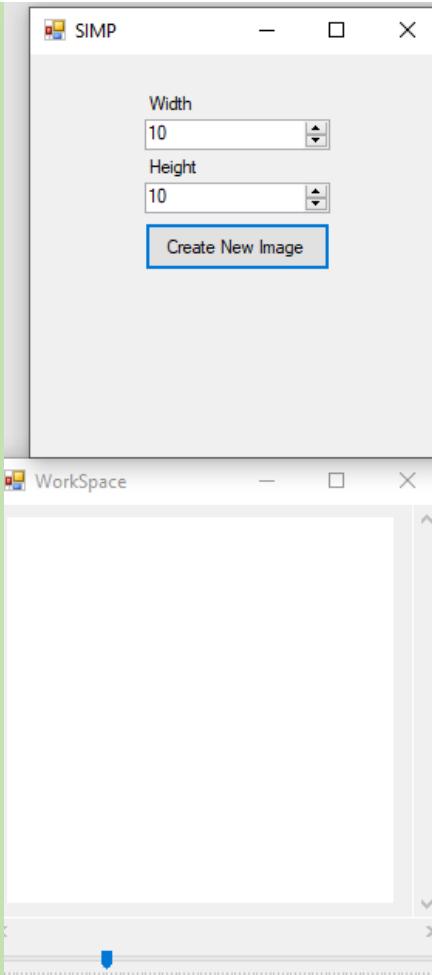
<code>new Image(1,1)</code>	6 A 1px, 1px image	 	A very small picture is created, but it is not centrally aligned in the image.
<code>new Image(10,0)</code>	7 The parameters are rejected and no image is created		A height of 0 is not accepted, and autocorrects to 1
<code>new Image(0,10)</code>	8 The parameters are rejected and no image is created		A width of 0 is not accepted, and autocorrects to 1

<code>new Image(0,0)</code>	9 The parameters are rejected and no image is created		Both boxes are checked independent of each other so this poses no new issue
<code>new Image(100000,10)</code>	10 The parameters are rejected and no image is created		The width is autocorrected back down to the allowed maximum
<code>new Image(10,100000)</code>	11 The parameters are rejected and no image is created		The height is autocorrected back down to the allowed maximum
<code>new Image(-1,10)</code>	12 The parameters are rejected and no image is created		The invalid width is corrected

<code>new Image(10,-1)</code>	13 The parameters are rejected and no image is created		The invalid height is corrected
<code>new Image(10)</code>	14 The parameters are rejected and no image is created		In cases where a parameter is left blank the last valid value is used instead (in this case 50)

```
new Image("10","10")
```

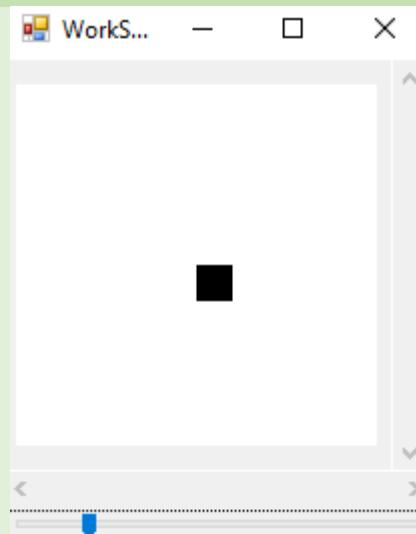
- 15 The parameters are rejected and no image is created



It is impossible to create an image from text

```
SetPixel(5,5,Black)
```

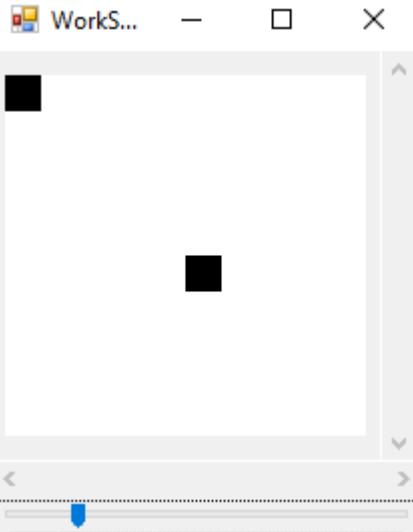
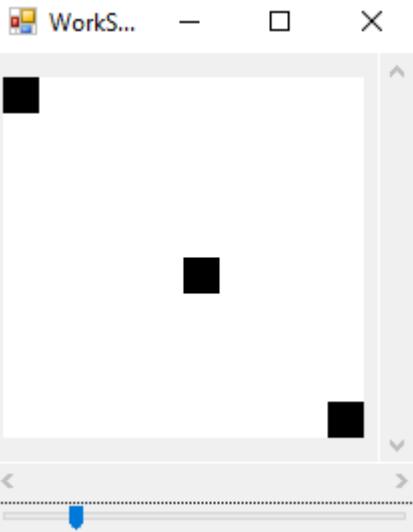
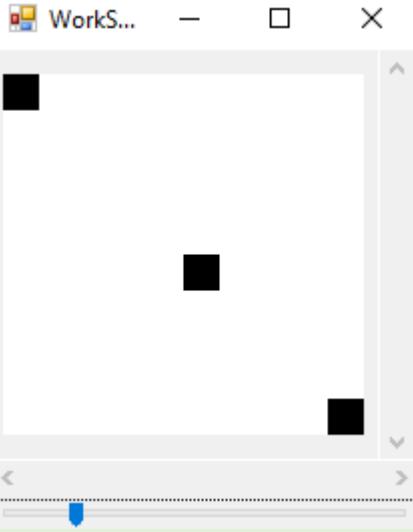
- 16 A pixel near the middle of the image is set to black

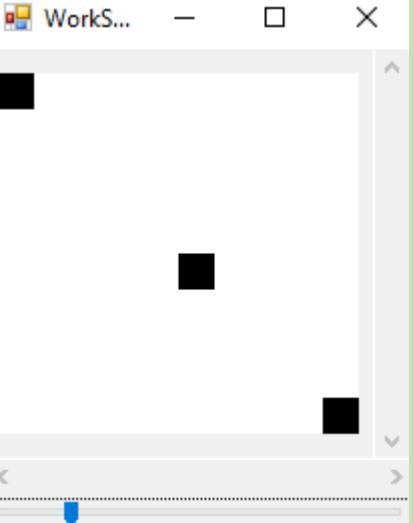
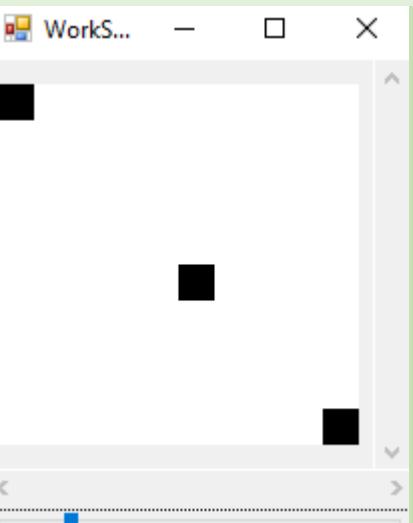


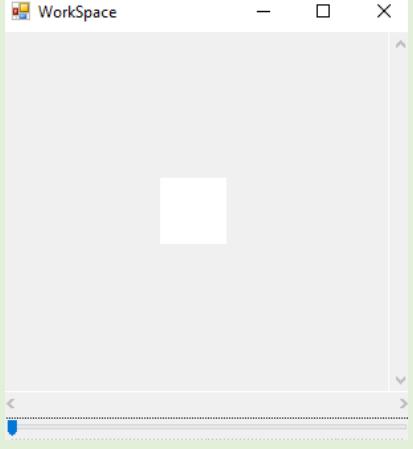
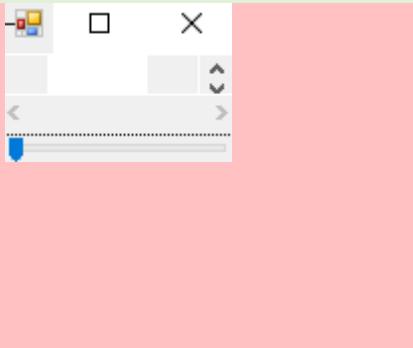
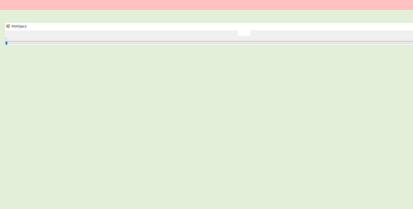
```
SetPixel(5,5,Green)
```

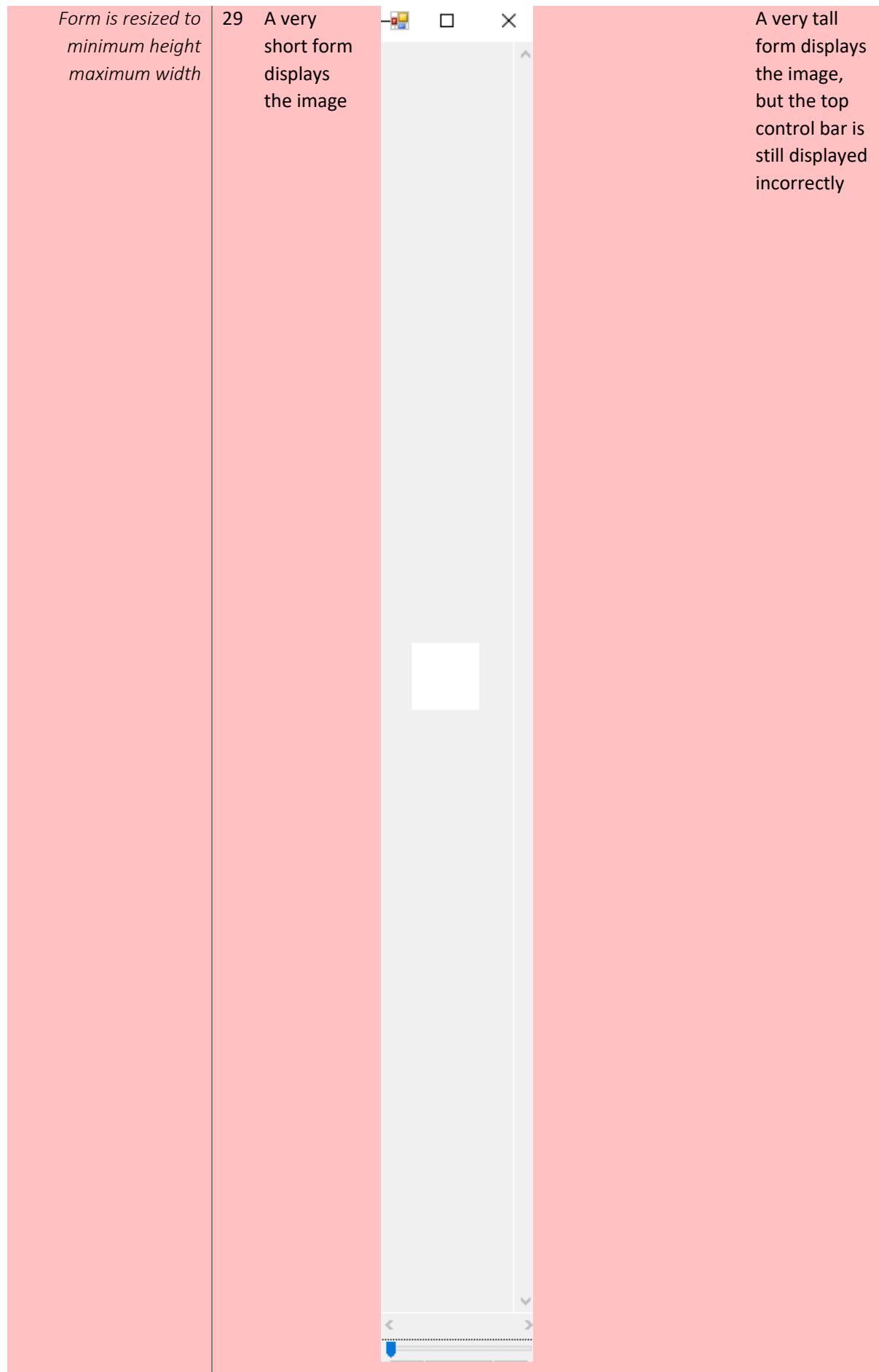
- 17 A pixel near the middle of the image is set to yellow

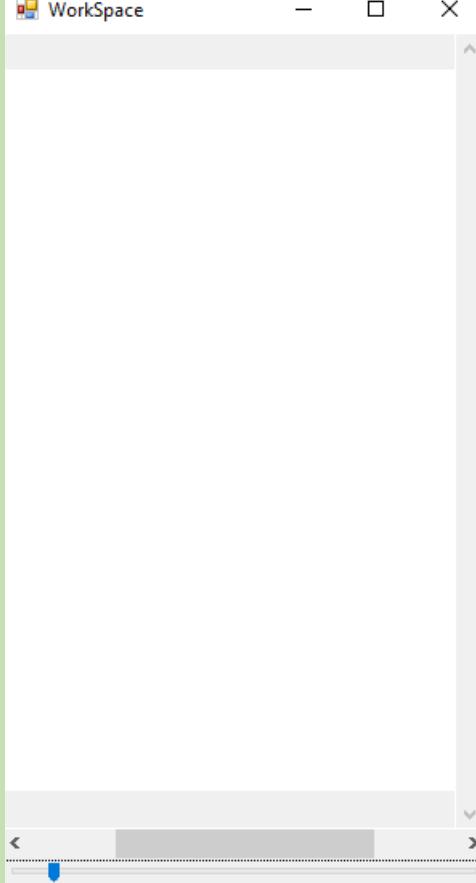
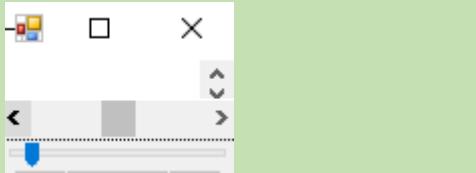
This is currently impossible, however previous tests indicate that colour displaying is possible.

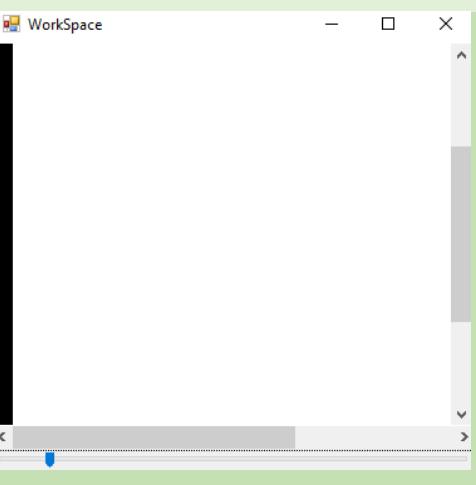
<code>SetPixel(0,0,Black)</code>	18 A pixel in the top-left corner is set to black		A pixel at the top left is set
<code>SetPixel(9,9,Black)</code>	19 A pixel in the bottom-right corner is set to black		
<code>SetPixel(-1,0,Black)</code>	20 No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored

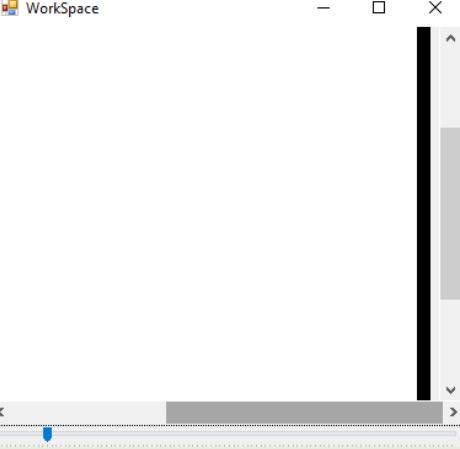
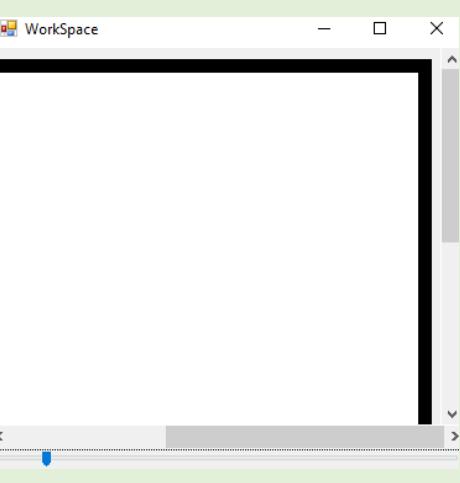
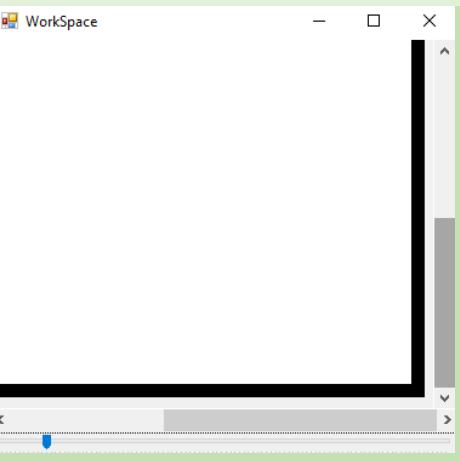
<code>SetPixel(0,-1,Black)</code>	21	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored
<code>SetPixel(10,0,Black)</code>	22	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored
<code>SetPixel(0,10,Black)</code>	23	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored

<i>Form is started at normal size</i>	<b>24</b> Image Displays in the middle of the form		
<i>Form is maximized</i>	<b>25</b> Image displays in the middle of the large form		The image (while small) is correctly displayed in the middle of the large form
<i>Form is minimized</i>	<b>26</b> No image is displayed (as form is currently invisible)	No image	There is no problem when the program is minimized
<i>Form is resized to smallest possible</i>	<b>27</b> The form cannot be made smaller than the image		A small amount of the image is shown, however the minimize button is displayed over the icon
<i>Form is resized to minimum width maximum height</i>	<b>28</b> A very thin form displays the image		A very wide form correctly displays the image in its centre



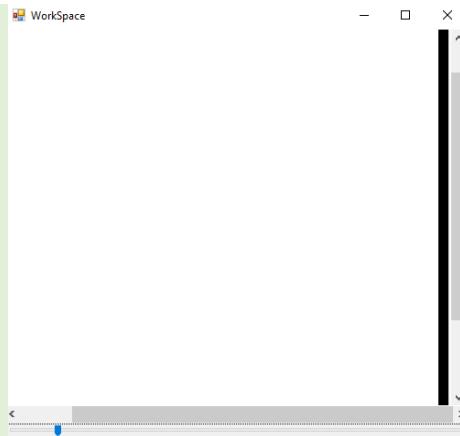
<p><i>The form's size is rapidly changed.</i></p>	<p>30 The image is very quickly moved around but remains centred</p>	<p>No image, but program remains stable</p>	<p>The program can be very quickly manipulated but still appear smooth</p>
<p><i>Form is resized to half the size of image</i></p>	<p>31 Bar is half size of bounds and in its centre position</p>		<p>The bar is in its correct position</p>
<p><i>Form is resized to its smallest position</i></p>	<p>32 Bar is small but still central.</p>		<p>The bar still remains close to the centre of the image</p>

<p><i>Form is resized back to half size of image</i></p>	<p>33 Bar is half size of bounds and in its centre position</p>	
<p><i>Horizontal Scroll bar is moved to far left</i></p>	<p>34 The far left of the image is displayed, but no more</p>	 <p>The far left is displayed correctly</p>

<p><i>Horizontal Scroll bar is moved to far right</i></p>	<p>35 The far right of the image is displayed, but no more</p>		<p>The far right is displayed correctly</p>
<p><i>Vertical Scroll bar is moved to highest</i></p>	<p>36 The highest of the image is displayed, but no more</p>		<p>The top right is now correctly displayed</p>
<p><i>Vertical Scroll bar is moved to lowest</i></p>	<p>37 The lowest of the image is displayed, but no more</p>		<p>The bottom right is now correctly displayed</p>

*Horizontal scroll bar is moved far right, then form size increased*

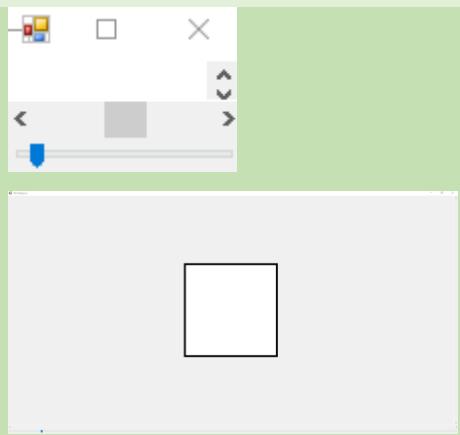
38 Centre locations is moved to the left when needed



The centre location is now correctly moved when the size increases

*Form is resized to smallest, then maximized*

39 The bars disappear and the view is normal



The program can handle such a sudden change in size

## Error Handling

Fixing Error #3, #4, #5 and #6

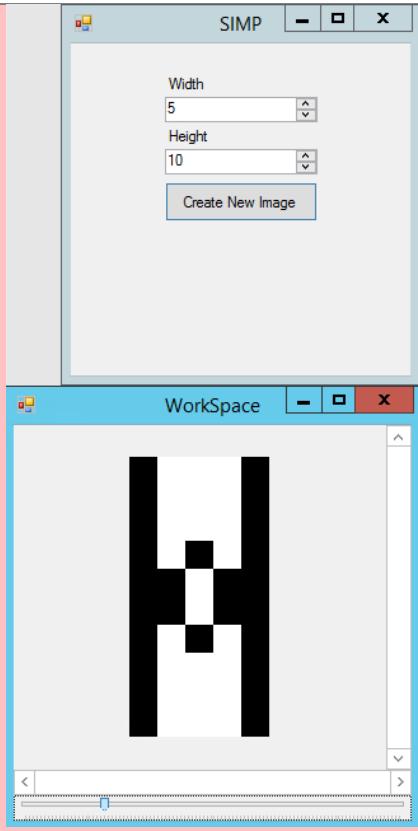
The error arises from the fact that on an odd-dimensioned image, when the display zoom centre is calculated, the width and height are simply halved:

```
zoomSettings.displayCentreLocation = new DisplayPoint(width/2,height/2);
```

However this would lead to (on odd numbered images) a centre location that didn't align with any file pixels. In order to solve this a division and multiplication system has been implemented to round the centre location to the nearest file pixel:

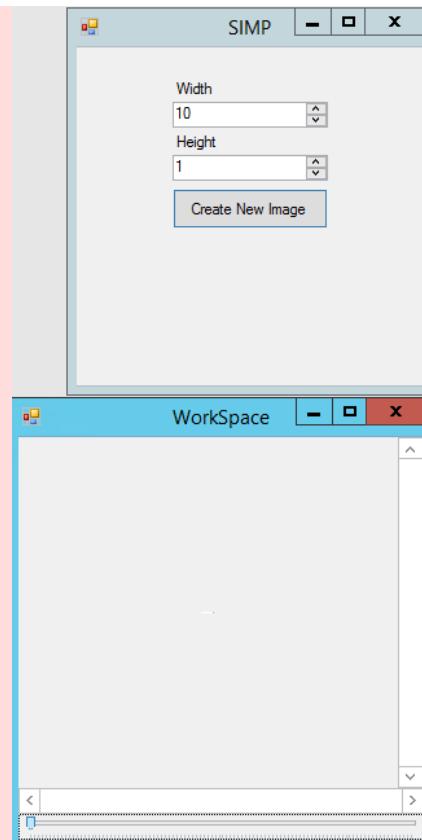
```
public void CalcDisplayCentreLocation(int width, int height) {  
    int newX = ((width/2)/zoom)*zoom;  
    int newY = ((height/2)/zoom)*zoom;  
    displayCentreLocation = new DisplayPoint(newX,newY);  
}
```

So, performing the tests again:

Test	ID	Expected Result	Actual Result	Comment
<code>new Image(5,10)</code>	3	A 5px, 10px image		A tall image is created, and now displays its needed pixels correctly.

`new Image(10,1)`

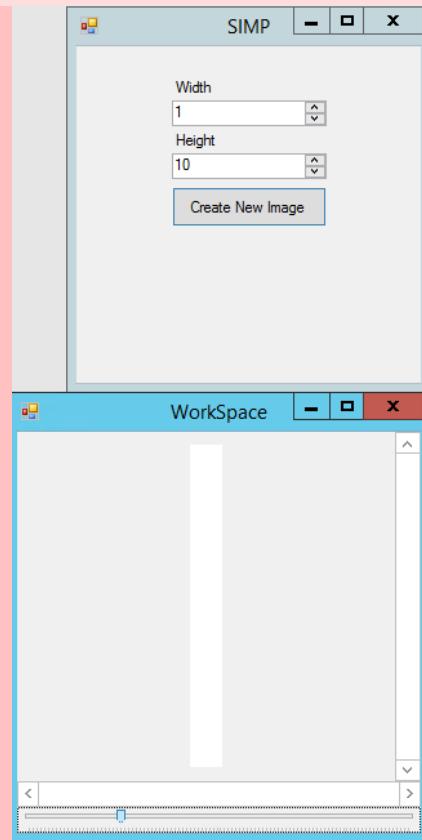
4 A 10px,  
1px image



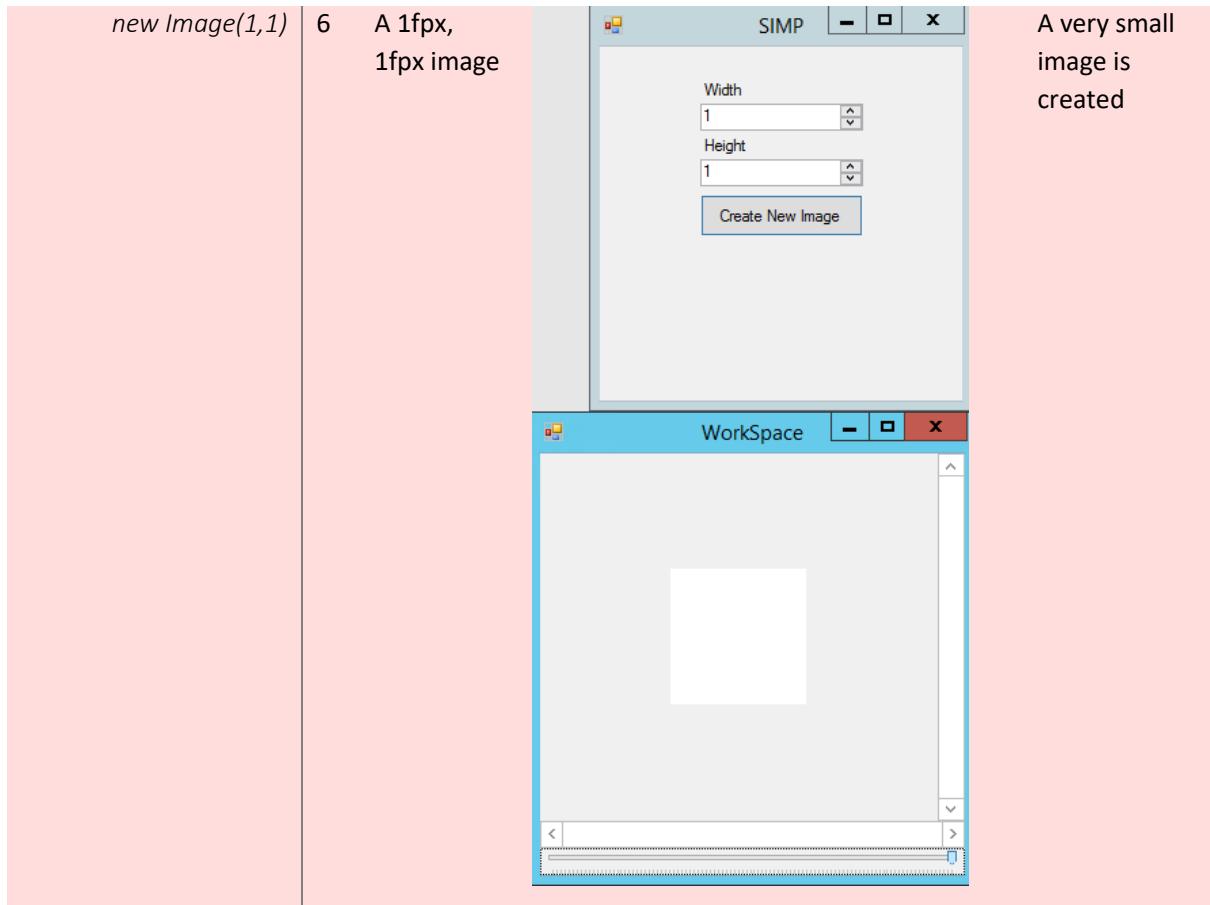
A very thin  
image is  
created

`new Image(1,10)`

5 A 1px,  
10px  
image



A very tall  
image is  
created



This now means that SIMP has passed **Alpha Testing**. It will now be handed to the Stakeholders for their initial thoughts and **Beta Testing**.

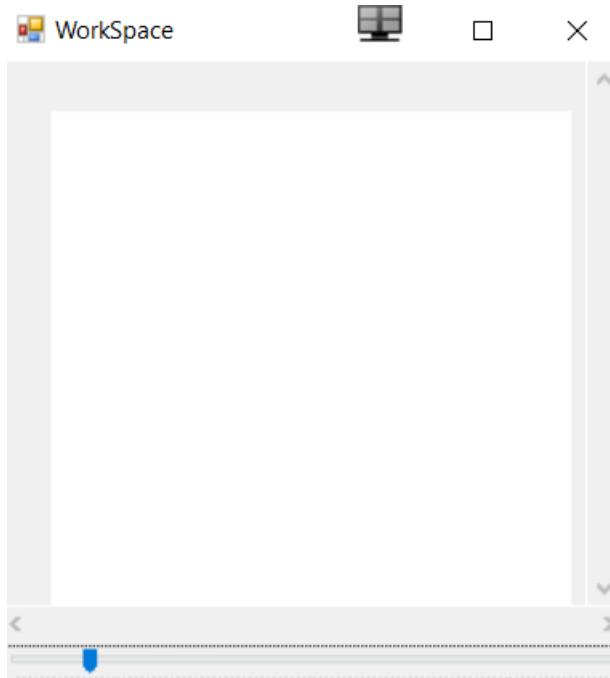
## 2.3.2 Client Testing & Feedback

---

### Alex G Client Feedback

Alignment at startup issue

After talking to a stakeholder, Alex G, he noted that when the program started the picture was not correctly aligned:



The error with the code happens because of the way the constructor is ordered:

```
private void Constructor(int width, int height) {
    image = new SIMP.Image(width,height,this);

    // Sets the dimensions of the workspace to the dimensions of the form
    CalculateDimensions();

    // Stores itself casted as a form
    attachedForm = (Form)this;

    // Updates the form
    UpdateDisplayBox(true);

    // Defines levels of padding
    leftPadding = SimpConstants.WORKSPACE_LEFT_PADDING;
    rightPadding = SimpConstants.WORKSPACE_RIGHT_PADDING;
    topPadding = SimpConstants.WORKSPACE_TOP_PADDING;
    bottomPadding = SimpConstants.WORKSPACE_BOTTOM_PADDING;
}
```

As the dimensions and updates are done *before* the padding is defined, so the program initially displays as if the padding is 0

A simple code re-order resolves this:

```
private void Constructor(int width, int height) {
    image = new SIMP.Image(width,height,this);

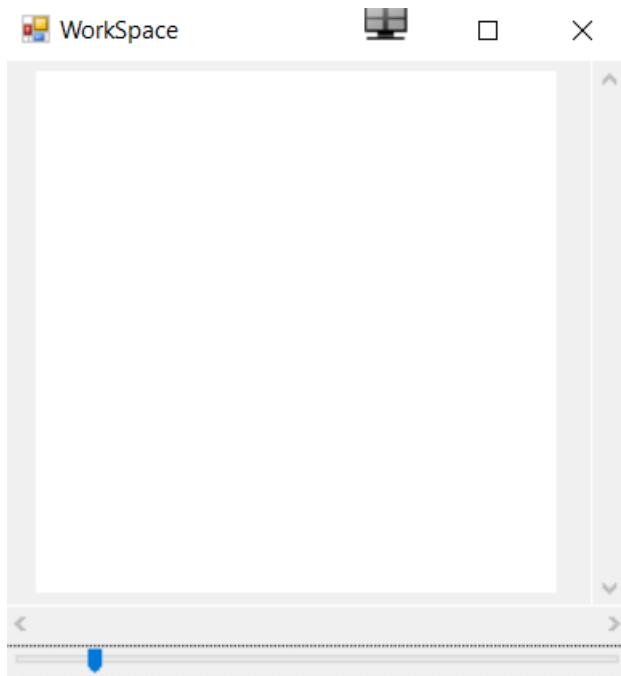
    // Stores itself casted as a form
    attachedForm = (Form)this;

    // Defines levels of padding
    leftPadding = SimpConstants.WORKSPACE_LEFT_PADDING;
    rightPadding = SimpConstants.WORKSPACE_RIGHT_PADDING;
    topPadding = SimpConstants.WORKSPACE_TOP_PADDING;
    bottomPadding = SimpConstants.WORKSPACE_BOTTOM_PADDING;

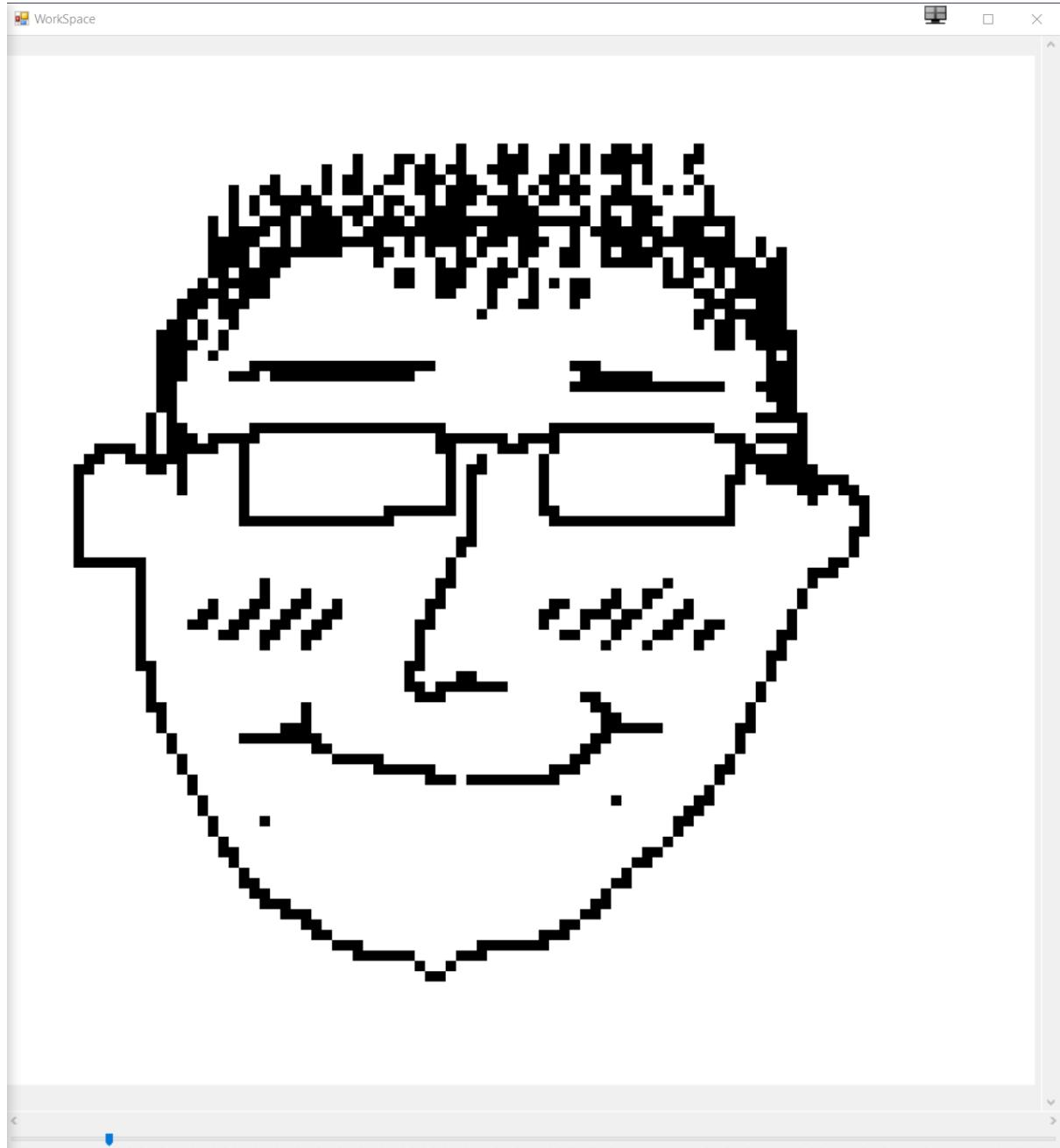
    // Sets the dimensions of the workspace to the dimensions of the form
    CalculateDimensions();

    // Updates the form
    UpdateDisplayBox(true);
}
```

So the program now starts correctly.



However, he was able draw an image



## Scrolling Upgrade

Alex G also noted that currently scrolling is rather cumbersome, as the vertical scroll bars, horizontal scroll bars cannot be scrolled through via the mouse pointer.

He suggested that scrolling normally should move the vertical bar upwards, pressing shift+scroll should move the horizontal bar, and pressing control+scroll should zoom in or out.

However, a first attempt to code this proved unsuccessful:

```
void WorkspaceScroll(object sender, ScrollEventArgs e)
{
    MessageBox.Show("scroll");
}
```

The MessageBox did not show. This was because the 'scroll' event is designed for controls that have an attached scroll bar. The WorkSpace did not have a scroll bar directly attached to it so did not trigger this event.

This can be solved by using the MouseWheel event:

```
this.MouseWheel += new MouseEventHandler(WorkspaceMouseWheel);
```

A small snipped of code can then be added to update stored variables on whether the control keys are pressed or not:

```
void WorkspaceKeyDown(object sender, KeyEventArgs e)
{
    if (e.Shift) {
        shiftPressed = true;
    }
    if (e.Control) {
        controlPressed = true;
    }
}

void WorkspaceKeyUp(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.ShiftKey) {
        shiftPressed = false;
    }
    if (e.KeyCode == Keys.ControlKey) {
        controlPressed = false;
    }
}
```

Finally, some code for updating the values based on what the booleans are set to can be implemented:

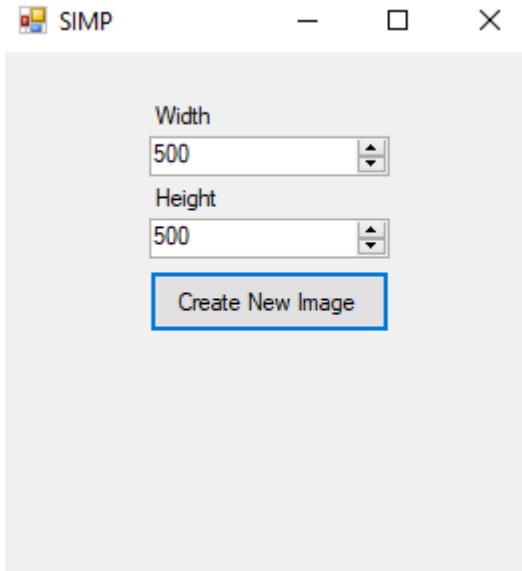
```
// zoom scrolling
if (controlPressed) {
    // if scrolled up
    if (e.Delta > 0) {
        if (barZoom.Value < barZoom.Maximum) {
            barZoom.Value++;
        }
    } else {
        if (barZoom.Value > barZoom.Minimum) {
            barZoom.Value--;
        }
    }
    BarZoomScroll(barZoom, new EventArgs());
}

// horizontal bar scrolling
else if (shiftPressed) {
    // if scrolled DOWN
    if (e.Delta < 0) {
        if (barHorizontal.Value < barHorizontal.Maximum) {
            barHorizontal.Value++;
        }
    } else {
        if (barHorizontal.Value > barHorizontal.Minimum) {
            barHorizontal.Value--;
        }
    }
}

// vertical bar scrolling
else {
    // if scrolled DOWN
    if (e.Delta < 0) {
        if (barVertical.Value < barVertical.Maximum) {
            barVertical.Value++;
        }
    } else {
        if (barVertical.Value > barVertical.Minimum) {
            barVertical.Value--;
        }
    }
}
```

## Permitting Custom Image Sizes

A few controls have been added to the starting MainForm. This allows the user to determine their image size, with a default of 50x50.



## Optimising Rectangle Drawing

My clients mentioned that the speed at which images are drawn is too slow, so this shall be the focus of today.

Currently the code for drawing a rectangle is quite inefficient. It creates a new solid brush for every new pixel every time it is drawn:

```
GFX.FillRectangle(new SolidBrush(pixels[x,y]),currentPoint.displayX,currentPoint.displayY,zoomSettings.zoom,zoomSettings.zoom);
```

This can be fixed by re-implementing the pixels array as an array of solid brushes rather than colours, as this is all that they are used for.

```
private SolidBrush[,] pixels;
```

This then means the drawing code can be implemented as:

```
GFX.FillRectangle(pixels[x,y],currentPoint.displayX,currentPoint.displayY,zoomSettings.zoom,zoomSettings.zoom);
```

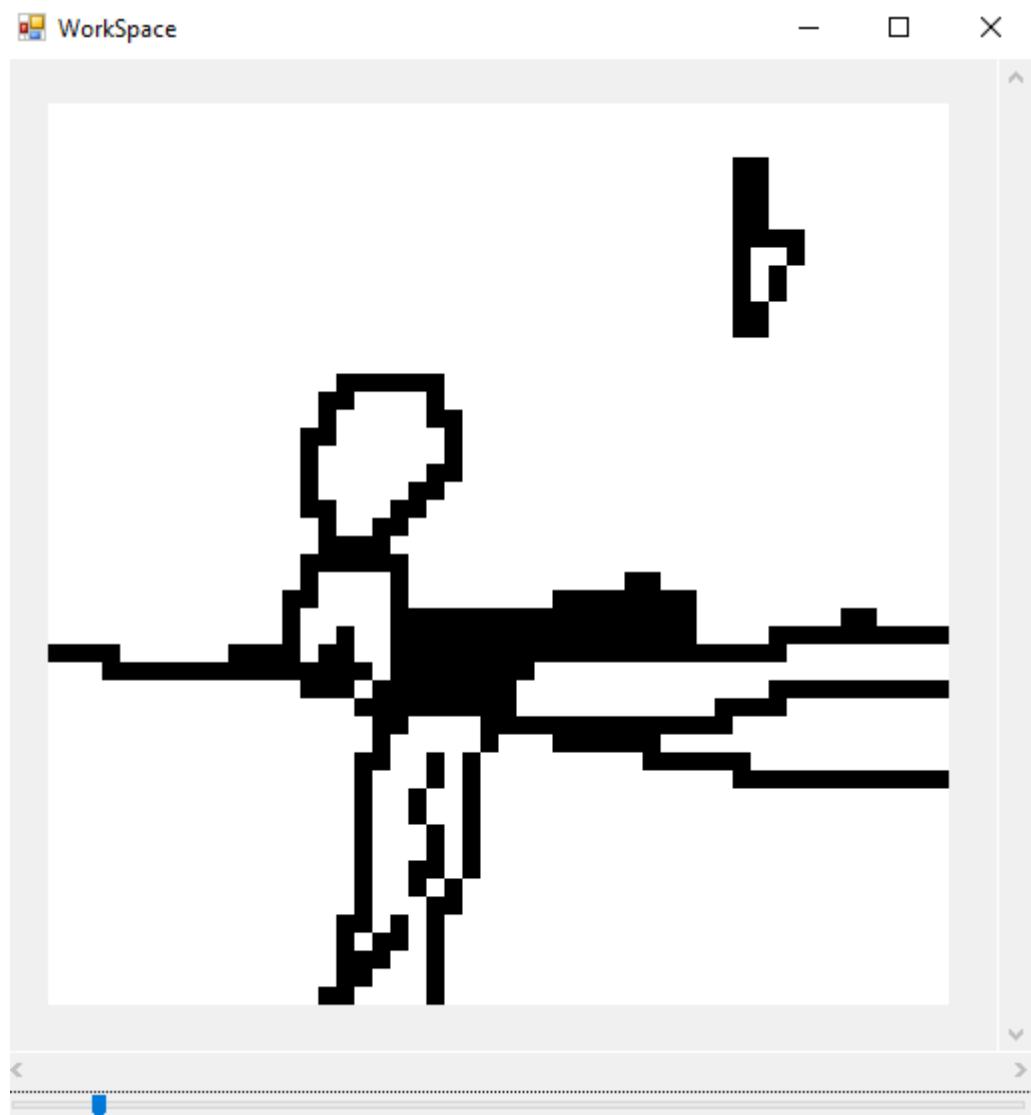
Which drastically reduces the code complexity, and increases performance.

## Alex H Client Feedback

### User Experience

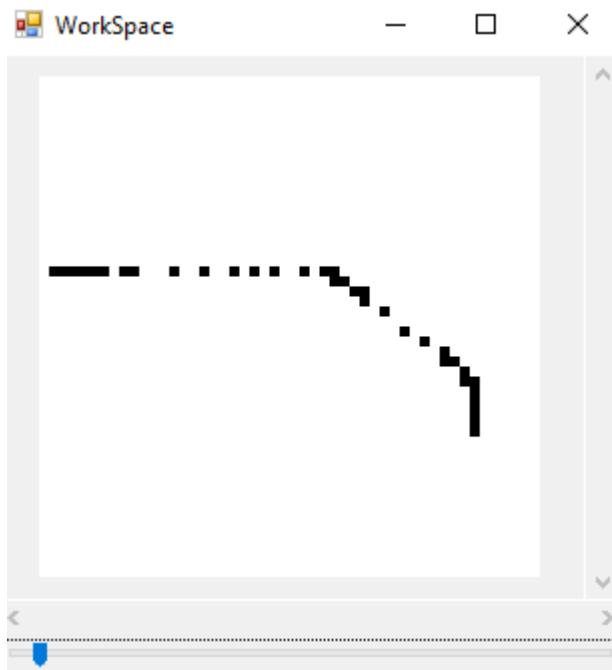
Alex H liked the start to the program so far, the zooming and scrolling capabilities, and the shortcuts. However he did experience a few issues.

Regardless he did draw a picture using the program



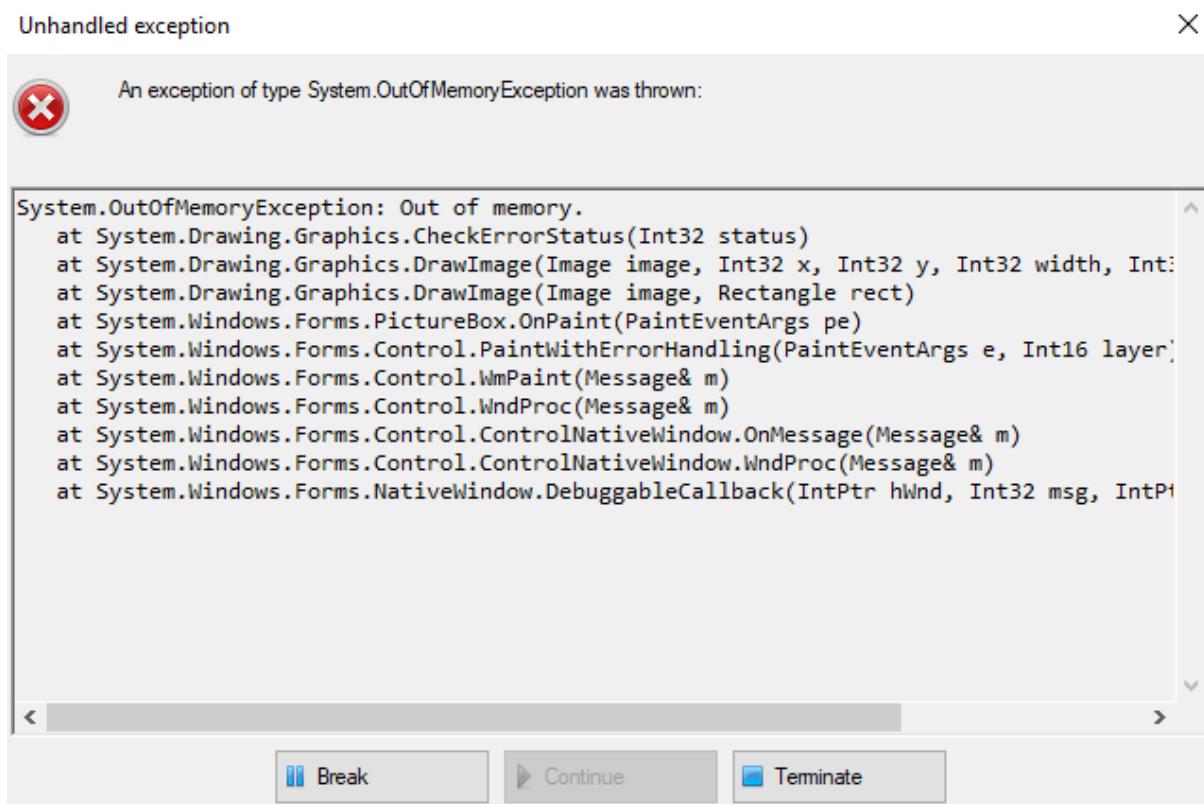
## Inconsistent Brush

The current brush is a very simple affair, however it means that when drawing larger lines there is the potential for gaps. This will be resolved later when the brush is fully implemented



## Memory Leak Crash

When Alex H was editing a large image, the program suddenly stopped. The error was due to an OutOfMemory exception:



The error occurred because each time a new image is created, the old image is not removed from memory. This means if too many images are created at the same time then they are made faster than the garbage collector can delete them, and the program crashes.

The cause of this was found to partially be that the `UpdateDisplayBox` function was being unnecessarily called, as a simple test showed:

```
int test = 0;

/// <summary>
/// Resizes, Relocates and (if redraw) updates image in the displayBox
/// </summary>
/// <param name="redraw"></param>
public void UpdateDisplayBox(bool redraw) {
    // Sets up the dimensions of displayBox
    ResizeDisplayBox();

    // Relocates the picture box
    RelocateDisplayBox();

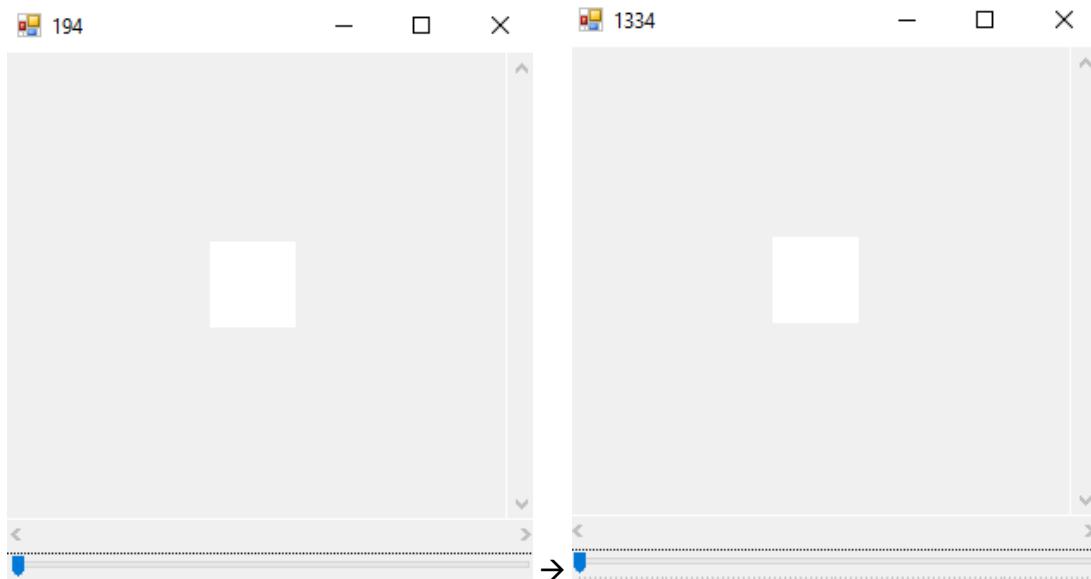
    if (redraw) {
        // Displays to the picture box
        displayBox.Image = image.GetDisplayImage(displayBox.Width, displayBox.Height);
        //displayBox.Image = image.GetDisplayImage(10,10);
    }

    // Updates the progress bars
    UpdateBar(EAxis.X, barHorizontal);
    UpdateBar(EAxis.Y, barVertical);

    test++;
}

Text = test.ToString();
}
```

This counts each time the function is called. However even then the program was idling the counter rapidly increased:



The error came from a conflict of functions. In the HasSizeChanged form, width was being compared to the size of the form:

```
private bool HasSizeChanged() {
    // if width has changed
    if (width != DisplayRectangle.Width) {
        return true;
    }
}
```

However was being set to the size of the form factored with padding:

```
private void CalculateDimensions() {
    width = DisplayRectangle.Width - (leftPadding + rightPadding);
```

This meant that HasSizeChanged always returned true, and the image was updated many extra times per second.

While this helped, when drawing on the image it is still necessary to redraw the same image many times. This is due to several unused images filling up memory, as they are not disposed of correctly.

Apps (6)

> Microsoft Word (32 bit)	0%	50.2 MB	0 MB/s
> Microsoft Word (32 bit) (2)	0%	107.0 MB	0 MB/s
> SIMP - SharpDevelop	0%	94.9 MB	0 MB/s
> SIMP (32 bit) (2)	7.0%	1,551.0 MB	0 MB/s
> Task Manager	0.2%	18.2 MB	0 MB/s
> Windows Explorer	0.2%	56.5 MB	0 MB/s

As shown in the above image SIMP is taking up much more memory than is needed. This means the C# Garbage Collector must be called explicitly to remove the extras.

To do this, a timer has been implemented, to call the garbage collect every once in a while:

```
updateCount++;
```

```
if (updateCount >= SimpConstants.WORKSPACE_REFRESH_PERIOD) {
    updateCount = 0;
    GC.Collect();
}
```

Where the refresh period is a constant. This reduces the memory massively but does increase the CPU usage if the period is too low.

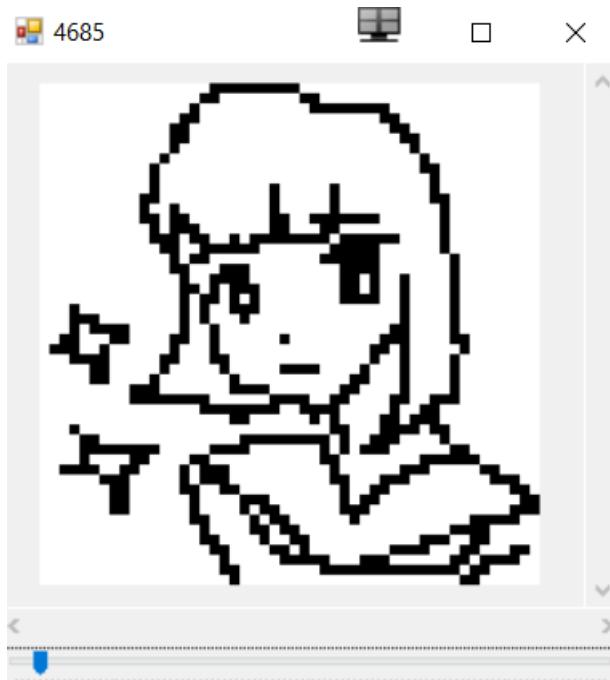
Apps (6)

> Microsoft Word (32 bit)	0%	50.2 MB	
> Microsoft Word (32 bit) (2)	0%	88.2 MB	
> SIMP - SharpDevelop	0.5%	96.1 MB	
> SIMP (32 bit) (2)	14.3%	25.6 MB	
> Task Manager	0.3%	18.5 MB	
> Windows Explorer	0.1%	56.7 MB	

Through testing a good constant value (5) was found.

## Dheshpreet Feedback

Today when Dheshpreet tested the program, she did generally enjoy the program, and was able to draw a picture.

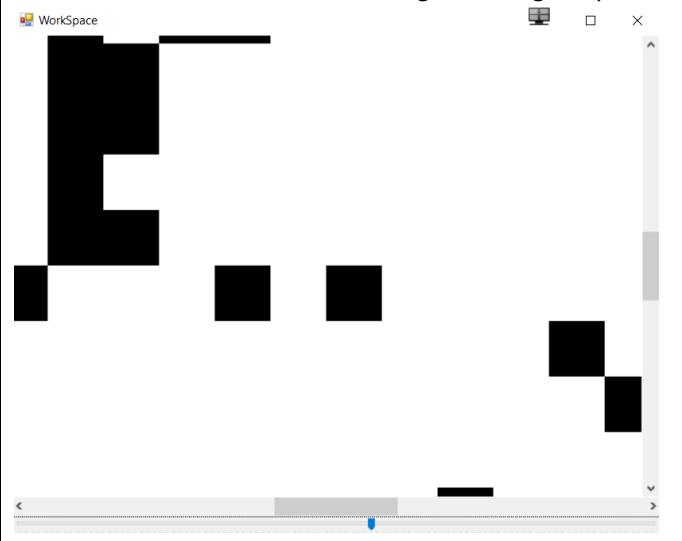
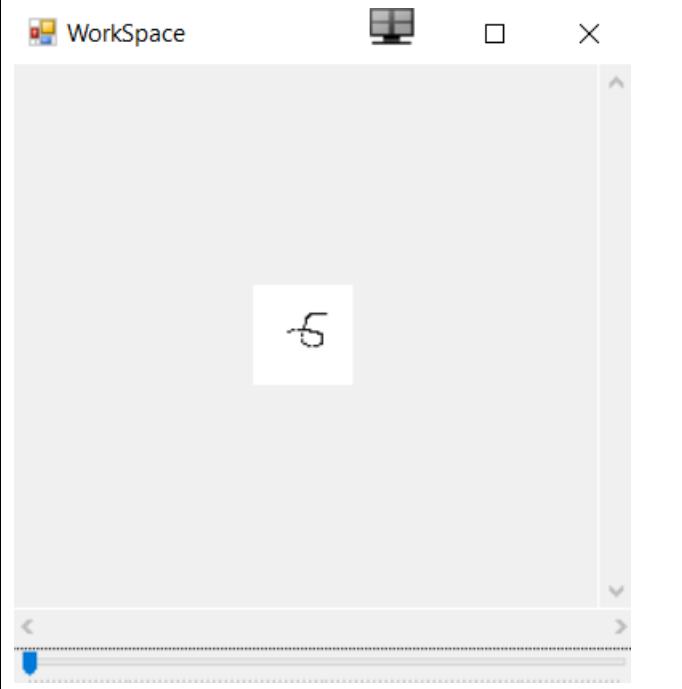


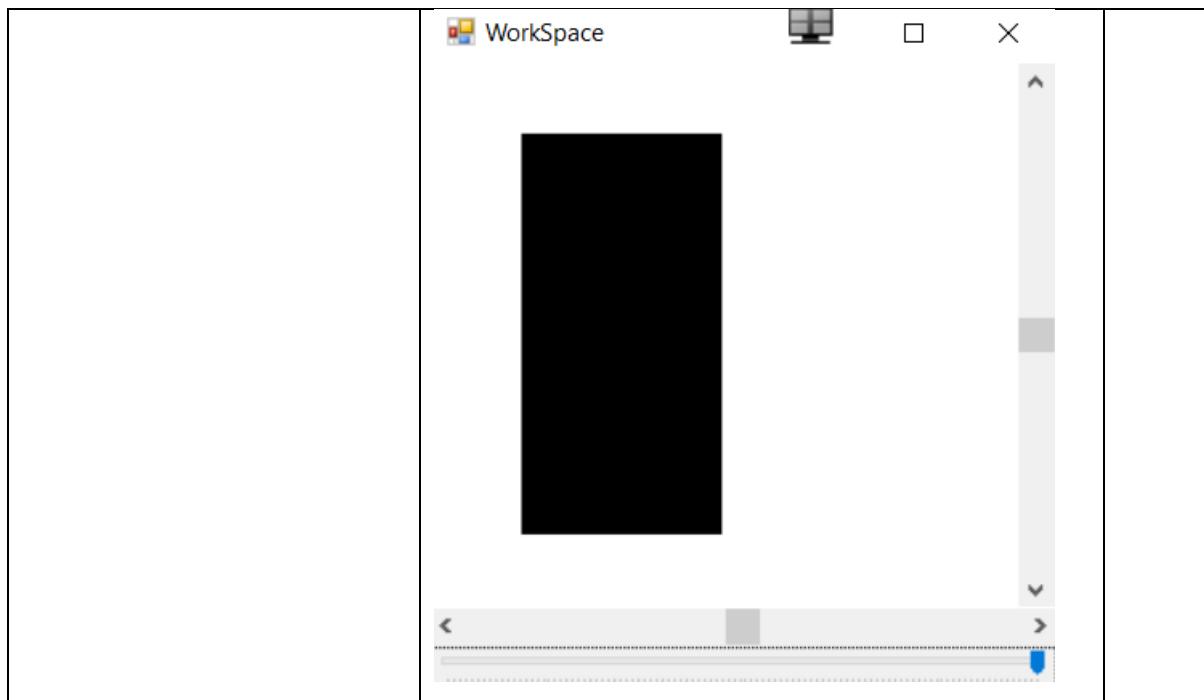
However she did find it difficult to draw without an undo feature. As this is a complicated feature it cannot be added now, so should be implemented in the next development phase.

### 2.3.3 Success Criteria Evaluation

Now that this section is completed, the following criteria can be now marked as fulfilled:

Feature	Proof	Code
Section B – Other editing tools		
Image viewer	<p>Screenshot of a currently being viewed image</p> A screenshot of a pixelated black and white image of a person's face, drawn in a simple drawing application. The image is displayed in a window titled 'WorkSpace' with standard window controls (minimize, maximize, close). The drawing is composed of black pixels on a white background.	B1

Bitmap image editor	<p>Screenshot of a zoom in on the image showing the pixels</p> 	B2
Zoom in (no zoom out)	<p>Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller</p> 	B9



So this makes the full diagram:

Not completed

To be done this section

Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3

RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11
<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
<b>Section C – File System</b>		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
<b>Section D – Usability</b>		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2

Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3
--------------------------------------	---	----

# Section 3 – Brushes & Tools

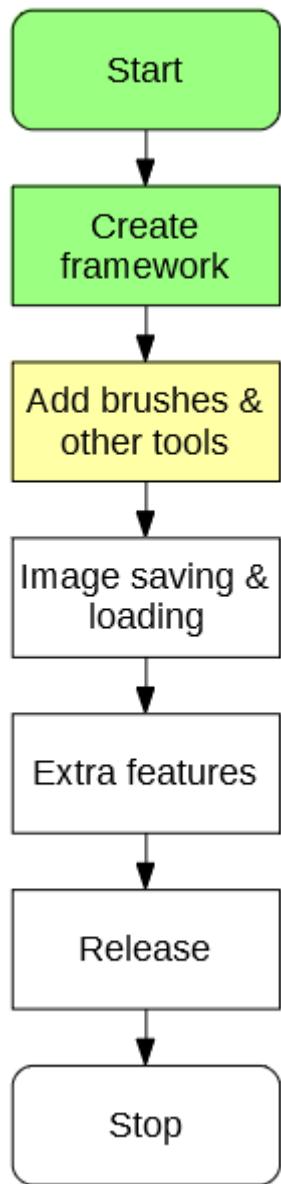
## General Contents

---

They will go here

## 3.1 Design

The brushes and tools section will contain many of the basic image editing tools, based upon the framework made in the previous section.



### 3.1.1 Success Criteria Fulfilment Plan

In this section the following success criteria are planned to be completed:

Not completed

To be done this section

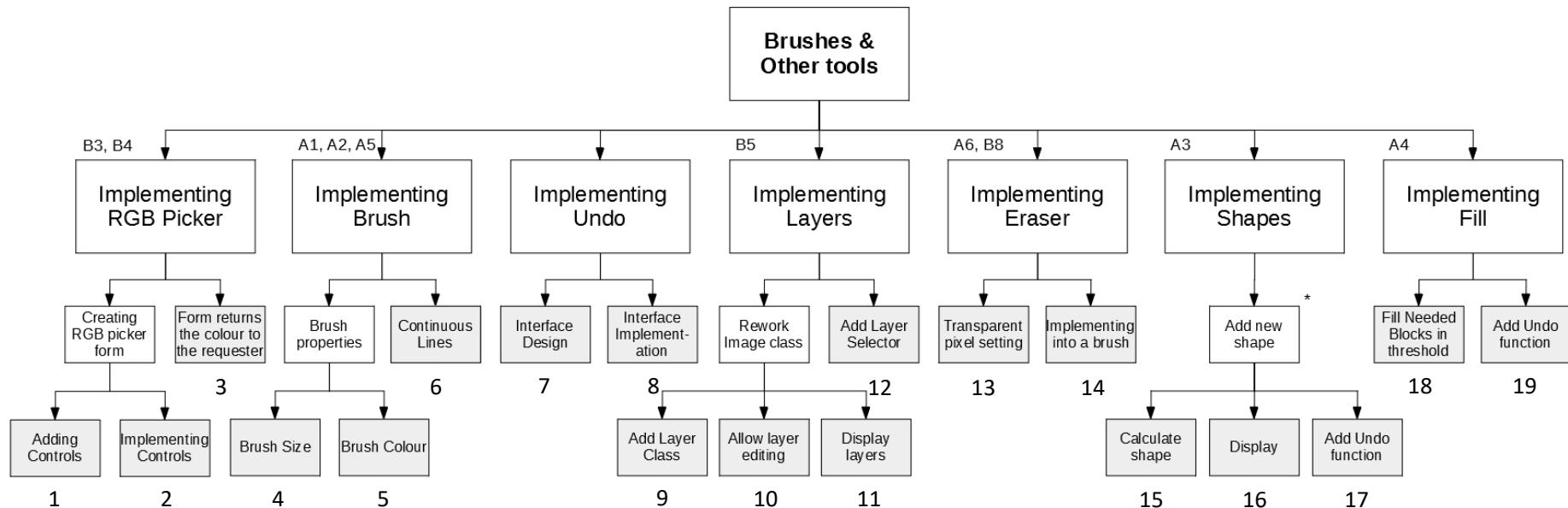
Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11

<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
<i>Creating a new image</i>	<i>Screenshot of a blank 300x300 square image</i>	C1
<i>Importing images</i>	<i>Screenshot of the file browser showing an image preview, and screenshot showing the image in the program</i>	C2
<i>Exporting images</i>	<i>Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder</i>	C3
<i>Supporting PNG and JPEG</i>	<i>Screenshot showing the file browser which accepts both PNG and JPEG images</i>	C4
<i>Saving and loading from a proprietary format</i>	<i>Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded</i>	C5
Section D – Usability		
<i>Program should be stable and not crash.</i>	<i>A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”</i>	D1
<i>Program should be easy to use</i>	<i>75% yes response to asking stakeholders “Did you find the program easy to use?”</i>	D2
<i>Features should be easily accessible</i>	<i>From the default state of the program, any feature will need to be activated by no less than 4 clicks</i>	D3

This should leave a reasonably functional (though not complete) editing program by the end of the section.

### 3.1.2 Section Decomposition



Each lowest level block has a label, which will become an algorithm designed in this document.

These algorithms will then be combined together.

### 3.1.3 Class Design

---

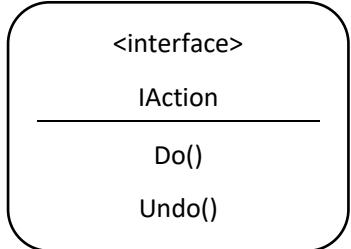
The necessary classes for this section are:

- IAction (for undo)
- PixelAction
- ITool
- IToolProperty
- NumericalProperty
- ColourProperty
- LineTool
- Layer

#### 3.1.3.1 IAction

The IAction interface will encapsulate a generic action in the program. This enforces that every action in the program must have the possibility to be both done and undone.

Class Diagram



Methods

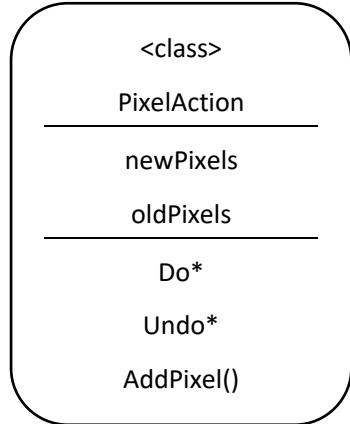
Method	Params	Return type	Justification
Do()	Workspace workspace	None	Will do the action implemented by the action on the workspace
Undo()	Workspace workspace	None	Will undo the action implemented by the action on the workspace

### 3.1.3.2 PixelAction

Implements IAction

Will encapsulate the concept of an action applied to the pixels of the image.

Class Diagram



Properties

Property	Datatype	Justification
newPixels	Dictionary of FilePoint to Pixels	Stores a record of all the changes made to pixels
oldPixels	Dictionary of FilePoint to Pixels	Stores a record of what the pixels used to look like

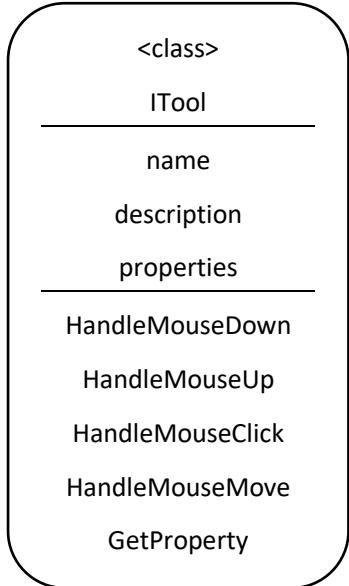
Methods

Method	Params	Return type	Justification
Do()	Workspace workspace	None	Inherited from IAction
Undo()	Workspace workspace	None	Inherited from IAction
AddPixel	FilePoint pixelLocation, Colour oldColour, Colour newColour	None	Adds a change of pixel to the record

### 3.1.3.3 ITool

The ITool interface will implement the generic idea of a tool. This will be used by the tool checking code to decide what tool is active.

#### Class Diagram



#### Properties

Property	Datatype	Justification
name	String	Stores the name of the tool to be displayed
description	String	Stores the description of the tool to be displayed
properties	List of ToolProperties	Stores the properties of that tool

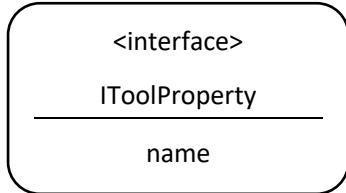
#### Methods

Method	Params	Return type	Justification
HandleMouseDown	FilePoint clickLocation, MouseButton button	None	Will run code when the mouse is pressed down
HandleMouseUp	FilePoint clickLocation, MouseButton button	None	Will run code when the mouse is released
HandleMouseClick	FilePoint clickLocation, MouseButton button	None	Will run code when the mouse is clicked
HandleMouseMove	FilePoint oldLocation, FilePoint newLocation	None	Will run code when the mouse is moved from one position to another
GetProperty	String propertyName	ToolProperty	Will return the specific property when asked for its name

### 3.1.3.4 IToolProperty

The IToolProperty encapsulates a generic property that a tool may have, and can be further defined by specific implementations of the class. This means that there will be a common format for all tools.

Class Diagram



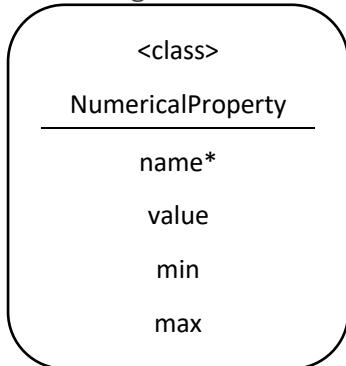
Properties

Property	Datatype	Justification
name	String	Stores the name of the property of the tool (e.g. size)

### 3.1.3.5 NumericalProperty

The numerical property encapsulates a property that is represented by a number (e.g. thickness).

Class Diagram



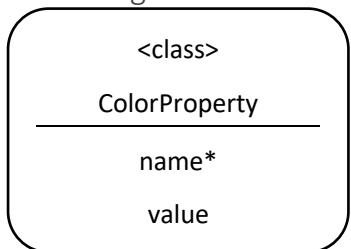
Properties

Property	Datatype	Justification
name*	String	Inherited from IToolProperty
value	Integer	Stores the value of the property
min	Integer	Stores the minimum value of the property
max	Integer	Stores the maximum value of the property

### 3.1.3.6 ColorProperty

The numerical property encapsulates a property that is represented by a color (e.g. brush colour).

Class Diagram

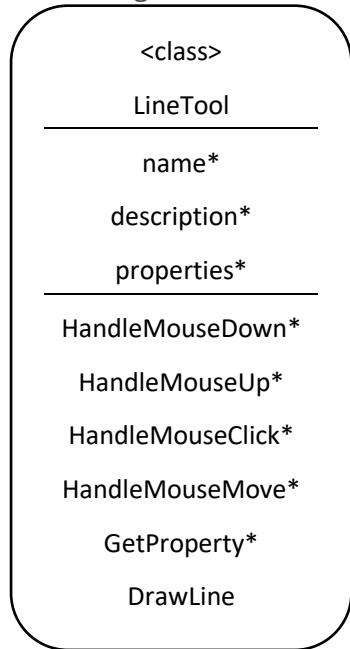


Properties

Property	Datatype	Justification
name*	String	Inherited from IToolProperty
value	Color	Stores the value of the property

### 3.1.3.7 LineTool

#### Class Diagram



#### Properties

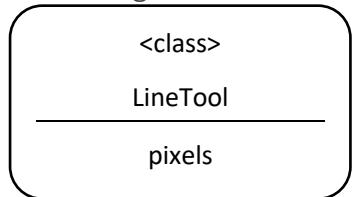
Property	Datatype	Justification
name	String	Inherited from ITool
description	String	Inherited from ITool
properties	List of ToolProperties	Inherited from ITool

#### Methods

Method	Params	Return type	Justification
HandleMouseDown	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseUp	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseClick	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseMove	FilePoint oldLocation, FilePoint newLocation	None	Inherited from ITool
GetProperty	String propertyName	ToolProperty	Inherited from ITool
DrawLine	None	None	Draws the line when required

### 3.1.3.8 Layer

#### Class Diagram

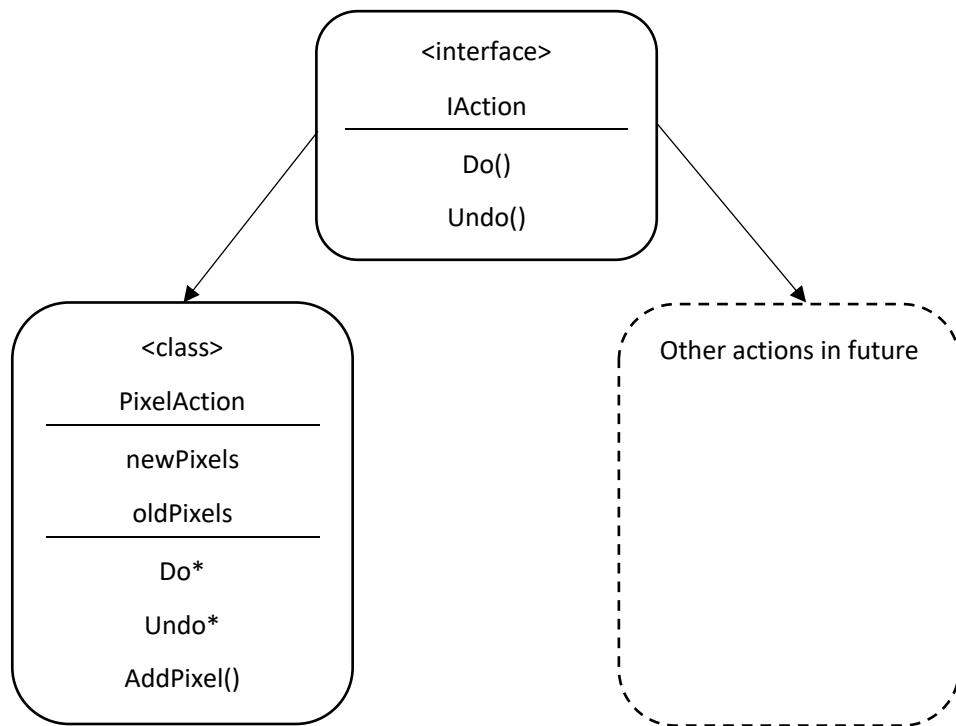


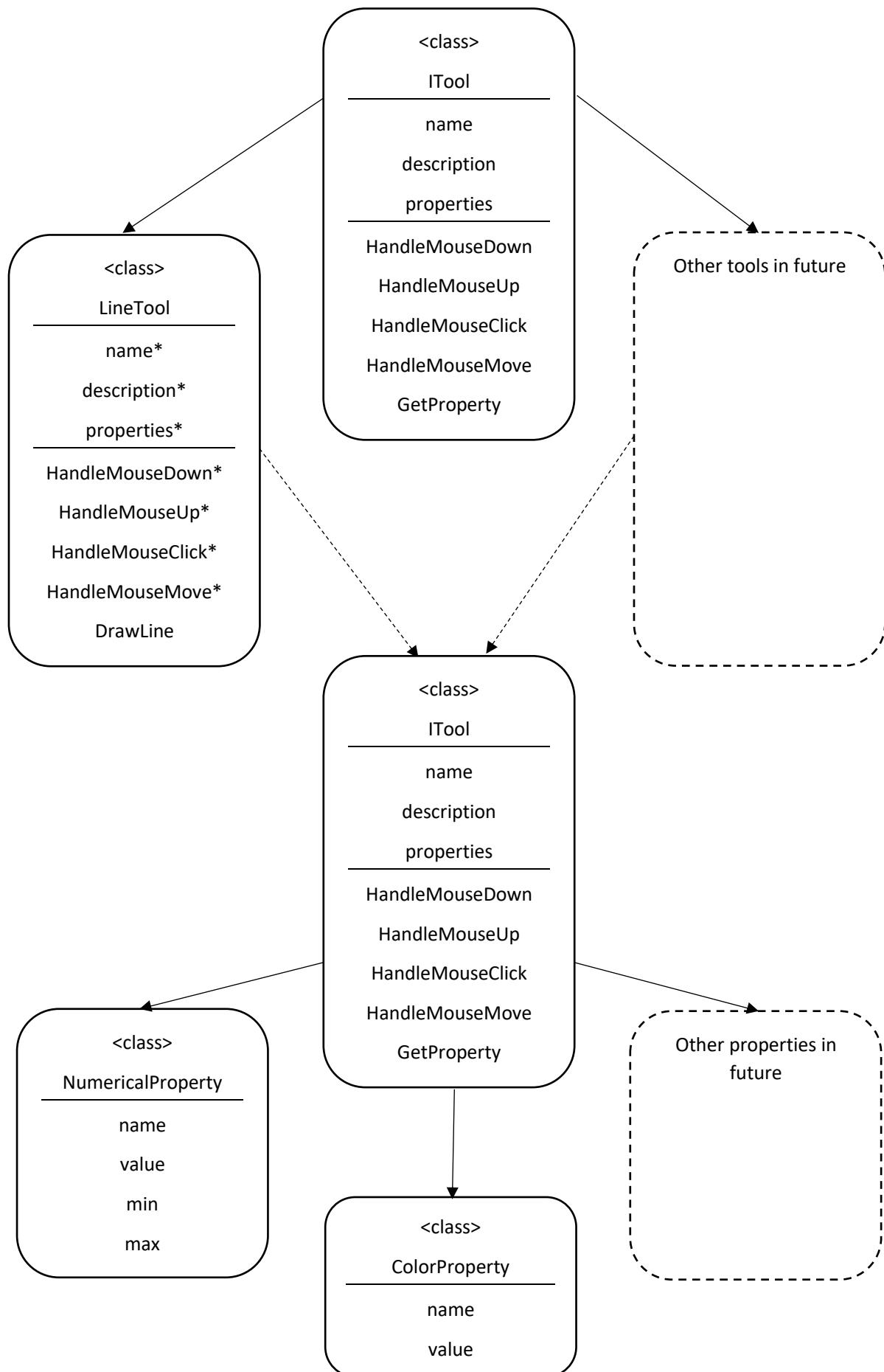
#### Properties

Property	Datatype	Justification
pixels	Array of Colours	Stores the colours needed in this class

### 3.1.4 Class Relation Diagram

---





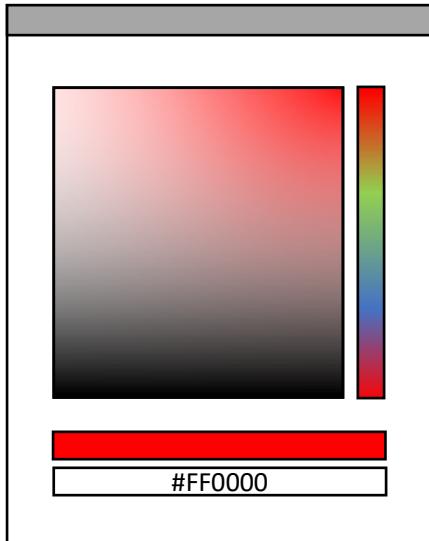
### 3.1.5 Algorithm Design

#### Algorithm 3.1 Form Controls

The algorithm form controls will be, from my analysis, inspired by the GIMP colour picker:



Initial Design

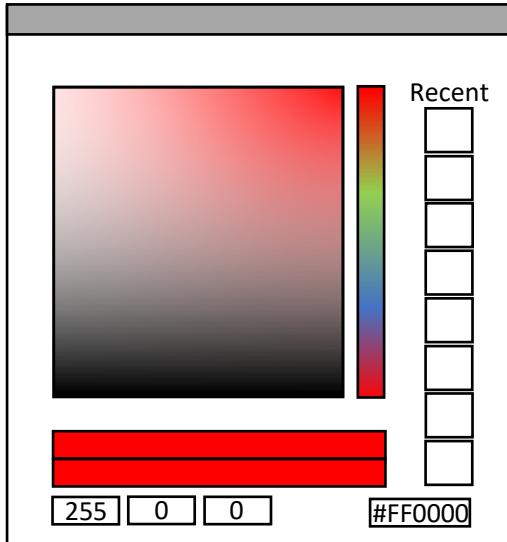


## Stakeholder feedback

I showed the initial design to my stakeholders, and they had the following changes to be made:

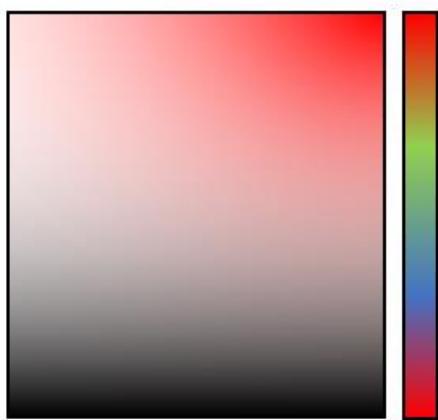
- “An option to see your old colour would be good”
- “If there were separate editable boxes for the R, G and B values that would be nice”
- “Is the box for the code editable? It should be”
- “There should be a place that stores your recently used colours”

## Updated Design

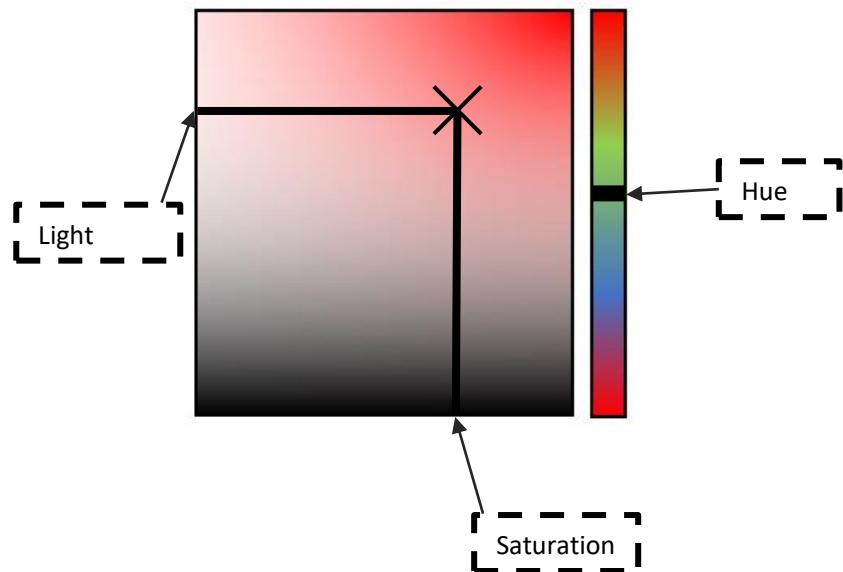


## Algorithm 3.2 Implementing Controls

### 3.2A Displaying Current Colour



The colour space can be designed using the HSL colour format. This is where colour is split into Hue (H), Saturation (S) and Light (L). On the diagram these 3 colours can be represented like so:



### 3.2B Converting HSL to RGB

The algorithm for converting between HSL and RGB is

When  $0 \leq H < 360$ ,  $0 \leq S \leq 1$  and  $0 \leq L \leq 1$ :

$$C = (1 - |2L - 1|) \times S$$

$$X = C \times (1 - |(H / 60^\circ) \bmod 2 - 1|)$$

$$m = L - C/2$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0^\circ \leq H < 60^\circ \\ (X, C, 0) & , 60^\circ \leq H < 120^\circ \\ (0, C, X) & , 120^\circ \leq H < 180^\circ \\ (0, X, C) & , 180^\circ \leq H < 240^\circ \\ (X, 0, C) & , 240^\circ \leq H < 300^\circ \\ (C, 0, X) & , 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

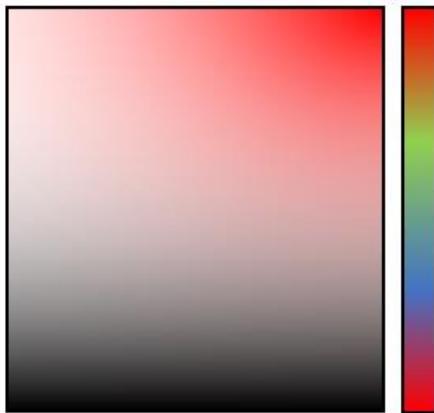
Pseudocode

Implementing the above diagram as pseudocode gives:

```
HSItoRGB(H, S, L) {
    C = (1 - Abs((2 * L) - 1)) * S
    X = C * (1 - Abs((H / 60) % 2) - 1)
    M = L - C/2
    IF H < 60 THEN
        r = C, g = X, b = 0
    ELSE IF H >= 60 && H < 120 THEN
        r = X, g = C, b = 0
    ELSE IF H >= 120 && H < 180 THEN
        r = 0, g = C, b = X
    ELSE IF H >= 180 && H < 240 THEN
        r = 0, g = X, b = C
    ELSE IF H >= 240 && H < 300 THEN
        r = X, g = 0, b = C
    ELSE IF H >= 300 && H < 360 THEN
        r = C, g = 0, b = X
    END IF
    R = (r + m) * 255
    G = (g + m) * 255
    B = (b + m) * 255
    RETURN new Colour(R, G, B)
}
```

### 3.2C Generating the colour square

Now that HSL can be converted to RGB, the colour square can be generated reasonably simply:



#### Pseudocode

```
FOR x = 0 to 100
    FOR y = 0 to 100
        currentColour = HSLtoRGB(hue, x/100, y/100)
        setPixel(x,y,currentColour)
    NEXT
NEXT
```

The other controls are much more straightforward in design, so there is no merit in designing them here.

#### Unit Test

	Test	ID	Expected Result	Comment
<i>Set Hue to Red, click top-left corner</i>	1	White		This tests that the top left corner is functional
<i>Click top-right corner</i>	2	Red		This tests that the top left corner is functional
<i>Click bottom-left corner</i>	3	Black		This tests that the bottom left corner is functional
<i>Click bottom-right corner</i>	4	Black		This tests that the bottom right corner is functional
<i>Click three random positions on the colour square</i>	5	Correct corresponding colour outputs		This tests that other areas on the colour square are valid
<i>Click three random positions on the hue slider</i>	6	Correct corresponding colour square, and output also changes		This tests that the hue slider can be changed and the rest of the form updates
<i>Set RGB values to 255,0,0</i>	7	Red, hue adjusts appropriately		This tests that a standard colour can be implemented

<i>Set RGB values to 255,255,255</i>	<b>8</b>	White	This tests that a maximum brightness colour can be implemented
<i>Set RGB values to 0,0,0</i>	<b>9</b>	Black	This tests that the darkest colour can be entered
<i>Set RGB values to -1,0,0</i>	<b>10</b>	Not accepted, round -1 to 0	This tests that an incorrect value is adjusted up appropriately
<i>Set RGB values to 256,255,255</i>	<b>11</b>	Not accepted, round 256 to 255	This tests that an incorrect value is adjusted down appropriately
<i>Set RGB values to 255,255</i>	<b>12</b>	Not accepted, missing value substituted for 0	This tests that if a value is left out, it will be substituted

### Algorithm 3.3 Returning Colour

This will be implemented using the knowledge that in C# all classes (including colour) are passed by reference.

Pseudocode

So in the workspace form, the colour designer can be called by:

```
RGBPicker newPicker = new RGBPicker(colortochange)
```

RGBPicker's constructor will look like so:

```
RGBPicker(color) {  
    storedColour = color  
    show form  
}
```

Then RGBPicker will return the colour by setting storedColour to the result.

## Algorithm 3.4 Brush Size

### 3.4A Storing Current Brush

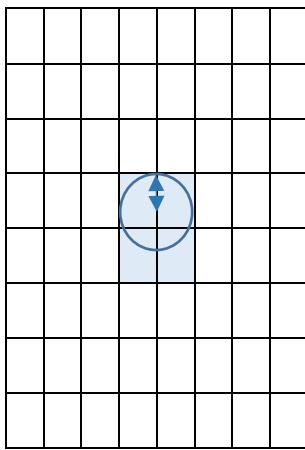
In order to determine the size of the brush, there must be a way to store what the brush looks like. In order to do this, there must be a store for the current brush.

Pseudocode

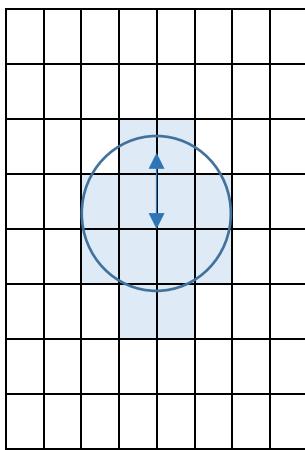
```
List currentBrush = new List of Points  
currentBrush.Add(0,0) // brush has a set point at the origin  
currentBrush.Add(-2,-2) // brush has a point 2 above and 2 to left of origin  
currentBrush.Add(2,2) // brush has a point 2 below and 2 to right of origin
```

### 3.4B Constructing current brush

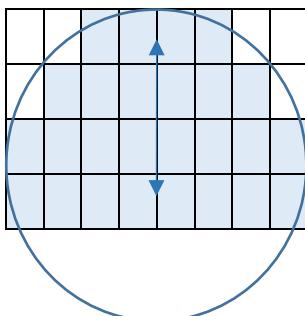
When generating brushes from certain sizes, it can be primarily planned by laying expectations:

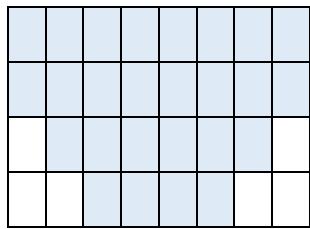


Brush size 1



Brush size 2



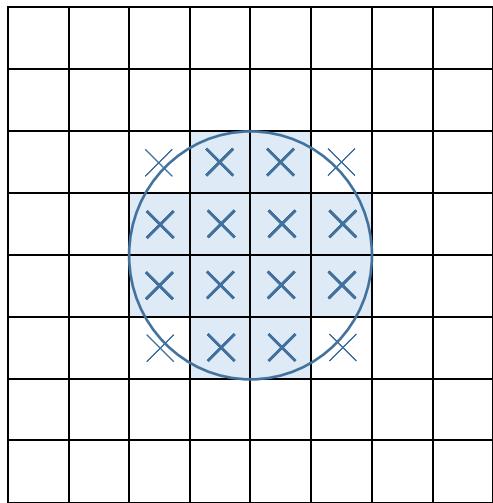


Brush size 4

In this case, it is clear that the size of the brush is determined by its radius.

In order to start generating the brushes, an **exhaustion algorithm** can be used. In this case, every possible pixel that can be a part of the brush is tried, and if it is inside the circle, it is added to the dimensions of the brush.

In the case of the Brush Size 2, every point marked with an x is checked.

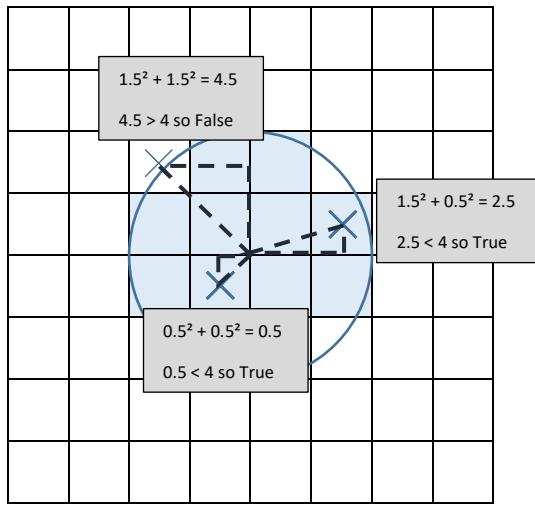


Generating the locations of these crosses is simple, the coordinates of each pixel is generated, and then 0.5 is added to centre it inside the pixel. This makes the pseudocode:

```
FOR x = brushSize*-1 TO brushSize
    FOR y = brushSize*-1 TO brushSize
        IF IsPointInCircle(x+0.5, y+0.5) THEN
            currentBrush.Add(x,y)
        END IF
    NEXT
NEXT
```

To check whether a point is inside a circle, simple Pythagoras can be used. The points distance to the origin can be calculated ( $x^2+y^2$ ), and if it is less than the brushSize<sup>2</sup>, then the point is inside the circle.

So again coming back to the size 2 brush:



In algorithmic form:

```
IsPointInCircle(x,y) {
    IF ((x*x)+(y*y) <= (brushSize*brushSize)) THEN
        RETURN true
    ELSE
        RETURN false
    END IF
}
```

### Algorithm 3.5 Brush Colour

When needing to change the brush colour, the RGBPicker form can be created, and edited from there.

## Algorithm 3.6 Continuous Lines

This code is to fix a problem that was noted during the first session of stakeholder feedback.

“The current brush is a very simple affair, however it means that when drawing larger lines there is the potential for gaps. This will be resolved later when the brush is fully implemented”



### 3.6A Bresenham's Line Algorithm

In order to implement this, straight lines must be drawn between any gaps in the pixels. In order to draw straight lines in this case, an algorithm known as **Bresenham's Line Algorithm** can be used. It is a well-established and efficient way of drawing a line between two points.

## Psuedocode

```
DrawLine(x1,y1,x2,y2) {
    IF x1 > x2 THEN
        tempX = x1
        tempY = y1
        x1 = x2
        y1 = y2
        x2 = tempX
        y2 = tempY
    END IF

    IF (x2 - x1) > Abs((y2 - y1)) // If change X is bigger than change Y
        A = 2 x (y2 - y1)
        B = A - (2 x (x2 - x1))
        P = A - (x2 - x1)
        currentY = y1

        FOR currentX = x1+1 TO x2
            IF P < 0 THEN
                P += A
            ELSE
                IF (y2 > y1) THEN
                    currentY++
                ELSE
                    currentY--
                END IF
                SetPixel(currentX,currentY)
                P += B
            END IF
        NEXT
    ELSE
        A = 2 x (x2 - x1)
        B = A - (2 x (y2 - y1))
        P = A - (y2 - y1)
        currentX = x1
    END IF
}
```

If the second pixel is further left than the first, they are swapped. This decreases the amount of potential point configurations

```

FOR currentY = y1+1 TO y2
    IF P < 0 THEN
        P += A
    ELSE
        IF (x2 > x1) THEN
            currentX++
        ELSE
            currentX--
        END IF
        SetPixel(currentX,currentY)
        P += B
    END IF
NEXT
END IF
}

```

### 3.6B Setting the pixels

When the brush is moved, the above algorithm can be used to set pixels between the points

Pseudocode

```

HandleMouseMove(oldPoint, newPoint) {
    SetPixel(oldPoint.x, oldPoint.y)
    DrawLine(oldPoint.x, oldPoint.y, newPoint.x, newPoint.y)
    SetPixel(newPoint.x, newPoint.y)
}

```

### Unit Test

Test	ID	Expected Result	Comment
<i>Draw a 1 width line across middle of image</i>	1	A line appears and is displayed on the image	This tests that a normal line can be drawn on the image
<i>Draw a 1 width line at the edge of the image</i>	2	A line is drawn at the edge	This tests that a normal line can be drawn at the edge of the image
<i>Draw a 4 width line across middle of image</i>	3	A thicker line is displayed on the image	This tests that a thick line can be drawn on the image
<i>Draw a 4 width line at the edge of the image</i>	4	A thicker line is displayed at the edge	This tests that a thick line can be drawn at the edge of the image

<i>Move the mouse rapidly drawing a 1 width line</i>	5	The drawn line has no visible breaks	This tests whether the filled line algorithm works correctly
<i>Move the mouse rapidly drawing a 4 width line</i>	6	The drawn line has no visible breaks	This tests whether the filled line algorithm works correctly with thicker brushes

## Algorithm 3.7 Designing Undo Interface

The interfaces should be designed and implemented in accordance to class design 3.1.3.1 & 3.1.3.2. The reason for the implementation this way is to have a common method of performing (and undoing) any action in the program.

## Algorithm 3.8 Implementing Undo Interface

### 3.8A New methods

The workspace should then include three new public methods:

```
PerformAction(action) {  
    action.Do(this)  
    RecordAction(action)  
}  
  
PerformActionSilent(action) {  
    Action.Do(this)  
}  
  
RecordAction(action) {  
    pastActions.add(action)  
    futureActions.flush()  
}
```

The purpose of these two data structures will be explained in 3.8B

The three methods are **justified** as:

- PerformAction is the general use action that will be performed and then recorded (in case it needs to be undone)
- PerformSilentAction is used in the case that an action is done to the image that should *not* be recorded, should be used very sparsely and in conjunction with RecordAction
- RecordAction is used after a silent action is performed, to make sure the action is recorded. These two alternatives are used when an action consists of many smaller actions, but only the larger change should be recorded (e.g. a brush draw may consist of many individual smaller line actions, but only the stroke as a whole should be recorded).

### 3.8B Dealing with previous actions

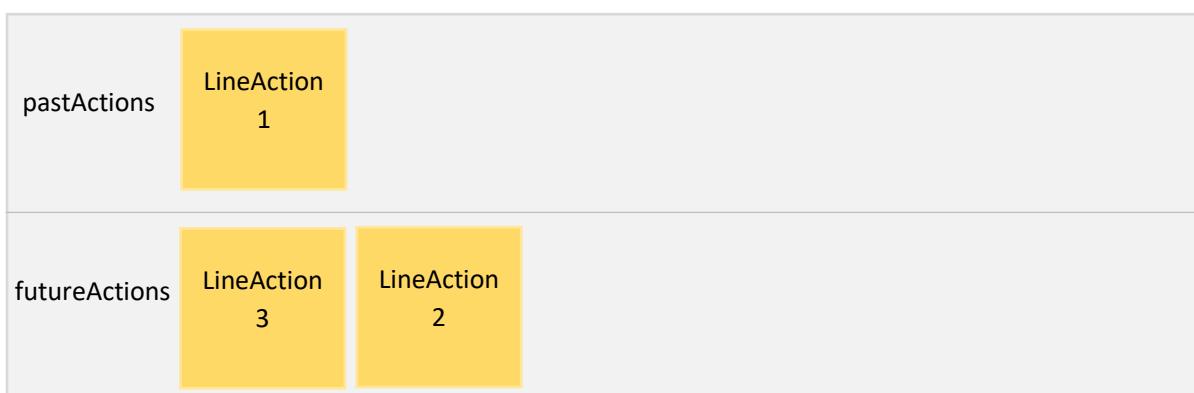
In order to implement the undoing and redoing that is needed by the program, two data structures are needed, being the pastActions and futureActions

Property	Datatype	Justification
pastActions	Stack of Actions	Stores the actions previously performed by the program. Only the last action needs to be accessed so a stack is needed
futureActions	Stack of Actions	Stores the actions that will be performed in the future, necessary for redoing (undo moves into the 'past', redo moves back from the past to its 'future', which is the present)

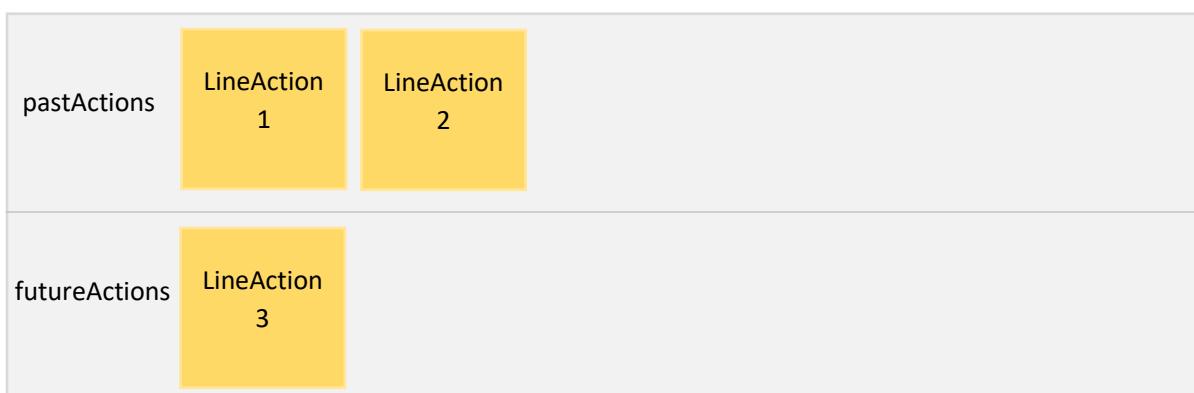
In diagrammatic form, if the user draws three lines then the contents of the data structures should be:



Then if the user undoes two of those lines then two actions should be moved into the future:



Then if the user chooses to redo a line the first action from the future is popped



Finally, if the user makes a new line the futureActions stack is cleared, it's no longer relevant.



Thus, the pseudocode for adding actions, undoing and redoing becomes:

```
RecordAction(action) {
    pastActions.Push(action)
    futureActions.Flush()
}

Undo() {
    lastAction = pastActions.Pop()
    lastAction.Undo()
    futureActions.Push(lastAction)
}

Redo() {
    nextAction = futureActions.Pop()
    nextAction.Do()
    pastActions.Push(nextAction)
}
```

### Algorithm 3.8C Implementing PixelAction

The PixelAction interface is designed to encapsulate any action that changes the pixels of the image, so implementing its Do() and Undo() methods is comparatively easy.

#### Pseudocode

```
Do() {  
    FOR EACH pixel IN newPixels  
        SetPixel(pixel.location,pixel.colour)  
    NEXT  
}  
  
Undo() {  
    FOR EACH pixel IN oldPixels  
        SetPixel(pixel.location,pixel.colour)  
    NEXT  
}
```

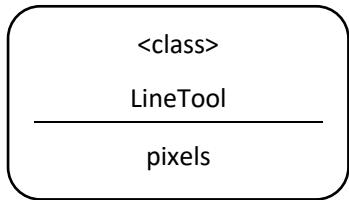
#### Unit Test

	Test ID	Expected Result	Comment
<i>Undo an action (on default image)</i>	1	Cannot be done	Tests that an action cannot be undone if no action has been performed
<i>Redo an action (on default image)</i>	2	Cannot be done	Tests that an action cannot be redone if there is no future
<i>Perform an action</i>	3	Action is performed onto image	Tests that actions can still be performed
<i>Undo the action</i>	4	Action is undone	Tests that an action can be undone
<i>Redo the action</i>	5	Action is redone	Tests that an action can be redone
<i>Undo the action</i>	6	Action is undone	Tests that a previously redone action can be undone
<i>Redo the action</i>	7	Action is redone	Tests that a previously undone action can be redone
<i>Perform another action</i>	8	Action is done	Tests that actions can still be done from here
<i>Undo both actions</i>	9	Both actions redone	Tests that multiple actions can be redone
<i>Undo an action</i>	10	Cannot be done	Tests that the program stops undoing when there are no previous actions

<i>Redo both actions</i>	11	Both actions redone	Tests that both actions can be redone
<i>Redo an action</i>	12	Cannot be done	Tests that the program stops redoing when there is no future
<i>Undo an action, and perform a new action</i>	13	New action is performed on top of old action	Tests that actions can be performed on a previously undone action
<i>Redo an action</i>	14	Cannot be done	Tests that the future is cleared when a new action is performed

### Algorithm 3.9 Implementing Layer Class

The structure of the layer class can be implemented in accordance to [3.1.3.5](#)



Then the Image class should be updated to contain a list of layers, as well as a reference to its currentLayer

Property	Datatype	Justification
layers	List of Layers	Stores all of the image's current layers
currentLayer	Layer	Contains a reference to the layer that is currently being edited

### Algorithm 3.10 Allow layer editing

In order to achieve this, there must firstly be a way to change which layer is selected. This can be done using the following code:

```
ChangeLayer(newLayerID) {
    currentLayer = layers[newLayerID]
}
```

Then to begin editing the layer, the SetPixel and GetPixel functions must be edited

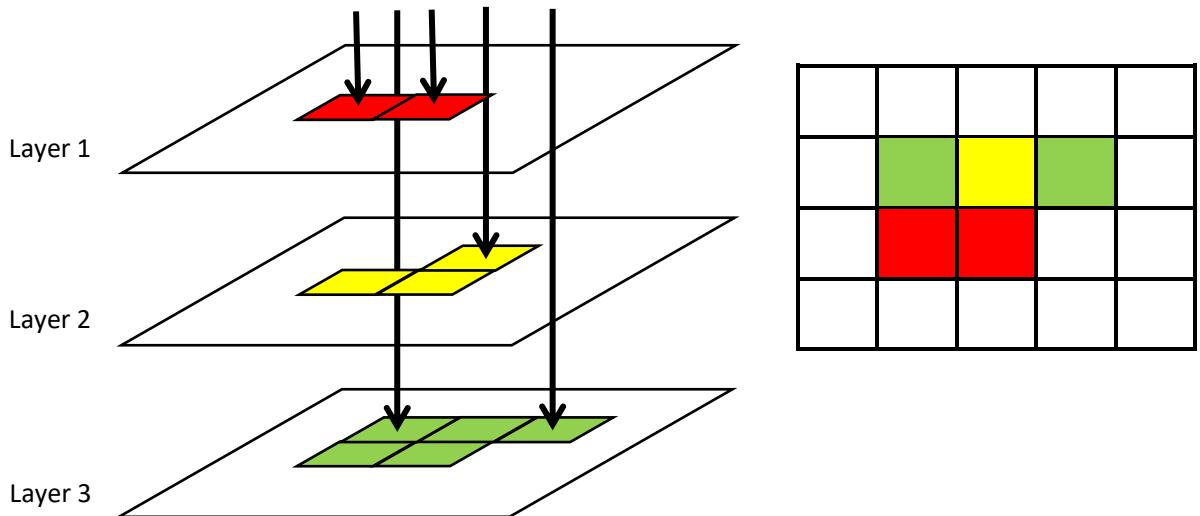
```
SetPixel(location, colour) {
    currentLayer.pixels[location] = colour
}

GetPixel(location) {
    RETURN currentLayer.pixels[location] = colour
}
```

This means that **no functionality is lost** and despite internal changes, externally nothing has changed, meaning that the code is more compatible

### Algorithm 3.11 Display layer

In order to create the image now, when displaying a pixel each layer must be iterated through to find the first non-blank pixel, or in other words the process must look like:



Where the layer blocks the view to the layer below it, when a non-transparent pixel is seen.

This makes the pseudocode:

```
DisplayPixel(x,y) {  
    FOR EACH layer IN layers  
        IF layer.pixels[x,y] is transparent  
            STOP REPEAT  
        ELSE  
            //Draw pixel on image (as before)  
        END IF  
    NEXT  
}
```

### Algorithm 3.12 Add layer selector

In order to allow the user to manipulate the layer, a layer tool must be added that allows the user to select which layer is active, deselect the layer, and show/hide a layer. A potential design for this could look like:

Initial Design



Stakeholder feedback

After showing the stakeholders the design, I got the following feedback:

- “Looks solid, but will you be able to delete a layer?”
- “How can I deselect a layer?”
- “What about the rearranging tools, deleting and adding new layers?”

Addressing this feedback leads to:

Updated Design



### 3.11A Drawing icon for layer

Each layer must contain a small icon to let the user know roughly what the layer looks like, drawing this image is much simpler as the entire image will always be displayed, the only difficulty will be downscaling the image to its smaller size.

A simple way to resolve this is to iterate through each pixel and find out which file pixel to draw from it, making the pseudocode:

```
DrawIcon() {
    FOR x = 1 TO iconHeight
        FOR y = 1 TO iconHeight
            imageX = (x * imageWidth) / iconWidth
            imageY = (y * imageHeight) / iconHeight
            DrawRectangle(x,y,1,1,colours[imageX,imageY])
        NEXT
    NEXT
}
```

### Unit Test

	Test	ID	Expected Result	Comment
<i>Select the top layer and draw onto it</i>		1	There is drawing on layer	This tests whether a layer can be drawn onto
<i>Attempt to delete the layer</i>		2	The layer cannot be deleted as it is the only one	This tests whether the program is prevented from having 0 layers
<i>Select lower layer and draw onto it at same location as top layer</i>		3	There is drawing on lower layer, but it is obviously below	This tests whether the user can tell what layer they are drawing onto
<i>Move lower layer up</i>		4	The lower layers moves on top of previous top layer	This tests whether the drawing adjusts when the layers are moved
<i>Move the highest layer up</i>		5	Should be impossible as it cannot go up	This tests whether the highest layer cannot be moved too high
<i>Move the lowest layer down</i>		6	Should be impossible as it cannot go down	This tests whether the lowest layer cannot be moved too low
<i>Add a new layer</i>		7	A new (empty) layer is created	This tests whether a new layer can be made
<i>Add 10 new layers</i>		8	Many new layers are added	This tests whether the system can handle many layers
<i>Layers are labelled then randomly rearranged</i>		9	The layers are moved	This tests whether the program can handle moving many layers
<i>The 11 layers are deleted</i>		10	The layers disappear in the order in which they are added in the list	This tests that the order is maintained when removing many rearranged elements from the list

<i>The final layer is deleted</i>	11	Prevented as there is only one layer	This tests whether the removal prevention code still works
-----------------------------------	----	--------------------------------------	--

## Algorithm 3.13 & 3.14 Implementing eraser

This was initially thought to be a more complex class, however the eraser can simply be implemented as a special LineTool, where its colour is set to `Color.Transparent`, which has an alpha (opaqueness) value of 0.

## Algorithm 3.15 Calculating Shapes

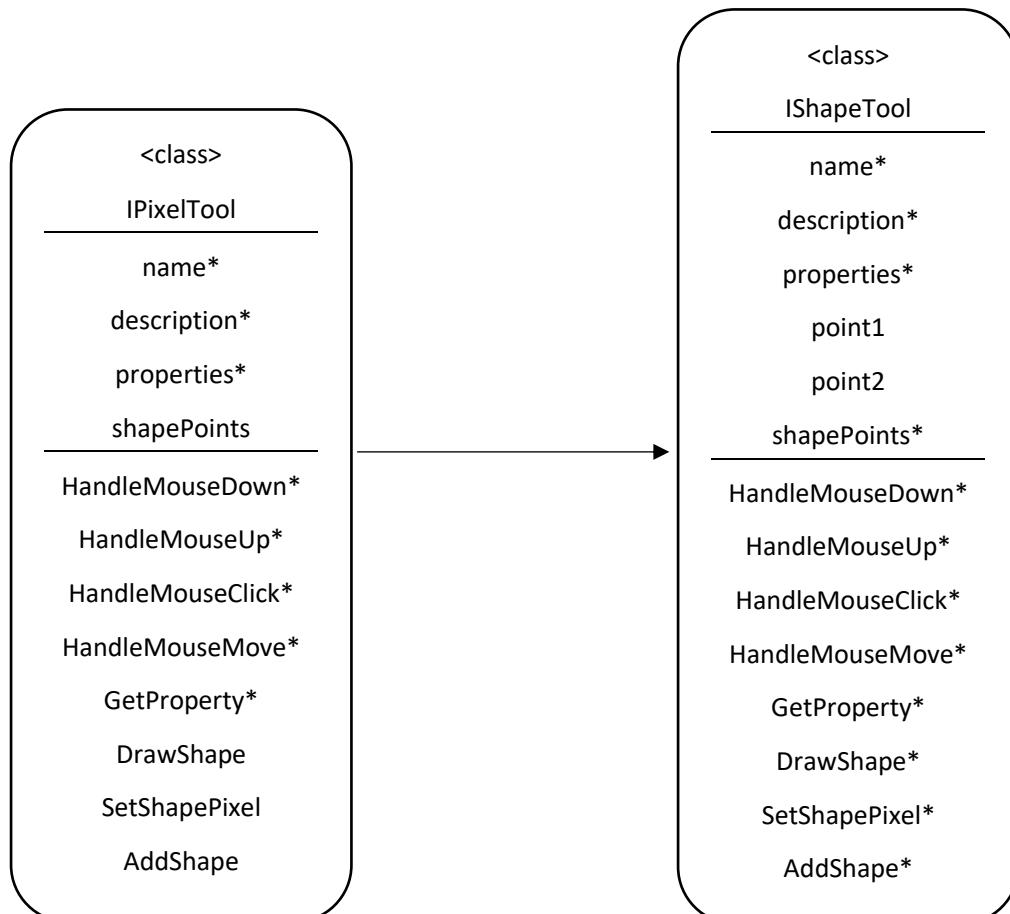
In order for the shape tool to work, there must be some simple shape calculation tools. Drawing from one of the image editing tools outlined in the analysis, **Paint**, there will be four implemented shapes:

- Line
- Rectangle
- Circle

### 3.15A Generic Shape Calculations

#### Common Class

In order for shapes to be implemented easily, there will be a base class that they inherit from `IShapeTool`, which inherits from `IPixelTool`, which inherits from `ITool`. This abstract class will encapsulate the generic process of adding a shape; click once, click somewhere else, shape is made. It will leave the calculations to make the shape up to its child classes. This means that the class will be implemented by:



The new properties are:

Property	Datatype	Justification
point1	FilePoint	The location of the top-left corner of the shape
point2	FilePoint	The location of the bottom-right corner of the shape
shapePoints	List of FilePoints	The points of the pixels that the shape will set

Method	Params	Return type	Justification
DrawShape	None (uses local Variables)	None	Draws the shape
AddShapePoint	Integer x, Integer y	None	Adds a point to be set to the shape. Will be outputted when AddShape is called
AddShape	None	None	Outputs the completed shape to the image

### Handling Clicks and resolving points

Handling the Click events is reasonably simple, just requiring a check for whether the first point has been set or not

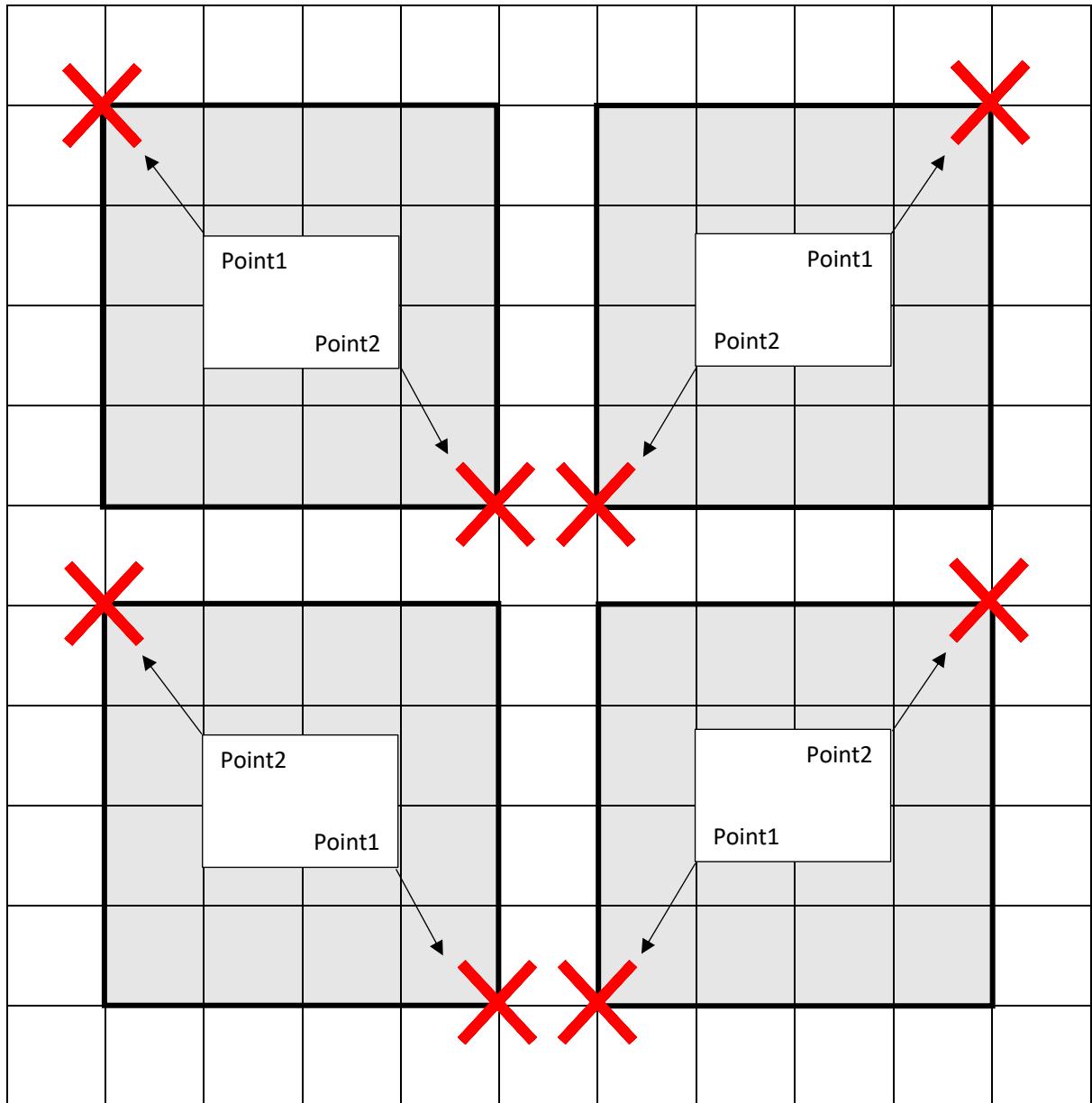
```
HandleMouseClick(clickLocation, button) {
    IF point1 is null THEN
        point1 = clickLocation
    ELSE
        point2 = clickLocation
        ResolveLocations()
        DrawShape()
        AddShape()
    END IF
}
```

After the second point has been set, the shape is created. This requires two steps:

- Resolving the two points to make them uniform
- Drawing it

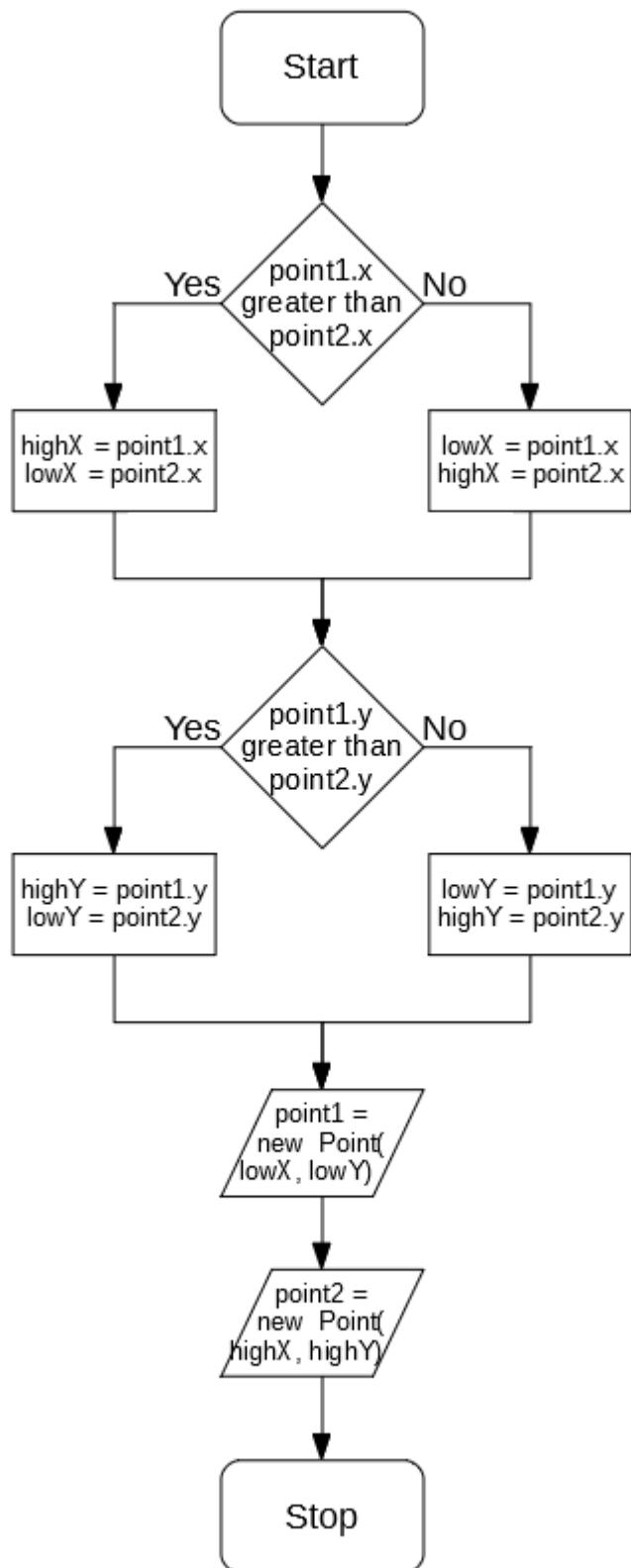
## Resolving points

There is a need to resolve points, because it is unknown the order in which the user will create the two points, and there are four potential outcomes:



Having four potential situations is not desirable, as it means programming for four cases. To resolve this, the points can be modified so that, no matter what, they look like the top-left situation.

In order to do that, the general algorithm must involve finding the highest and lowest out of each category, or in a flow chart:



This now means that point1 will always have lower coords than point2

### 3.15B Line Drawing

LineDrawing can be accomplished by using Bresenham's Line Algorithm code implemented previously in [Algorithm 3.6](#), requiring no additional pseudocode

### 3.15C Rectangle Drawing

Drawing a rectangle between two points can be accomplished using iterations:

```
DrawShape() {
    FOR x = point1.x TO point2.x
        FOR y = point1.y TO point2.y
            AddShapePoint(x,y)
        NEXT y
    NEXT x
}
```

### 3.15D Circle Drawing

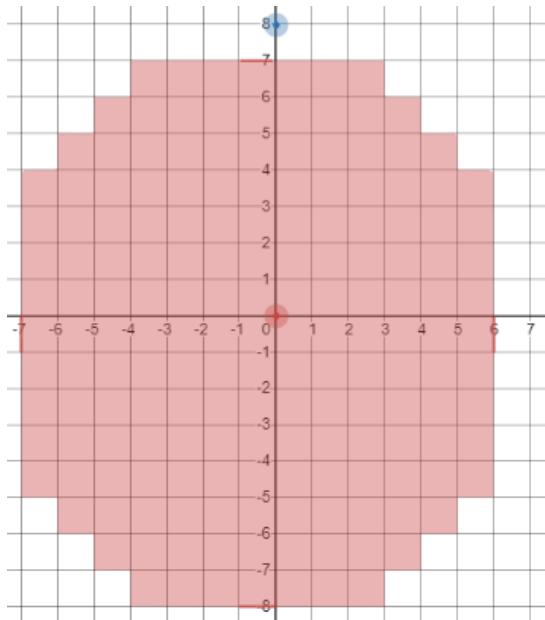
Circle drawing is a more in-depth process, but can be implemented reasonably simply using a similar process to [Algorithm 3.4](#). All potential candidate points are checked as to whether they appear in the circle (or ellipse) generated, if so, draw it.

This means a condition for an ellipse will need to be defined. In order to do this a standard representation can be used:

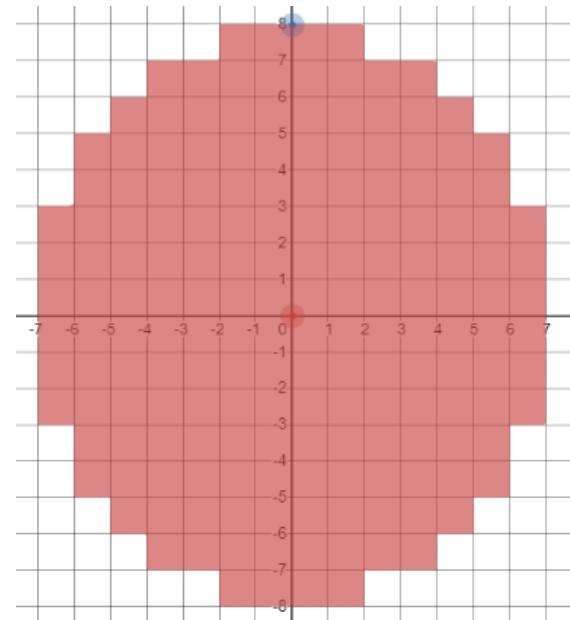
$$\frac{(x - (x' + 0.5))^2}{w^2} + \frac{(y - (y' + 0.5))^2}{h^2} \leq 1$$

*Representation of a sphere of width w and height h with a centre of (x',y')*

0.5 is subtracted from each point in order to make sure the *centre* of each pixel is checked, as this most accurately represents where the pixel should go, and means there is no bias between top left pixels and bottom right pixels.



*Without correction*



*With correction*

So every potential x and y can be checked to see if it satisfies that condition, if so, draw the circle

```
DrawShape() {
    centreLocX = (point1.X + point2.X)/2 + 0.5
    centreLocY = (point1.Y + point2.Y)/2 + 0.5
    widthSquared = (point2.X - point1.X) ^ 2
    heightSquared = (point2.Y - point1.Y) ^ 2

    FOR x = point1.X TO point2.X
        FOR y = point1.Y TO point2.Y
            IF ((x - centreLocX) ^ 2) / widthSquared +
                ((y - centreLocY) ^ 2) / heightSquared <= 1 THEN
                AddShapePoint(x,y)
            END IF
        NEXT y
    NEXT x
}
```

## Unit Test

	Test	ID	Expected Result	Comment
Create line with points (5,5) & (10,10)	1	A line is drawn from (5,5) to (10,10)	This tests that a line can be created using an above left first point	
Create line with points (5,10) & (10,5)	2	A line is drawn from (5,10) to (10,5)	This tests that a line can be created using an below left first point	
Create line with points (10,5) & (5,10)	3	A line is drawn from (10,5) to (5,10)	This tests that a line can be created using an below right first point	
Create line with points (10,10) & (5,5)	4	A line is drawn from (10,10) to (5,5)	This tests that a line can be created using an above right first point	
Create line with points (5,5) & (10,8)	5	A shorter line is drawn	This tests that a shorter, non-square line can be drawn	
Create line with points (5,5) & (8,10)	6	A thinner line is drawn	This tests that a thinner, non-square line can be drawn	
Create line with points (5,5) & (10,5)	7	A single row is drawn	This tests that a shape can be drawn that has same Y	
Create line with points (5,5) & (5,10)	8	A single column is drawn	This tests that a shape can be drawn that has same X	
Create rectangle with points (5,5) & (10,10)	9	A rectangle is drawn from (5,5) to (10,10)	This tests that a rectangle can be created using an above left first point	
Create rectangle with points (5,10) & (10,5)	10	A rectangle is drawn from (5,10) to (10,5)	This tests that a rectangle can be created using an below left first point	
Create rectangle with points (10,5) & (5,10)	11	A rectangle is drawn from (10,5) to (5,10)	This tests that a rectangle can be created using an below right first point	
Create rectangle with points (10,10) & (5,5)	12	A rectangle is drawn from (10,10) to (5,5)	This tests that a rectangle can be created using an above right first point	
Create rectangle with points (5,5) & (10,8)	13	A shorter rectangle is drawn	This tests that a shorter, non-square rectangle can be drawn	
Create rectangle with points (5,5) & (8,10)	14	A thinner rectangle is drawn	This tests that a thinner, non-square rectangle can be drawn	
Create rectangle with points (5,5) & (10,5)	15	A single row is drawn	This tests that a shape can be drawn that has same Y	
Create rectangle with points (5,5) & (5,10)	16	A single column is drawn	This tests that a shape can be drawn that has same X	

<i>Create circle with points (5,5) &amp; (10,10)</i>	17	A circle is drawn from (5,5) to (10,10)	This tests that a circle can be created using an above left first point
<i>Create circle with points (5,10) &amp; (10,5)</i>	18	A circle is drawn from (5,10) to (10,5)	This tests that a circle can be created using an below left first point
<i>Create circle with points (10,5) &amp; (5,10)</i>	19	A circle is drawn from (10,5) to (5,10)	This tests that a circle can be created using an below right first point
<i>Create circle with points (10,10) &amp; (5,5)</i>	20	A circle is drawn from (10,10) to (5,5)	This tests that a circle can be created using an above right first point
<i>Create circle with points (5,5) &amp; (10,8)</i>	21	A shorter circle is drawn	This tests that a shorter, non-square circle can be drawn
<i>Create circle with points (5,5) &amp; (8,10)</i>	22	A thinner circle is drawn	This tests that a thinner, non-square circle can be drawn
<i>Create circle with points (5,5) &amp; (10,5)</i>	23	A single row is drawn	This tests that a shape can be drawn that has same Y
<i>Create circle with points (5,5) &amp; (5,10)</i>	24	A single column is drawn	This tests that a shape can be drawn that has same X

### Algorithm 3.16 Display & Algorithm 3.17 Log Undo

This can be accomplished by implementing the AddShapePoint function, as it needs to log any set pixels into a list

```
AddShapePoint(x,y) {
    shapePoints.Add(new FilePoint(x,y))
}
```

This means that any points are logged and then eventually outputted. To output them, this list can be iterated through to create a [PixelAction](#) object which will be executed by the image.

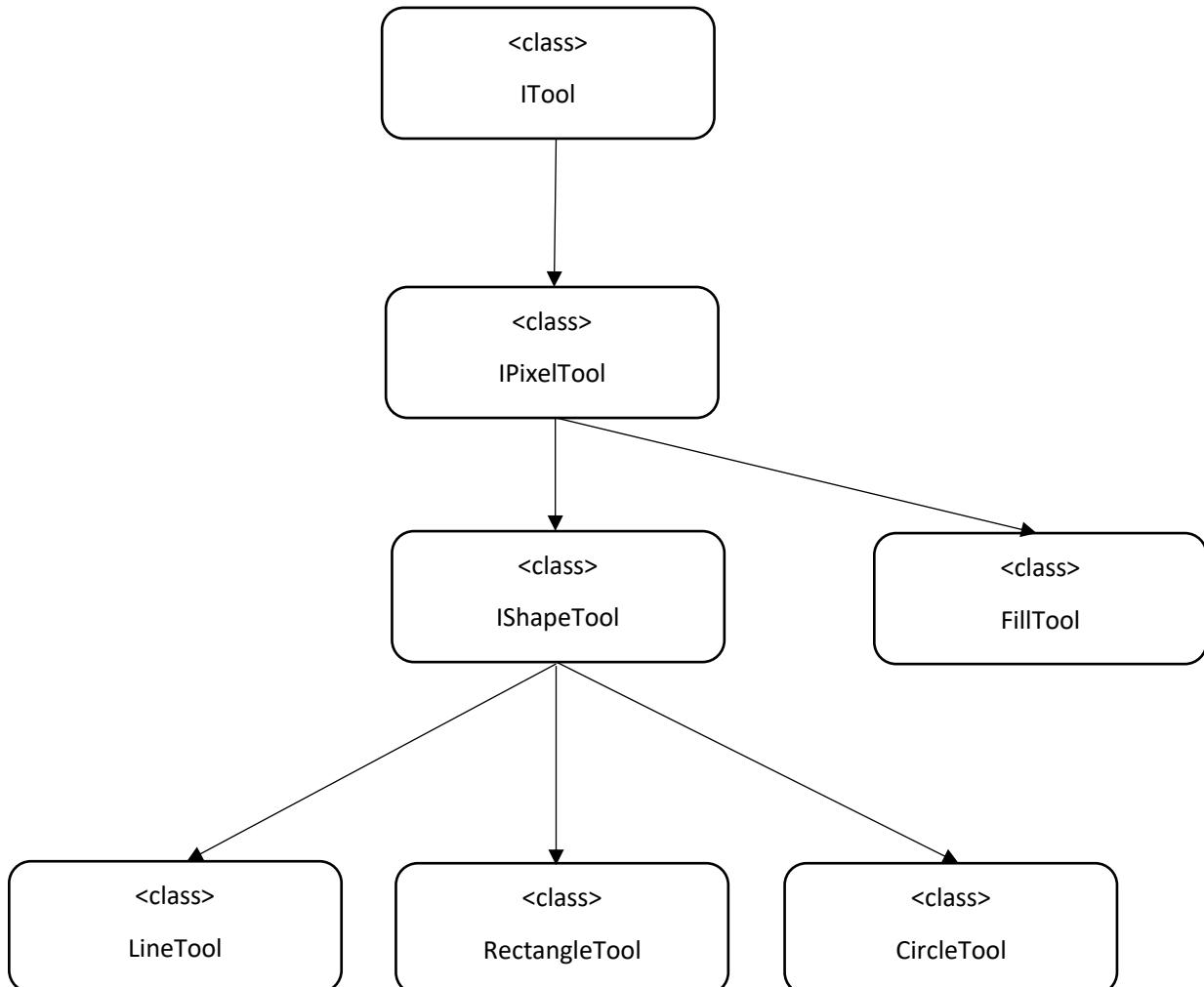
```
AddShape() {
    newAction = new PixelAction()
    newColour = GetProperty("colour")
    FOR EACH shapePoint IN shapePoints
        oldColour = image.GetPixel(x,y)
        newAction.AddPixel(x,y,oldColour,newColour)
    NEXT shapePoint
    image.PerformAction(newAction)
}
```

This demonstrates how the existing action classes can be used to safely perform an edit to the image.

### Algorithm 3.17 & Algorithm 3.18 Fill Tool

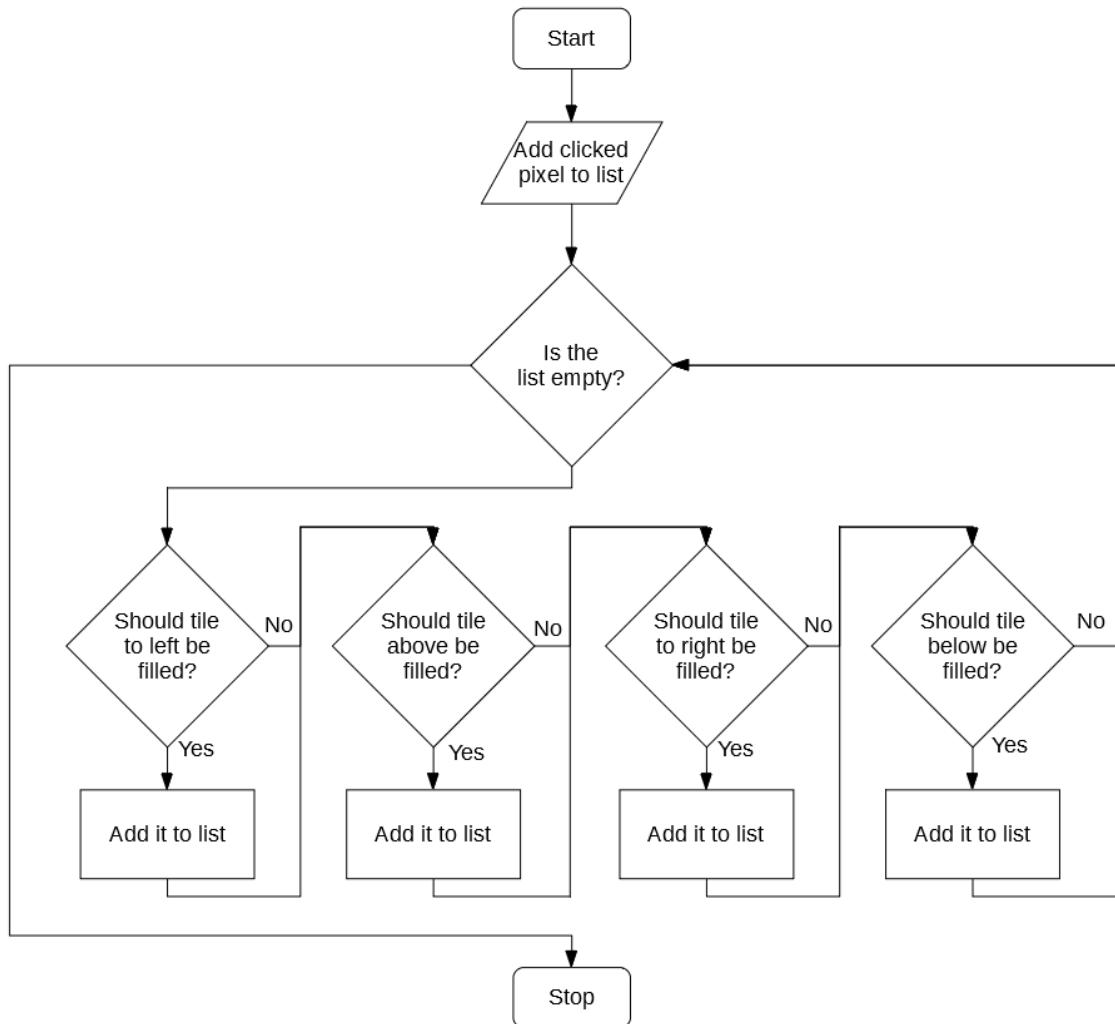
In order to implement the fill tool, a simple algorithm will be used. This is to save on programming time and a faster algorithm may not make much difference.

The FillTool will inherit from the existing IPixelTool. This makes the inheritance tree for the current tools:

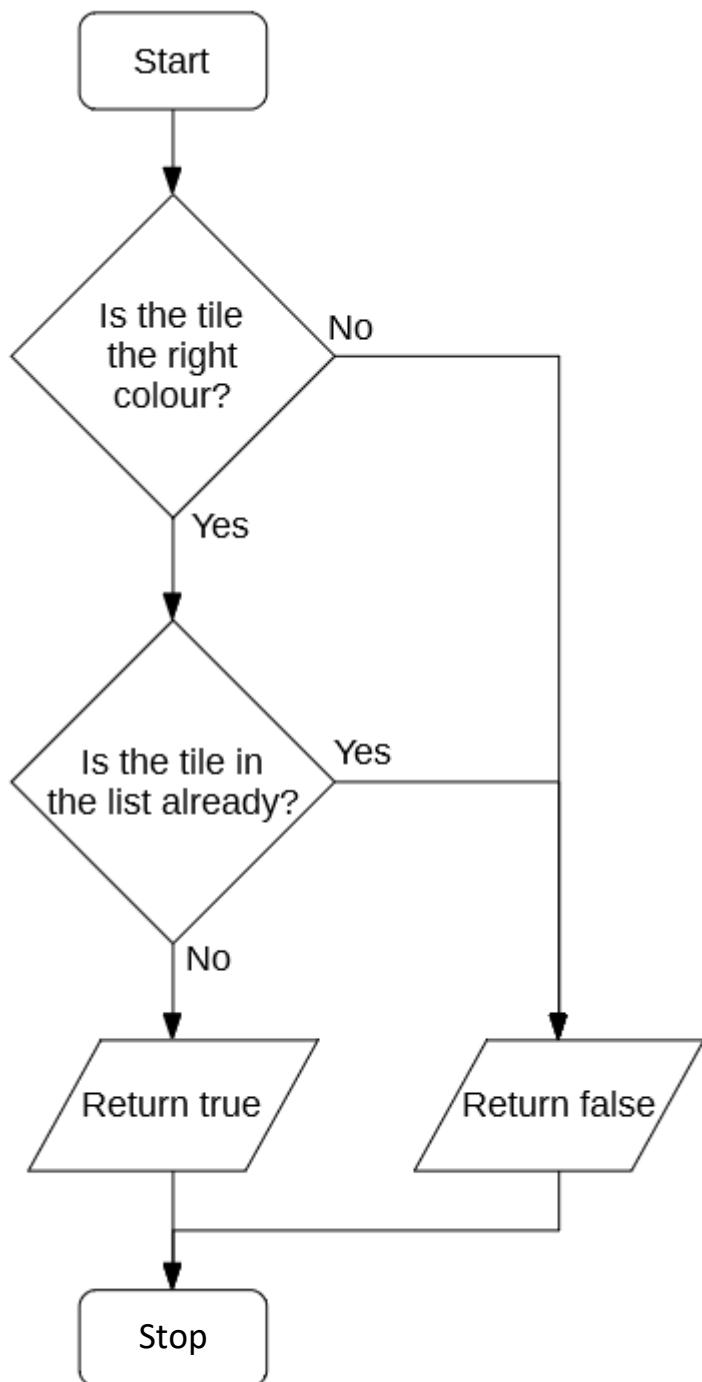


In order to create the fill algorithm, a simple structure will be used. When clicking on a starting point, that point will be parsed, and then the four adjacent points will be parsed, and so on, until the algorithm is complete.

In flowchart form, the algorithm for this becomes:



In order to determine whether a tile should be added to the list, the following algorithm can be used:

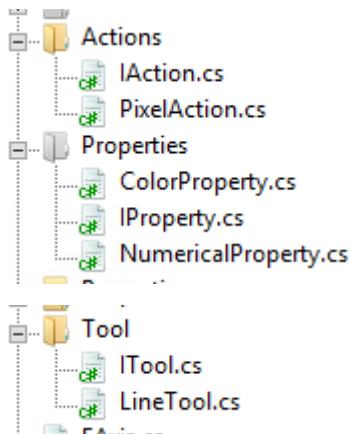


# 3.2 Development

## 05/11/2019 Class Design

---

Today I implemented the necessary classes for the next phase of SIMP. They have been organised into folders. They will be implemented later:



## 06/11/2019 Class Implementation

---

The following classes have been implemented:

### IAction

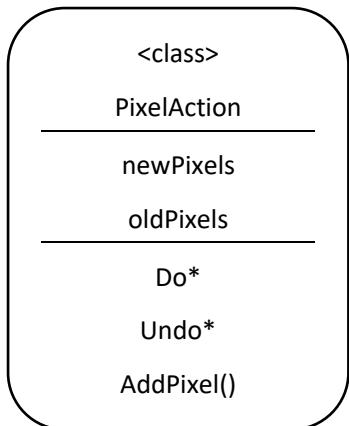
IAction has been implemented with its two functions in accordance to 3.1.3.1:

```
<interface>
  IAction
  _____
  Do()
  Undo()
```

```
public interface IAction
{
  void Do();
  void Undo();
}
```

### PixelAction

PixelAction has been partly implemented, its Do() and Undo() will be implemented later in development, in accordance to 3.1.3.2:



```
public class PixelAction : IAction
{
    private Dictionary<FilePoint,Color> oldPixels;
    private Dictionary<FilePoint,Color> newPixels;

    public PixelAction()
    {
        oldPixels = new Dictionary<FilePoint, Color>();
        newPixels = new Dictionary<FilePoint, Color>();
    }

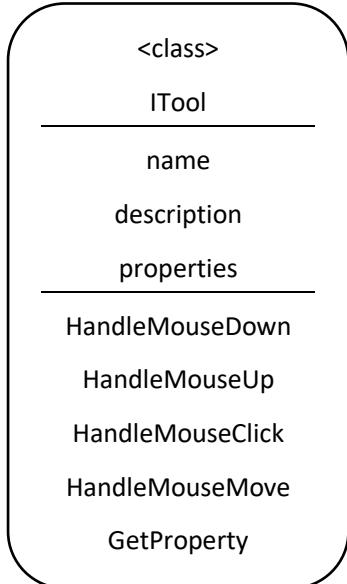
    public void AddPixel(FilePoint pixelLocation, Color oldColour, Color newColour) {
        // saves the old and new colours in the dictionary
        oldPixels[pixelLocation] = oldColour;
        newPixels[pixelLocation] = newColour;
    }

    public void Do(Workspace workspace) {
        //TODO: PixelAction Do()
        throw new NotImplementedException();
    }

    public void Undo(Workspace workspace) {
        //TODO: PixelAction Undo()
        throw new NotImplementedException();
    }
}
```

## ITool

ITool has received a major change, it has been changed from an Interface to an Abstract Class. This is because the GetProperty method is common code for each class, there should be no need for each class to implement its own GetProperty method. Thus an Abstract Class is a better fit:



```

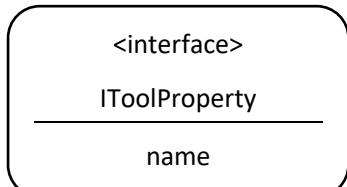
public abstract class ITool
{
    public string name;
    public string description;
    public List<SIMP.Properties.IProperty> properties

    public abstract void HandleMouseDown(FilePoint clickLocation, MouseButtons button);
    public abstract void HandleMouseUp(FilePoint clickLocation, MouseButtons button);
    public abstract void HandleMouseClick(FilePoint clickLocation, MouseButtons button);
    public abstract void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation);

    public SIMP.Properties.IProperty GetProperty(string propertyName) {
        //TODO: ITool GetProperty()
        throw new NotImplementedException();
    }
}
  
```

## IPROPERTY (IToolProperty)

The simple IToolProperty interface has been implemented, though has been renamed to IPROPERTY (as there is no obligation for it to be attached to a tool), and has also been changed to a static class, in accordance to 3.1.3.4:

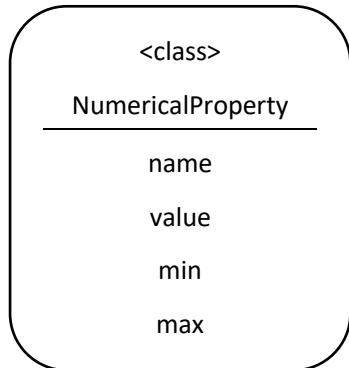


```

public abstract class IPROPERTY
{
    public string name;
}
  
```

## NumericalProperty

NumericalProperty has been implemented in accordance to 3.1.3.5:



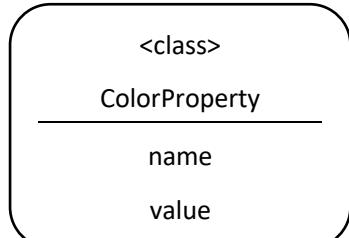
```
public class NumericalProperty : IProperty
{
    public int value;
    public int min;
    public int max;

    public NumericalProperty(string name, string value, string min, string max)
    {
        this.name = name;
        this.value = value;
        this.min = min;
        this.max = max;
    }
}
```

(Name is inherited from IProperty)

## ColorProperty

NumericalProperty has been implemented in accordance to 3.1.3.6:



```
public class ColorProperty : IProperty
{
    public Color value;

    public ColorProperty(string name, Color value)
    {
        this.name = name;
        this.value = value;
    }
}
```

## LineTool

The skeleton of LineTool has been implemented in accordance to 3.1.3.7:

```
public class LineTool : ITool
{
    public LineTool(string name, string description, Color color)
    {
        this.name = name;
        this.description = description;

        this.properties.Add(new ColorProperty("Color",color));
        this.properties.Add(new NumericalProperty("Brush Size",1,1,100));
    }

    public override void HandleMouseDown(FilePoint clickLocation, MouseButtons button) {
        //TODO: LineTool HandleMouseDown()
        throw new NotImplementedException();
    }

    public override void HandleMouseUp(FilePoint clickLocation, MouseButtons button){
        //TODO: LineTool HandleMouseUp()
        throw new NotImplementedException();
    }

    public override void HandleMouseClick(FilePoint clickLocation, MouseButtons button) {
        //TODO: LineTool HandleMouseClick()
        throw new NotImplementedException();
    }

    public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
        //TODO: LineTool HandleMouseMove()
        throw new NotImplementedException();
    }

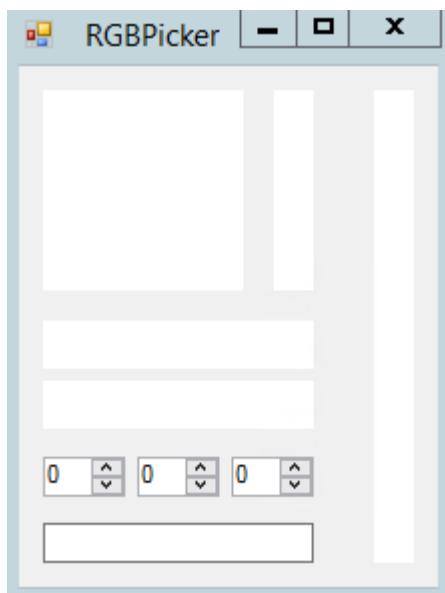
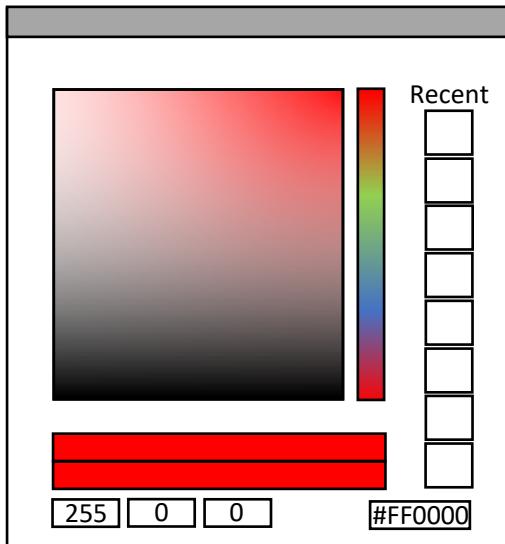
    public void DrawLine() {
        //TODO: LineTool DrawLine()
        throw new NotImplementedException();
    }
}
```

Layer

## 09/11/2019 RGB Picker design

### Form Design

Today the basic outline of the RGB picker has been implemented, in accordance to the updated design in Algorithm 3.1:



There has been a single change made, the hex code input box has been moved to below the color input boxes, as it more closely part of that section. However none of the code for the form has been completed.

## HSL to RGB

The algorithm for converting from HSL notation to RGB notation has been implemented, in accordance to Algorithm 3.2B:

```

HSLtoRGB(H, S, L) {
    C = (1 - Abs((2 * L) - 1)) * S
    X = C * (1 - Abs(((H / 60) % 2) -1))
    M = L - C/2
    IF H < 60 THEN
        r = C, g = X, b = 0
    ELSE IF H >= 60 && H < 120 THEN
        r = X, g = C, b = 0
    ELSE IF H >= 120 && H < 180 THEN
        r = 0, g = C, b = X
    ELSE IF H >= 180 && H < 240 THEN
        r = 0, g = X, b = C
    ELSE IF H >= 240 && H < 300 THEN
        r = X, g = 0, b = C
    ELSE IF H >= 300 && H < 360 THEN
        r = C, g = 0, b = X
    END IF
    R = (r + m) * 255
    G = (g + m) * 255
    B = (b + m) * 255
    RETURN new Colour(R,G,B)
}

public static Color HSLtoRGB(double h, double s, double l) {
    double c = (1 - Math.Abs((2*l)-1)) * s;
    double x = c * (1-Math.Abs(((h/60)%2)-1));
    double m = l-(c/2);
    double r,g,b;
    if (h < 60) {
        r = c; g = x; b = 0;
    }
    else if (h >= 60 && h < 120) {
        r = x; g = c; b = 0;
    }
    else if (h >= 120 && h < 180) {
        r = 0; g = c; b = x;
    }
    else if (h >= 180 && h < 240) {
        r = 0; g = x; b = c;
    }
    else if (h >= 240 && h < 300) {
        r = x; g = 0; b = c;
    }
    else {
        r = c; g = 0; b = x;
    }
    int R = (int)((r+m) * 255);
    int G = (int)((g+m) * 255);
    int B = (int)((b+m) * 255);

    return Color.FromArgb(R,G,B);
}

```



## Generating Colour Square

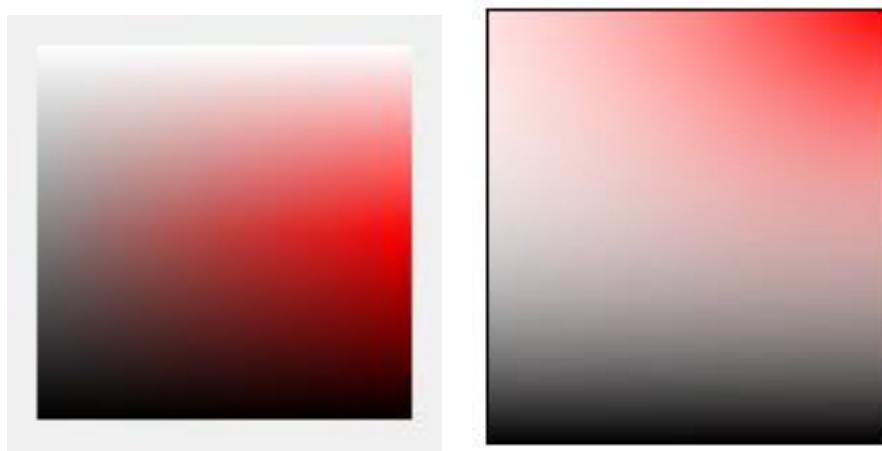
The code to generate the Colour Square has been implemented, in accordance to Algorithm 3.2C:

```
private void UpdateColourSquare() {
    float hue = _startColor.GetHue();
    Bitmap newImage = new Bitmap(100,100);

    // generate each pixel
    for (float x = 0; x < 100; x++) {
        for (float y = 0; y < 100; y++) {
            // uses 1-() in order to flip in y axis
            Color currentColour = SIMPRGB.HSLtoRGB(hue,x/100,1-(y/100));
            newImage.SetPixel((int)x,(int)y,currentColour);
        }
    }

    picColourSquare.Image = newImage;
}
```

However the output square is different to that expected in the design:



This is due to the fact that the analysed program, GIMP, uses HSV colour notation rather than HSL. The calculation for HSV differ slightly, resulting in a different square. HSV cannot be used in SIMP without extra work due to the fact that the internal C# libraries for Color use HSL notation, as the Hue of a colour can be gotten with a pre-written function. This means the program will run faster.

## Other Graphics

Code has been written for the other graphics objects, making the GUI currently look like so:



## 10/11/2019 RGB Picker functionality

### Numerical Inputs

Some simple code for updating the current colour when a numeric input has been implemented. Each one checks the Boolean updating before proceeding, to make sure that the boxes aren't being updated by code:

```
void NumRValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb((byte)numR.Value, _currentColor.G, _currentColor.B);

    Update();
}

void NumGValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb(_currentColor.R, (byte)numG.Value, _currentColor.B);

    Update();
}

void NumBValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb(_currentColor.R, _currentColor.G, (byte)numB.Value);

    Update();
}

private new void Update()
{
    _updating = true;

    UpdateColourSquare();
    UpdateColourStrip();
    UpdateColourDisplays();
    UpdateColourInputs();

    _updating = false;
}
```

This prevents the current colour from being changed during update.

## Colour Strip Input

Gathering input from the Colour Strip is also reasonably simple. As the strip is 100px tall simply multiplying the Y by 100 gets the wanted hue:

```
void PicColourStripClick(object sender, EventArgs e)
{
    MouseEventArgs me = (MouseEventArgs)e;

    float hue = ((float)me.Location.Y)*3.6f;
    _currentColor = _currentColor.SetHue(hue);

    Update();
}
```

## Colour Square Input

The input from the Color Square can also be easily extracted, with some arithmetic on the X and Y of the clicklocation:

```
void PicColourSquareClick(object sender, EventArgs e)
{
    MouseEventArgs me = (MouseEventArgs)e;

    float hue = _currentColor.GetHue();
    float saturation = ((float)me.Location.X)/100;
    float lightness = 1-((float)me.Location.Y)/100;

    _currentColor = SIMPRGB.HSLtoRGB(hue,saturation,lightness);

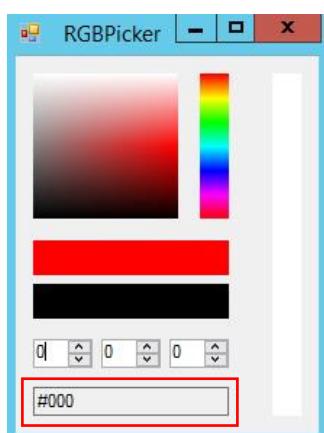
    Update();
}
```

## RGB Display

When the first code for the RGB display was implemented, a problem arose. This was because when the code converted a number to hexadecimal, it did not pad the number, meaning that a hex code could be generated that was less than 6 in width:

```
private void UpdateColourInputs() {
    numR.Value = _currentColor.R;
    numG.Value = _currentColor.G;
    numB.Value = _currentColor.B;

    txtRGB.Text = string.Format("#{0}{1}{2}", _currentColor.R.ToString("X"), _currentColor.G.ToString("X"), _currentColor.B.ToString("X"));
}
```



This can be fixed by adding some extra padding to each part of the hex code:

```
private void UpdateColourInputs() {
    numR.Value = _currentColor.R;
    numG.Value = _currentColor.G;
    numB.Value = _currentColor.B;

    string R = _currentColor.R.ToString("X").PadLeft(2, '0');
    string G = _currentColor.G.ToString("X").PadLeft(2, '0');
    string B = _currentColor.B.ToString("X").PadLeft(2, '0');

    txtRGB.Text = string.Format("#{0}{1}{2}", R, G, B);
}
```

The parts are now padded

## Problem with returning

When originally designing the RGBPicker feature, it was assumed that since a colour was an object, it would be passed by reference. Thus any changes to the colour would reflect in its original function. However this is not true for C#'s colour class, which is passed by value. This means that changing the reference won't work.

## Solution - Delegates

Instead, a **delegate** can be used, which is defined by the caller and will handle changing the colour when necessary:

```
public delegate void ColorCallback(Color newColour);
```

Thus, when a new Save button is clicked, this delegate is called, sending the new colour back to the caller and allowing it to do whatever is needed with the new colour:

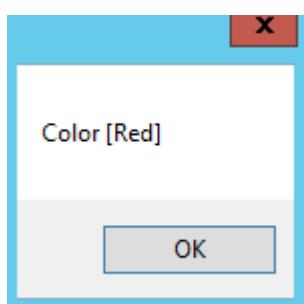
```
void BtnSaveClick(object sender, EventArgs e)
{
    // calls the callback to be handled by caller
    _updateCallback(_currentColor);

    Close();
}
```

Meaning a call can be made like this:

```
RGBPicker newPicker = new RGBPicker(Color.Red, delegate(Color newColour) {
    MessageBox.Show(newColour.ToString());
});
```

The colour is indeed correctly returned after pressing 'Save':



## 13/11/2019 Defining Tools

---

### Storing Tools

The simple brush has been added to the workspace, though only in basic class form. This was done by defining a list, and adding the relevant info for the linetool:

```
// Defines tools
tools.Add(new LineTool("Brush", "Simple Brush", Color.Black));
```

### Displaying Tools

However, it then becomes necessary to display the possible tools on the side of the program. Some simple code can be written to accomplish this:

```
// Adds buttons
int buttonY = 0;
foreach (ITool tool in tools) {
    Button newButton = new Button();
    newButton.Size = new Size(24, 24);
    newButton.Location = new Point(0, buttonY);
    newButton.Name = tool.name;
    newButton.Click += new EventHandler(ToolButtonClick);
    buttonY+=24;

    panTools.Controls.Add(newButton);
}
```

Then when the button is pressed:

```
void ToolButtonClick(object sender, EventArgs e) {
    Button buttonSender = (Button)sender;

    // disables every button except for pressed one
    foreach (Button button in panTools.Controls) {
        button.Enabled = true;
    }
    buttonSender.Enabled = false;

    // selects the tool
    foreach (ITool tool in tools) {
        if (tool.name.Equals(buttonSender.Name)) {
            currentTool = tool;
        }
    }
}
```

This means that `currentTool` will store the current tool being used.

## Hooking Tools

This means that code can be taken out of the current click events, and moved to be encapsulated by the tool:

Before:

```
void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.Black);
        UpdateDisplayBox(true);
    } else if (e.Button == MouseButtons.Right) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.White);
        UpdateDisplayBox(true);
    }
}

void DisplayBoxMouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.Black);
        UpdateDisplayBox(true);
    } else if (e.Button == MouseButtons.Right) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.White);
        UpdateDisplayBox(true);
    }
}
```

After:

```
void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
    currentTool.HandleMouseDown(image.DisplayPointToFilePoint(clickLocation),e.Button);
}

DisplayPoint oldLocation = new DisplayPoint(0,0);
void DisplayBoxMouseMove(object sender, MouseEventArgs e)
{
    //TODO: Finish the clicking stuff
    DisplayPoint newLocation = new DisplayPoint(e.Location.X,e.Location.Y);
    currentTool.HandleMouseMove(image.DisplayPointToFilePoint(oldLocation),image.DisplayPointToFilePoint(newLocation));
    oldLocation = newLocation;
}
```

Which uses a new constructor for `FilePoint`

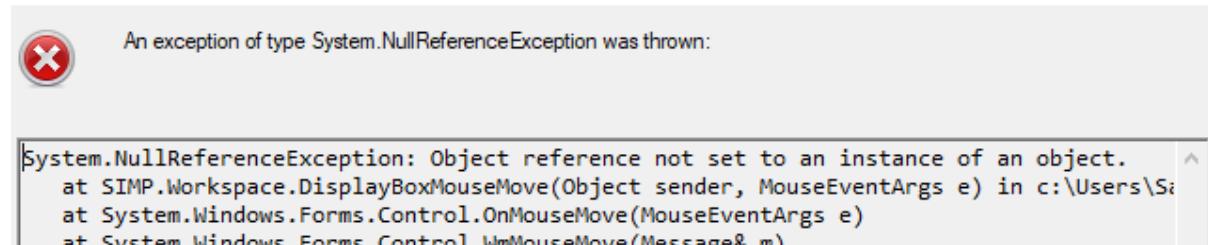
```
public FilePoint(Point p) : this(p.X,p.Y) {
    // no extra implementation
}
```

## Fixing a NullReferenceError

However, a problem emerges with this implementation. At the start of the program `currentTool` is unset and thus null, so when any reference to this before a tool is selected causes a `NullReferenceError` to occur.

Unhandled exception

X



However, to make it so that there is a clean way of showing that there is a blank tool, a `BlankTool` class can be created and statically stored in the encompassing `ITool` class:

```
internal class BlankTool : ITool
{
    internal BlankTool()
    {
        //nothing
    }

    // no implementation
    public override void HandleMouseDown(FilePoint clickLocation, MouseButtons button){}
    public override void HandleMouseUp(FilePoint clickLocation, MouseButtons button){}
    public override void HandleMouseClick(FilePoint clickLocation, MouseButtons button){}
    public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation){}
}
```

In `ITool`:

```
//blanktool
public static ITool BlankTool;

static ITool() {
    BlankTool = new BlankTool();
}
```

And so the Blank Tool can be set like so:

```
currentTool = ITool.BlankTool;
```

Meaning that the crash is avoided.

## 16/11/2019 Implementing LineTool

A simple implementation for the tool can be coded, using a Boolean for whether the brush is down or not:

```
public override void HandleMouseDown(FilePoint clickLocation, MouseButtons button) {
    //TODO: LineTool HandleMouseDown()
    _mouseDown = true;
}

public override void HandleMouseUp(FilePoint clickLocation, MouseButtons button){
    //TODO: LineTool HandleMouseUp()
    _mouseDown = false;
}

public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
    //TODO: LineTool HandleMouseMove()
    Color myColor = Color.Black;
    if (_mouseDown) {
        myWorkspace.image.SetPixel(newLocation,myColor);
        myWorkspace.UpdateDisplayBox(true);
    }
}
```

Which is able to draw. The tool will be iteratively improved on until the tool is complete.

### Implementing Colour Parameter

The first change that can be made is implementing the Colour property into the tool, which uses the GetProperty() method:

```
public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
    //TODO: LineTool HandleMouseMove()
    if (_mouseDown) {
        Color myColor = ((ColorProperty)GetProperty("Color")).value;
        myWorkspace.image.SetPixel(newLocation,myColor);
        myWorkspace.UpdateDisplayBox(true);
    }
}
```

Casts an IProperty to a ColorProperty to get its value

However GetProperty is not implemented, so must be implemented like so:

```
public SIMP.Properties.IProperty GetProperty(string propertyName) {
    //TODO: ITool GetProperty()
    foreach (IProperty property in properties) {
        if (property.name.Equals(propertyName)) {
            return property;
        }
    }
    throw new KeyNotFoundException("Couldn't find property " + propertyName);
}
```

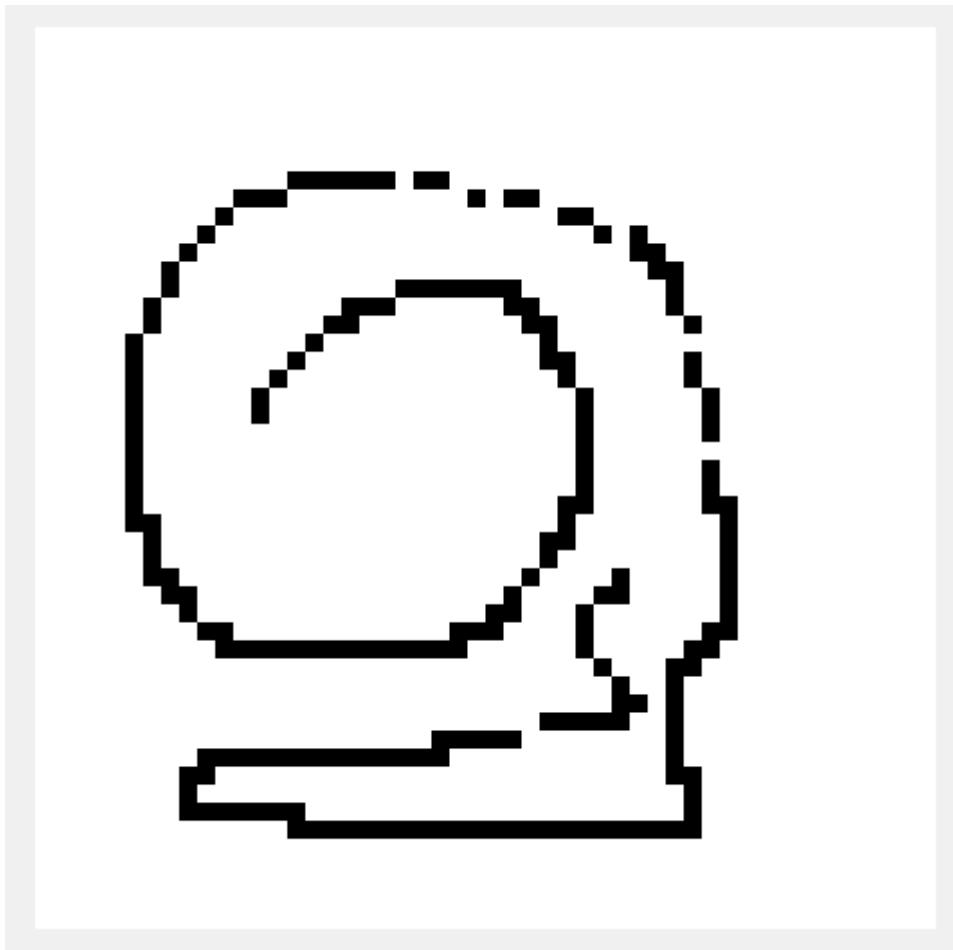
## Connecting Pixels – Bresenhem's

First, a check for whether two pixels is adjacent is implemented. This is because two adjacent pixels don't require Bresenhem's Algorithm, it can just be simply drawn:

```
private bool IsAdjacent(FilePoint point1, FilePoint point2) {
    if (Math.Abs(point1.fileX - point2.fileX) > 1) {
        return false;
    }
    if (Math.Abs(point1.fileY - point2.fileY) > 1) {
        return false;
    }
    return true;
}
```

Then, Bresenhem's algorithm can be implemented in accordance to Algorithm 3.6A

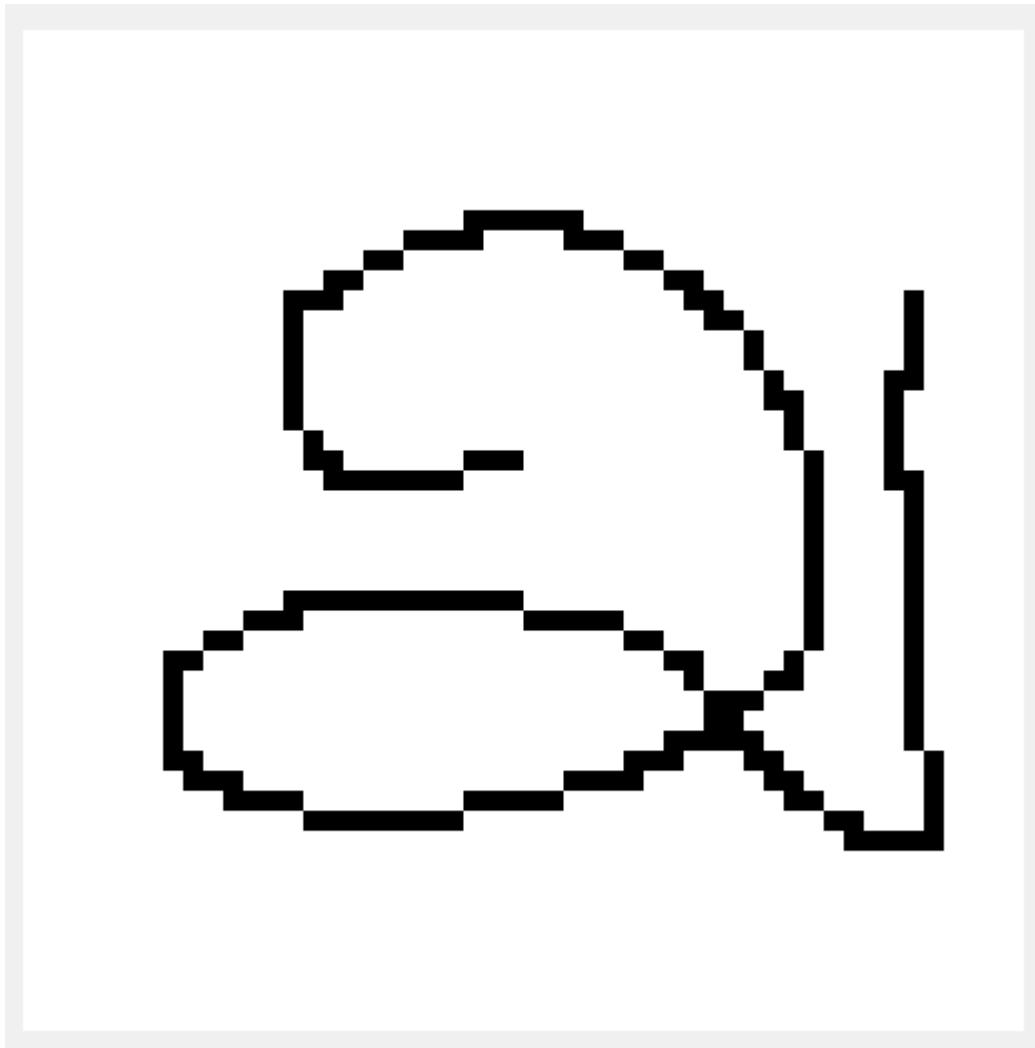
However, it was discovered that there was a problem with the implemented algorithm, it sometimes left a gap:



The problem turned out to be that the last pixel is not set by Bresenhem's Algorithm, so it needs to be set manually, with an extra line of code:

```
myWorkspace.image.SetPixel(newLocation,myColor);  
DrawLine(oldLocation,newLocation);
```

This makes sure all lines are now connected:



## 17/11/2019 Generating Brushes

In order to store the necessary points for the brush, Algorithm 3.4A has been implemented, making a list for this purpose:

```
private List<FilePoint> currentBrush;
```

Then, to generate the brush's points, Algorithm 3.4B has been implemented, which checks all candidate points:

```
FOR x = brushSize*-1 TO brushSize
    FOR y = brushSize*-1 TO brushSize
        IF IsPointInCircle(x+0.5, y+0.5) THEN
            currentBrush.Add(x,y)
        END IF
    NEXT
NEXT
```

```
private void GenerateBrush() {
    this.currentBrush = new List<FilePoint>();

    int radius = 10;

    for (float x = radius * -1; x < radius; x++) {
        for (float y = radius * -1; y < radius; y++) {
            if (IsPointInCircle(x + 0.5f, y + 0.5f)) {
                currentBrush.Add(new FilePoint((int)x,(int)y));
            }
        }
    }
}
```

Then the relevant test has been implemented, again according to Algorithm 3.4B:

```
IsPointInCircle(x,y) {
    IF ((x*x)+(y*y) <= (brushSize*brushSize)) THEN
        RETURN true
    ELSE
        RETURN false
    END IF
}
```

```
private bool IsPointInCircle(float x, float y, float radius) {
    // returns whether this condition is true or false
    return ((x*x)+(y*y)) <= (radius * radius);
}
```

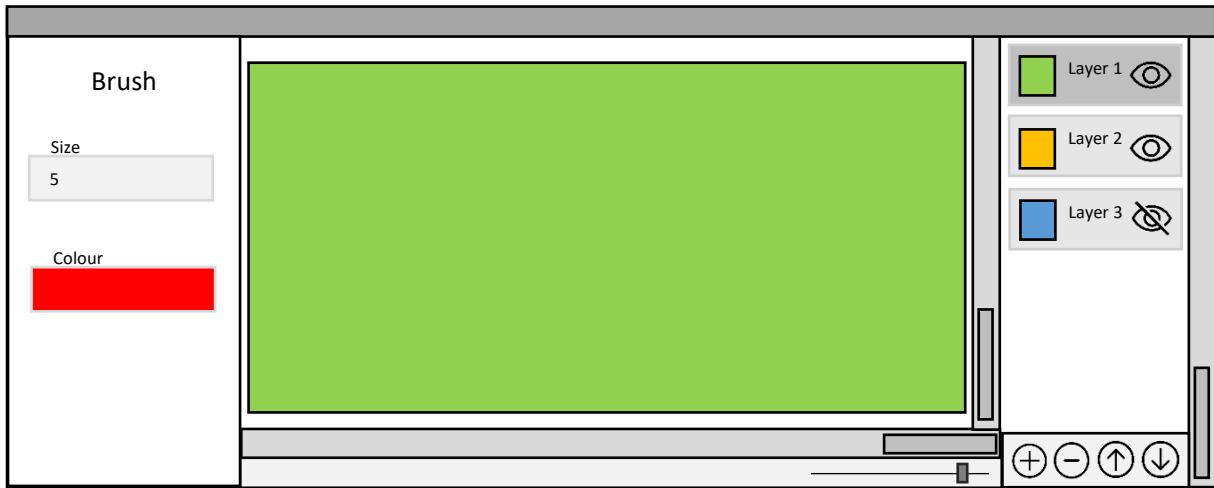
Finally, with a small piece of code to set the relevant pixels, thick brushes can now be used:



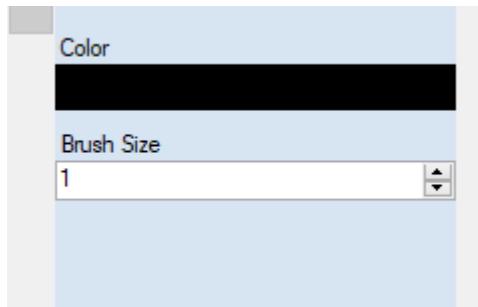
## 18/11/2019 Implementing Properties

All the properties thus far have been drawing from constants. It is now necessary that the user can edit these properties, so a new menu must be designed.

This menu should simply be a list of properties, placed onto the left hand side of the screen. As a mockup:



Then some code was implemented to generate the needed controls from a list of properties:



```

public void ShowTool() {
    leftPadding = SimpConstants.WORKSPACE_LEFT_PADDING + 200;
    panToolProperties.Visible = true;
    panToolProperties.Controls.Clear();

    Label lblToolname = new Label();
    lblToolName.Location = new Point(6, 0);
    lblToolName.Size = new Size(191, 23);
    lblToolName.Text = currentTool.name;
    panToolProperties.Controls.Add(lblToolname);

    int currentY = lblToolName.Height;
    int width = panToolProperties.Width;
    foreach (IProperty property in currentTool.properties) {
        Label newLabel = new Label();
        newLabel.Size = new Size(width, SimpConstants.PROPERTY_LABEL_HEIGHT);
        newLabel.Location = new Point(0, currentY);
        newLabel.Text = property.name;
        panToolProperties.Controls.Add(newLabel);
        currentY += SimpConstants.PROPERTY_LABEL_HEIGHT;

        currentY += SimpConstants.PROPERTY_GAP_HEIGHT;

        if (property is ColorProperty) {
            ColorProperty colorProperty = (ColorProperty)property;
            PictureBox newPicture = new PictureBox();
            newPicture.Size = new Size(width, SimpConstants.PROPERTY_FIELD_HEIGHT);
            newPicture.Location = new Point(0, currentY);
            newPicture.BackColor = colorProperty.value;
            newPicture.Click += colorProperty.onInteract;
            panToolProperties.Controls.Add(newPicture);
        } else if (property is NumericalProperty) {
            NumericUpDown newNum = new NumericUpDown();
            NumericalProperty numericalProperty = (NumericalProperty)property;
            newNum.Size = new Size(width, SimpConstants.PROPERTY_FIELD_HEIGHT);
            newNum.Location = new Point(0, currentY);
            newNum.Value = numericalProperty.min;
            newNum.Minimum = numericalProperty.min;
            newNum.Maximum = numericalProperty.max;
            newNum.Value = numericalProperty.value;
            newNum.ValueChanged += numericalProperty.onInteract;
            panToolProperties.Controls.Add(newNum);
        }
        currentY += SimpConstants.PROPERTY_FIELD_HEIGHT;
        currentY += SimpConstants.PROPERTY_SPACER_HEIGHT;
    }

    CalculateDimensions();
    UpdateDisplayBox(true);
}

```

After this, the value updating could be handled by a delegate model, where each Property has its own OnInteract delegate that is called whenever their respective field has been interacted with:

```

public abstract class IProperty
{
    public string name;
    public EventHandler onInteract;
    public Workspace myWorkspace;
}

```

onInteract is implemented by each property.

This now means that colour images can be created with SIMP, and multiple thicknesses of line:



## 19/11/2019 Stakeholder feedback

Today I presented the current version of SIMP for my stakeholders to use. I received the following feedback:

### Alex H

- Creating a brush of size 100 and then pasting on the image causes a large freeze
- Clicking the top of the hex selector fixes the hue at red until colour is changed
- Inputting hex code
- Better representation of how thick the brush will be
- Tutorial
- No copy paste
- No layers
- Brush starts unselected
- No copy and paste
- Creating new image should be inside workspace
- Must start drawing on canvas
- No recent colours
- No rubber
- No single pixel brush

### Alex G

- Colour picker improvements:
  - Holding button down should update in real-time
  - Picker area should have the brightest hue at top
  - Releasing button off grid doesn't update colour to max
- Side panel can't be hidden
- Zoom should display numerically how much zoom
- Holding mouse off canvas and then moving on doesn't work
- Resizing window to smallest when having properties open causes a crash

### Dheshpreet

- Needs a single pixel brush
- No eyedropper tool
- No layers
- No undo / redo
- Slow response times

### Categorizing Feedback

Some elements brought up by the stakeholder feedback are already planned or designed, such as copy & paste, recent colours, rubber, undo / redo & layers. The following can be fixed reasonably easily and so will be acted upon at this time.

Issue	ID	Requested by
<i>Using a brush of size 100 causes large slowdown</i>	1	Alex H
<i>Clicking the top of the hue selector box resets colour</i>	2	Alex H
<i>Starting program with brush selected</i>	3	Alex H
<i>Cannot start stroke off canvas</i>	4	Alex H, Alex G
<i>No recent colours</i>	5	Alex H
<i>No single pixel brush</i>	6	Alex H, Dheshpreet
<i>Holding button on colour picker should update in real-time</i>	7	Alex G
<i>Colour picker should be arranged with lightest hue on top</i>	8	Alex G
<i>Moving cursor off colour picker does not update to max</i>	9	Alex G
<i>Crash when making window with open properties too small</i>	10	Alex G
<i>Performance issues</i>	11	Dheshpreet

These 14 issues will be addressed before further progress is made.

## Performance issues with large brushes

The performance issue can be tracked to the following excerpt of code:

```
public void SetPixel(FilePoint filePoint, Color colour) {
    try {
        pixels[filePoint.fileX,filePoint.fileY] = new SolidBrush(colour);
    } catch (IndexOutOfRangeException) {
        // ignore request if it tried to set out of bounds
        // TODO: some sort of logging system to warn about this
    }
}
```

Under normal condition, only the valid section is used. However when big brushes are used the majority of the points are invalid, meaning many errors are thrown. Computationally error handling is very expensive, and so many expensive throws will slow the program massively.

To fix this, the brush checked code can be edited to check whether the pixel is in-bounds before checking it:

```
foreach (FilePoint brushPoint in currentBrush) {
    stampPoint = new FilePoint(x + brushPoint.fileX, y + brushPoint.fileY);
    if (stampPoint.fileX < 0 ||
        stampPoint.fileX >= myWorkspace.image.fileWidth ||
        stampPoint.fileY < 0 ||
        stampPoint.fileY >= myWorkspace.image.fileHeight) {
        continue;
    }
    myWorkspace.image.SetPixel(stampPoint,myColor);
}
```

## Performance issues displaying image

Currently when making brush manipulations, it is very slow to update. This is due to the fact that when a brush makes a change, the entire image is updated. This is very inefficient, so can be changed to only update changed pixels (if needed).

To do this, a list of changed pixels was introduced, which is updated whenever a change is made to the pixels:

```
public void SetPixel(FilePoint filePoint, Color colour) {
    try {
        pixels[filePoint.fileX,filePoint.fileY] = new SolidBrush(colour);
        changedPixels.Add(new FilePoint(filePoint));
    } catch (IndexOutOfRangeException) {
        // ignore request if it tried to set out of bounds
        // TODO: some sort of logging system to warn about this
    }
}
```

Then, when a brush makes a change to the image, the following code is called which only updates the changed pixels:

```
private void DrawChangedImage(Graphics GFX) {
    DisplayPoint displayChangedLoc;
    foreach (FilePoint changedPixel in changedPixels) {
        // finds this pixels location
        displayChangedLoc = FilePointToDisplayPoint(changedPixel);
        GFX.FillRectangle(pixels[changedPixel.fileX,changedPixel.fileY],
                          displayChangedLoc.displayX,displayChangedLoc.displayY,
                          zoomSettings.zoom,zoomSettings.zoom);
    }
    changedPixels.Clear();
}
```

This significantly improves performance on larger images.

## Using HSV rather than HSL

The reason why the colour square didn't look right to the stakeholders was because it used the HSL colour notation, where they are used to seeing HSV.

Changing the conversion from HSLtoRGB to HSVtoRGB is simple, and only requires a few line changes,

```
double c = s * l;
double x = c * (1-Math.Abs(((h/60)%2)-1));
double m = l - c;
double c = (1 - Math.Abs((2*l)-1)) * s;
double x = c * (1-Math.Abs(((h/60)%2)-1));
double m = l-(c/2);
```

However, it means that an algorithm for converting from RGB to HSL will be needed, as C# does not provide this.

This means an algorithm for the conversion will need to be implemented:

```
public static void RGBtoHSV(double r, double g, double b) {
    double R = r/255;
    double G = g/255;
    double B = b/255;

    double Cmax = Max(R,G,B);
    double Cmin = Min(R,G,B);

    double Δ = Cmax - Cmin;

    double h,s,v;
    if (Δ == 0) {
        h = 0;
    } else if (Cmax == R) {
        h = 60 * ((G-B)/Δ)%6;
    } else if (Cmax == G) {
        h = 60 * ((B-R)/Δ)+2;
    } else if (Cmax == B) {
        h = 60 * ((B-R)/Δ)+4;
    }

    if (Cmax == 0) {
        s = 0;
    } else {
        s = Δ/Cmax;
    }

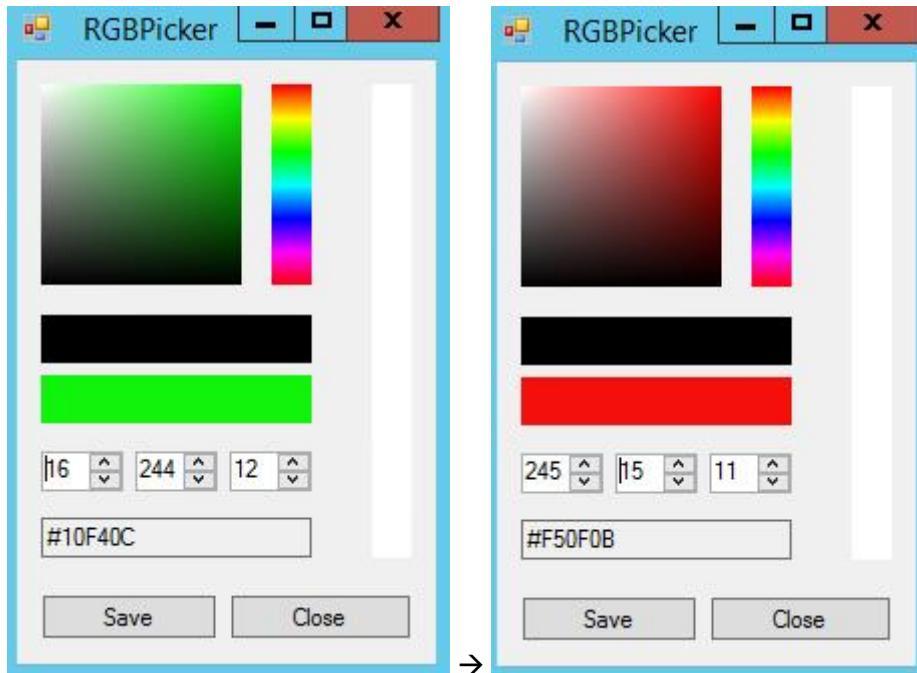
    v = Cmax;
}
```

Currently however, there is nothing to return, as there is no class for a HSV colour. This means one must be created:

```
struct HSVColor {
    public double H;
    public double S;
    public double V;

    public HSVColor(double H, double S, double V) {
        this.H = H; this.S = S; this.V = V;
    }
}
```

However when testing the RGB picker, a problem seems to emerge with the conversion, as it seems to incorrectly convert from RGB to HSV:



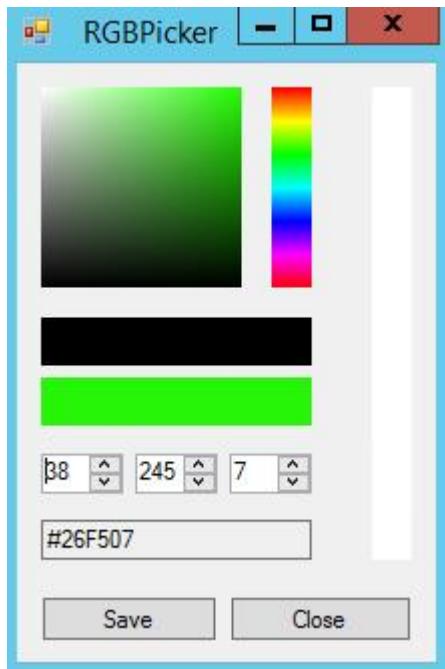
The problem was isolated to some missing brackets when calculating the hue, so could be corrected:

```

if (Δ == 0) {
    h = 0;
} else if (Cmax == R) {
    h = 60 * ((G-B)/Δ)%6;
} else if (Cmax == G) {
    h = 60 * (((B-R)/Δ)+2);
} else if (Cmax == B) {
    h = 60 * (((B-R)/Δ)+4;
}
→
if (Δ == 0) {
    h = 0;
} else if (Cmax == R) {
    h = 60 * (((G-B)/Δ)%6);
} else if (Cmax == G) {
    h = 60 * (((B-R)/Δ)+2);
} else if (Cmax == B) {
    h = 60 * (((B-R)/Δ)+4);
}

```

However the numeric input still appears to have some small errors, as in some cases they are unresponsive. For example in this scenario increasing the 'G' does nothing:



This is due to the fact that upon changing colour, all 3 values are recalculated. As the calculation is not reversible this leads to small inaccuracies that can lead to rounding errors.

The solution for this is to introduce a Boolean on the update code on whether the RGB nums are recalculated. This stops them from being recalculated when they are being edited:

```
private void UpdateColourInputs(bool updateNums) {
    Color displayColor = _currentColor.ToColor();

    if (updateNums) {
        numR.Value = displayColor.R;
        numG.Value = displayColor.G;
        numB.Value = displayColor.B;
    }

    string R = ((int)numR.Value).ToString("X").PadLeft(2, '0');
    string G = ((int)numG.Value).ToString("X").PadLeft(2, '0');
    string B = ((int)numB.Value).ToString("X").PadLeft(2, '0');

    txtRGB.Text = string.Format("#{0}{1}{2}", R, G, B);
}
```

## 20/11/2019 Further improvements

---

### Single Pixel Brush

A single pixel brush has been highly requested, due to the fact that all brushes are stored by radius, and the minimum radius is 1, the smallest possible diameter is 2.

In order to make it, a new class was implemented, SinglePixelLineTool. This inherits from LineTool but continues the constructor to replace the properties with ones to make a line of single pixel width.

However, the only way to make a single width line without redesigning the system is to have a radius of 0.5 (thus diameter 1).

This then introduces a problem – there is no property that accepts decimal input. Thus a new class must be made to have decimal input:

```
public class DecimalProperty : IProperty
{
    public decimal value;
    public decimal min;
    public decimal max;

    public DecimalProperty(string name, decimal value, decimal min, decimal max, Workspace myWorkspace)
    {
        this.name = name;
        this.value = value;
        this.min = min;
        this.max = max;
        this.myWorkspace = myWorkspace;

        this.onInteract = delegate(Object sender, EventArgs e) {
            this.value = (decimal)((NumericUpDown)sender).Value;
            myWorkspace.ShowTool();
        };
    }
}
```

Then this can be added to the Single Pixel Brush:

```
this.properties.Clear(); // removes all previous properties
this.properties.Add(new ColorProperty("Color", color, myWorkspace));
this.properties.Add(new DecimalProperty("Brush Size", 0.5M, 0.5M, 0.5M, myWorkspace));
```

However, when adding the brush, an error is introduced as the brush now contains a DecimalProperty rather than a NumericalProperty, so this case fails:

```
this.currentBrush = new List<FilePoint>();

float radius = (float)((NumericalProperty)GetProperty("Brush Size")).value;
```

The solution to this is to enforce that all properties stores the value as a public object:

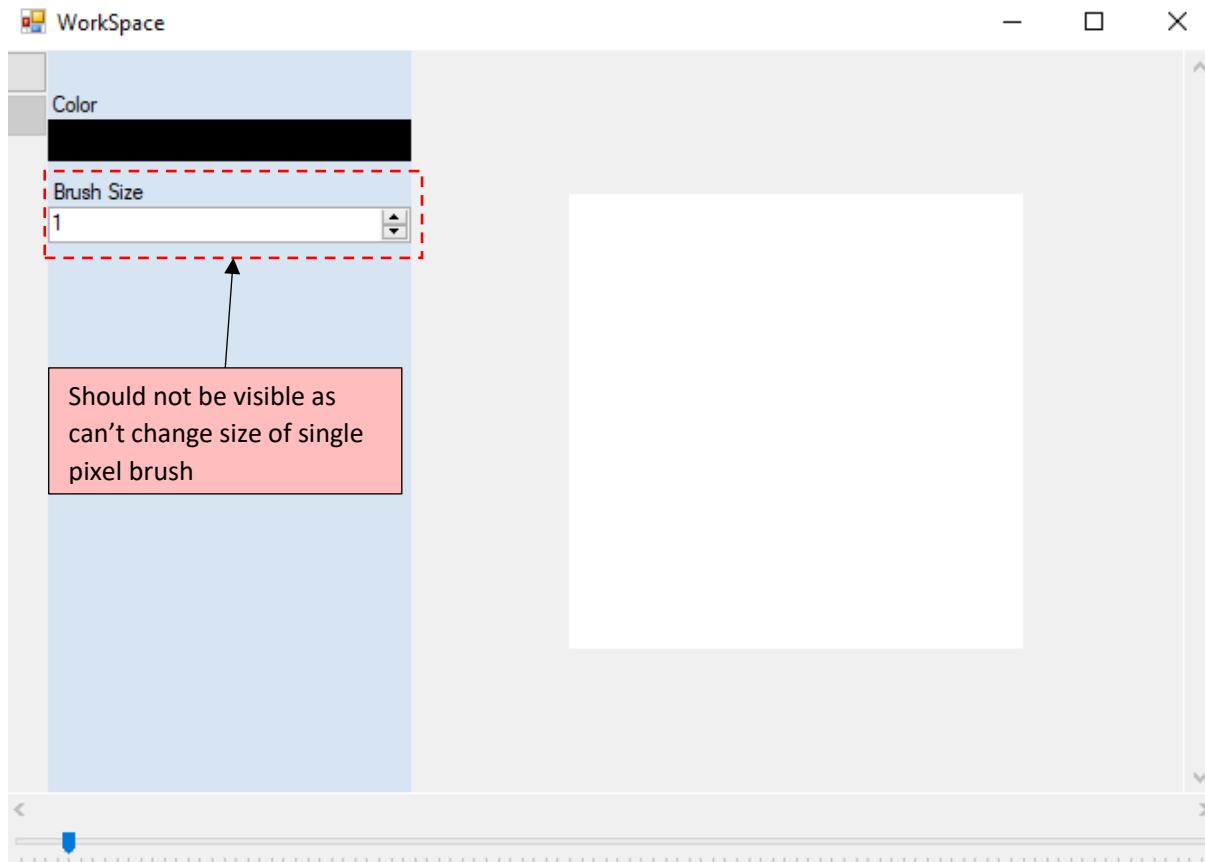
```
public abstract class IProperty
{
    public string name;
    public object value;
    public EventHandler onInteract;
    public Workspace myWorkspace;
}
```

Thus this object can be converted to a float when needed by the radius:

```
float radius = Convert.ToSingle(GetProperty("Brush Size").value);
```

## Property properties

However, there is still a problem, as the brush size is still being stored as a property, thus will show up on the properties pane:

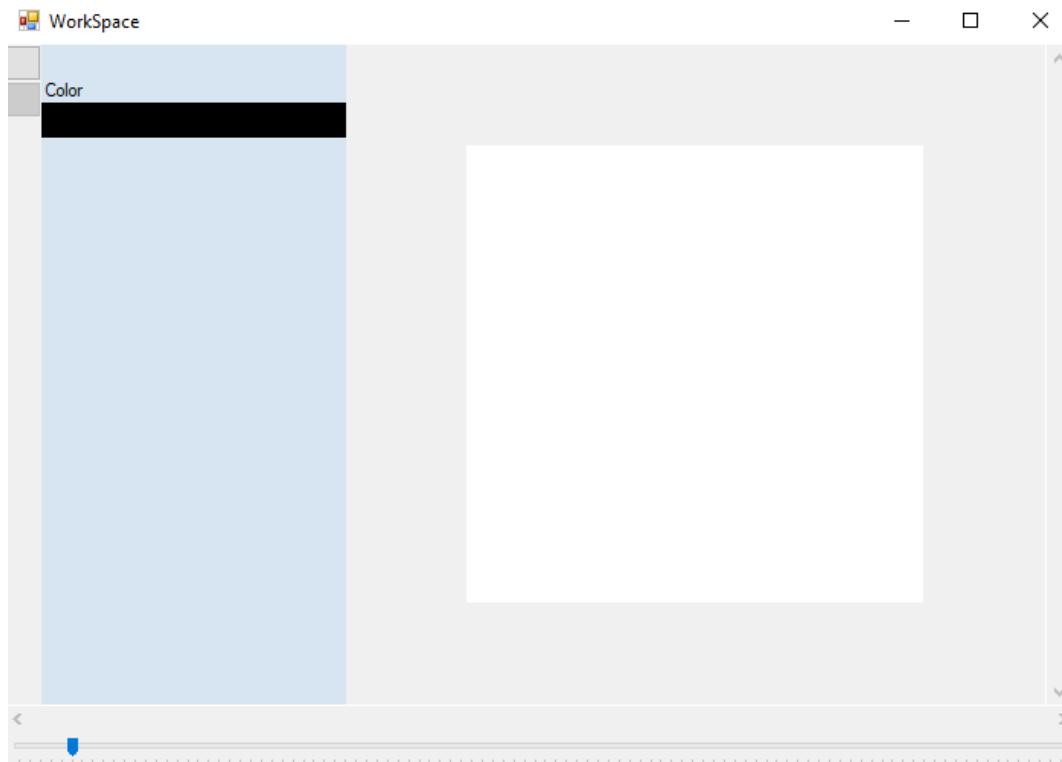


In order to do this, an enum has been introduced, to help organise three possible sorts of property, normal (visible), readonly (cannot be edited) or hidden (cannot be seen or interacted with)

```
public enum PropertyType {
    Normal,
    ReadOnly,
    Hidden
}
```

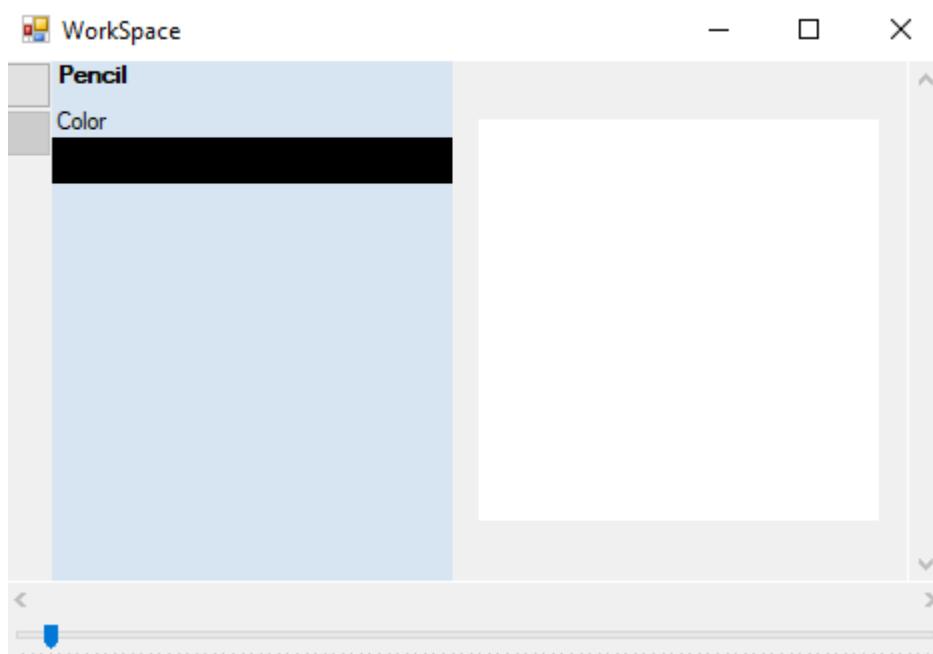
Then, with a check on the display code, hidden properties will not be shown:

```
// doesn't show invisibles
if (property.propertyType == PropertyType.Hidden) {
    continue;
}
```



### Tool title visibility fix

The title of the current tool has also been invisible, though this was missed by the stakeholders. The error stemmed from a missing capital, which has now been fixed:



## Changing starting tool

Changing the starting tool simply requires the program on startup to set the tool to the first items in the tool list:

```
currentTool = tools[0];
ShowTool();
```

So the program now starts with the brush selected.

## Fixing crash when resizing window

Currently there is an issue where changing the form size to the size of the properties bar causes a crash. This is because the window is smaller than the available space to put an image in, so the size is interpreted as zero, and is invalid for an image:

```
Bitmap newImage = new Bitmap(width,height);
Graphics GFX = Graphics.FromImage(newImage);
GFX.Clear(SystemColors.Control);
```

To fix this, a minimum size can be enforced constantly whenever the values are changed:

```
private int _leftPadding, _rightPadding, _topPadding, _bottomPadding;

public int leftPadding {
    get {
        return _leftPadding;
    }
    set {
        _leftPadding = value;
        SetMinimumSize();
    }
}
public int rightPadding {
    get {
        return _rightPadding;
    }
    set {
        _rightPadding = value;
        SetMinimumSize();
    }
}
public int topPadding {
    get {
        return _topPadding;
    }
    set {
        _topPadding = value;
        SetMinimumSize();
    }
}
public int bottomPadding {
    get {
        return _bottomPadding;
    }
    set {
        _bottomPadding = value;
        SetMinimumSize();
    }
}

private void SetMinimumSize() {
    this.MinimumSize = new Size(_leftPadding + _rightPadding + SimpConstants.WINDOWS_RIGHT_BAR_WIDTH + SimpConstants.WINDOWS_LEFT_BAR_WIDTH,
                               _topPadding + _bottomPadding + SimpConstants.WINDOWS_TOP_BAR_HEIGHT + SimpConstants.WINDOWS_BOTTOM_BAR_HEIGHT);
}
```

21/11/2019 Further improvements

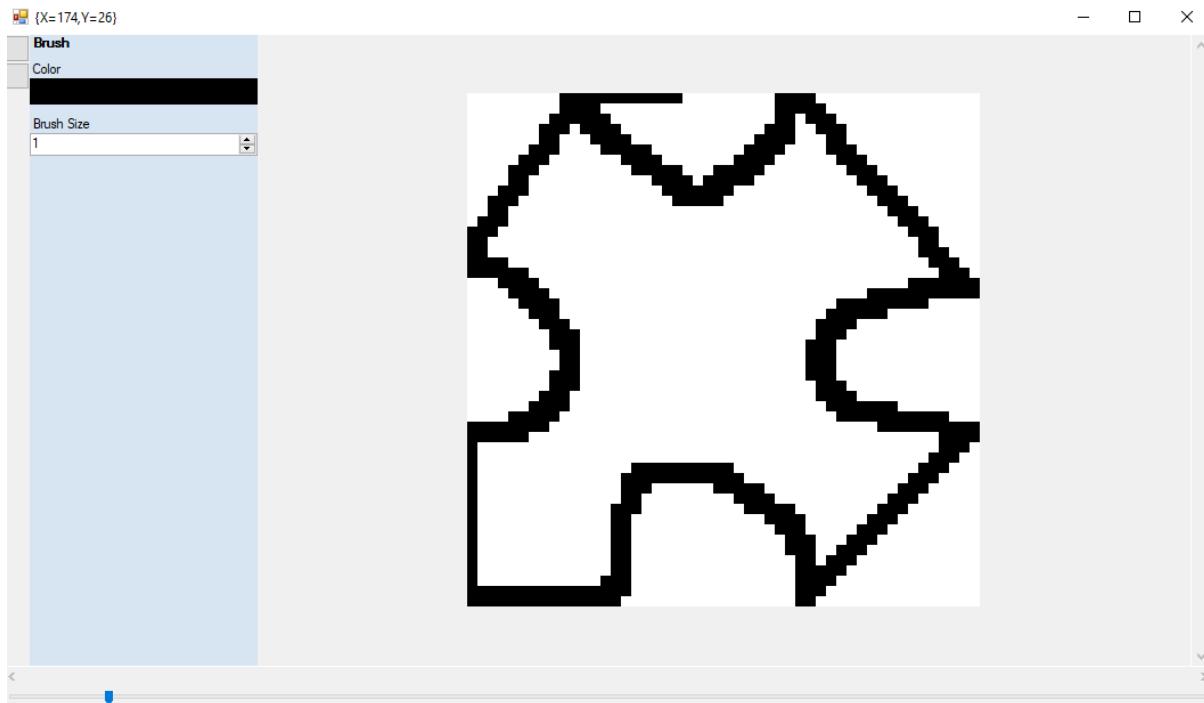
## Starting a stroke off of the workspace

Allowing the user to start a stroke off of the workspace firstly needs a way for the code to recognise that the mouse has been clicked out of bounds, with no attached location. This can be done by checking whether the inputted location is null:

```
public override void HandleMouseDown(FilePoint clickLocation, MouseButtons button) {
    if (clickLocation != null) {
        GenerateBrush();
        StampBrush(clickLocation.fileX,clickLocation.fileY);
        myWorkspace.UpdateDisplayBox(true, false);
    }
    _mouseDown = true;
}
```

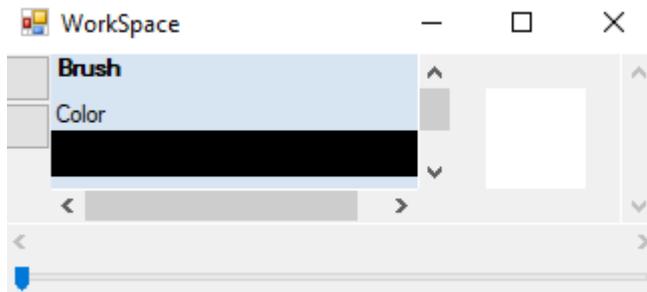
Then code can be called to override the normal movement when a movement is made on the workspace form:

However, after implementing this, a problem arose where when leaving the canvas and coming back on, a line would be drawn from the last location on the canvas to location or re-entry. For example:



In order to fix this, an extra line to update the oldLocation constantly makes the program forget about where the line last was on the canvas:

However this still has the same error, as this is now exactly the smallest image size. Adding a further constant now means that the window cannot be made too small. The smallest potential size is now:



So this fixes the error.

## Fixing Algorithmic Error

When designing a system for keeping a point inside of a picture box, an error was appearing where under some circumstances an out of bounds location was passed (even though there were checks in all 4 dimensions). The error turned out to be:

```
void PicColourSquareMouseMove(object sender, MouseEventArgs e)
{
    if (e.Button != MouseButtons.None) {
        Point newPoint = e.Location;
        if (e.Location.X < 0) {
            newPoint = new Point(0,e.Location.Y); // The X coord is fixed here...
        }
        if (e.Location.X >= picColourSquare.Width) {
            newPoint = new Point(picColourSquare.Width,e.Location.Y);
        }
        if (e.Location.Y < 0) {
            newPoint = new Point(e.Location.X,0); // However here, the potentially
                                                // invalid X coord is used
        }
        if (e.Location.Y >= picColourSquare.Height) {
            newPoint = new Point(e.Location.X,picColourSquare.Height);
        }
        PicColourSquareClick(sender,new MouseEventArgs(e.Button,e.Clicks,newPoint.X,newPoint.Y,e.Delta));
    }
}
```

To fix this, the updated X is used in the fixing of the Y coord.

22/11/2019 Undo

### Implementing image functions

The code for implementing undo in a brush has been implemented, however it has an issue. The system needs to make sure that no duplicate points are added, however the system does not quite work:

```
if (!setPixels.ContainsKey(stampPoint)) { // if this pixel is not yet in the history
    setPixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
    if (!strokePixels.ContainsKey(stampPoint)) {
        MessageBox.Show(stampPoint.ToString())
        strokePixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
    }
}
```

The reason being that because stampPoint (a FilePoint) is an object, its **hash** will be compared to any others in the list, not its location. This means that any equal points with a different hash can still be added to the list.

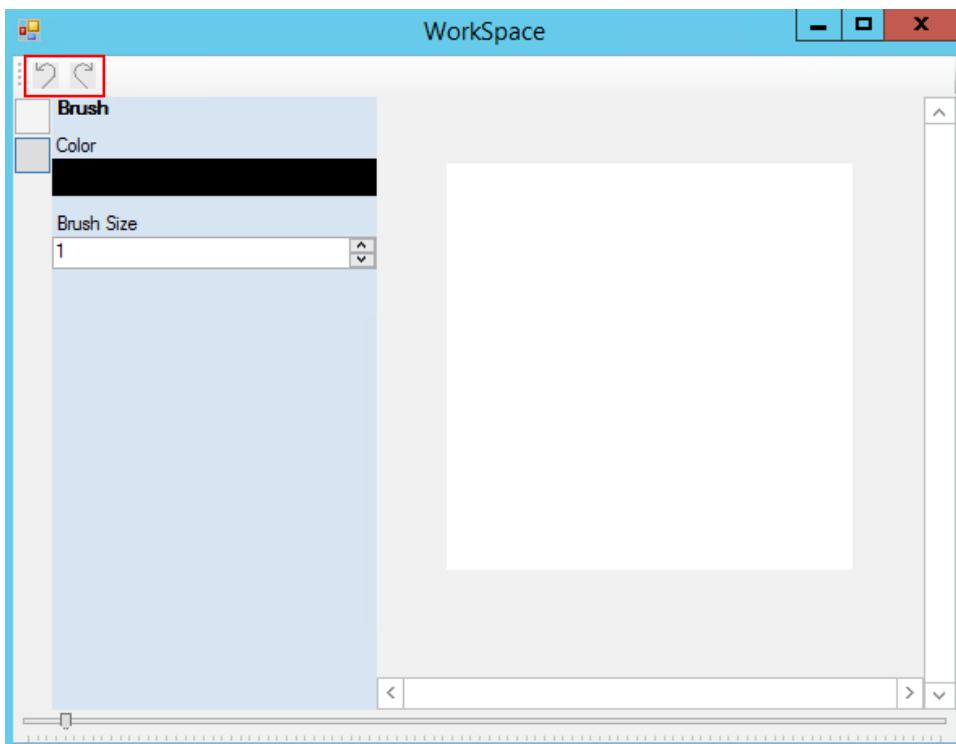
To fix this, a list of primitive strings can be used instead to check whether an item is in the list:

```
if (!setHashes.Contains(stampPoint.ToString())) { // if this pixel is not yet in the history
    setPixels.Add(stampPoint, currentColor);
    setHashes.Add(stampPoint.ToString());
    if (!strokeHashes.Contains(stampPoint.ToString())) {
        strokePixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
        strokeHashes.Add(stampPoint.ToString());
    }
}
```

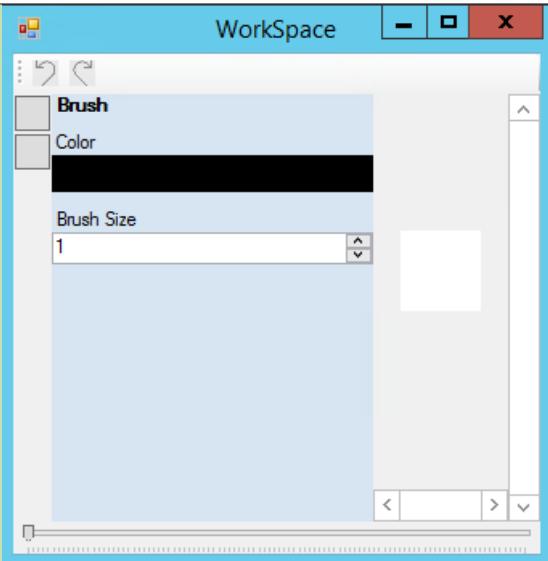
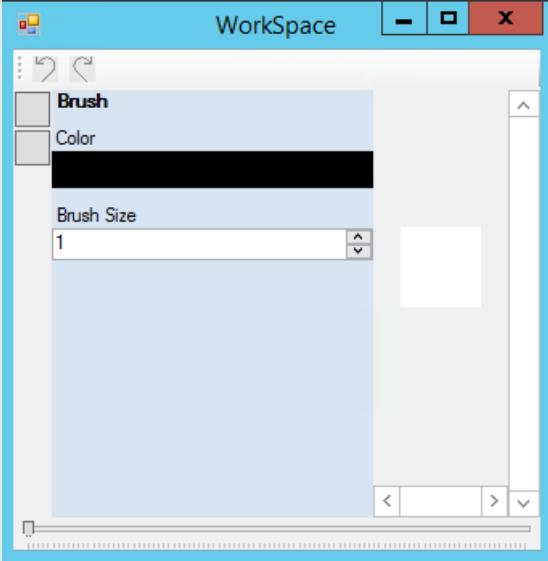
Now the 'hash' (a string representation) will be compared instead

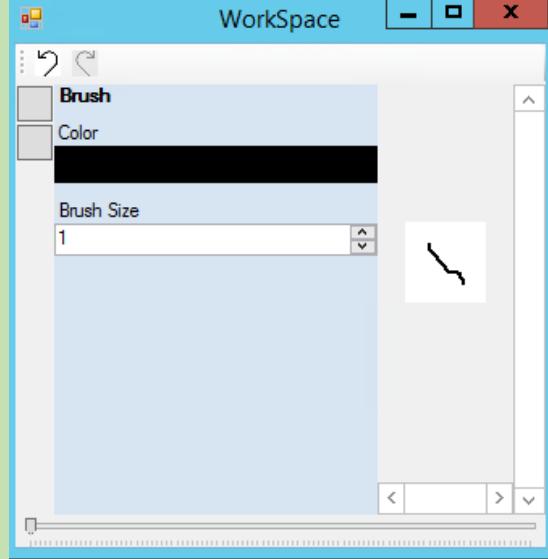
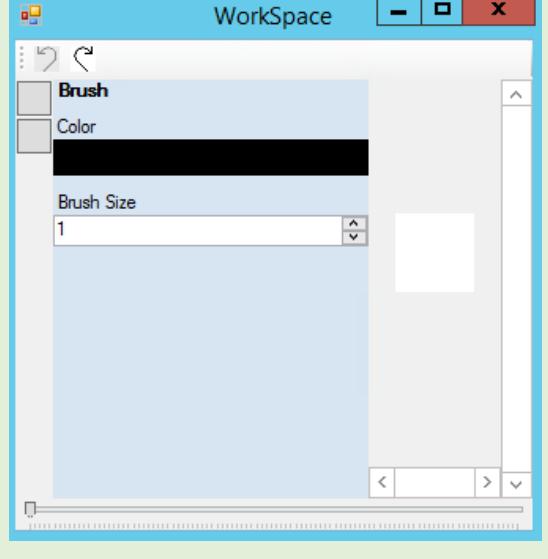
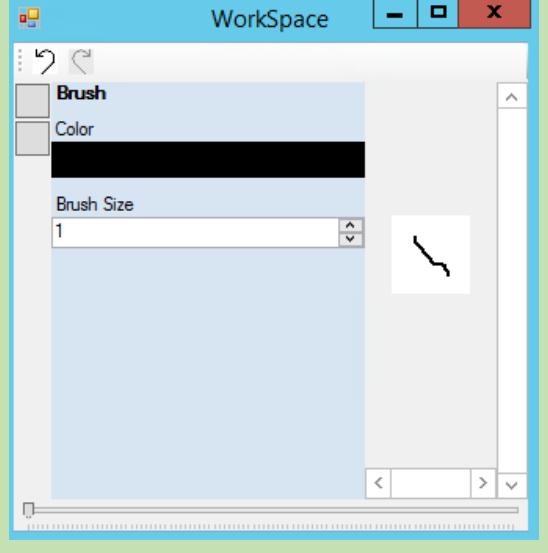
Now, the literal string will be compared, and this is much more effective.

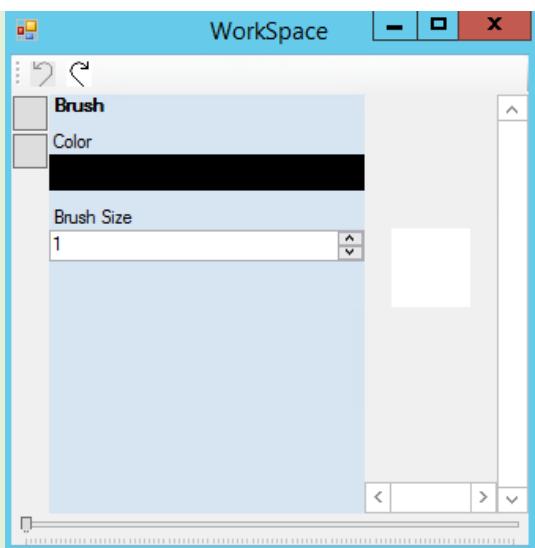
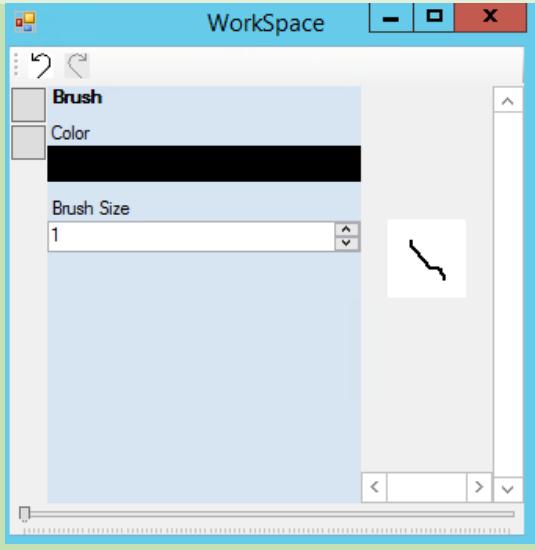
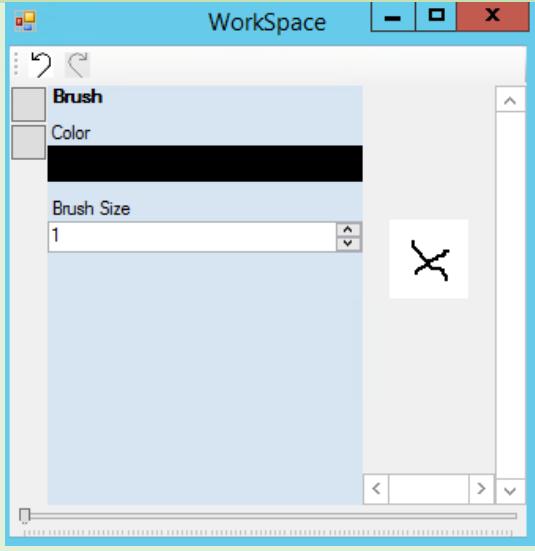
After adding some undo and redo buttons, it is now time to do the unit test:

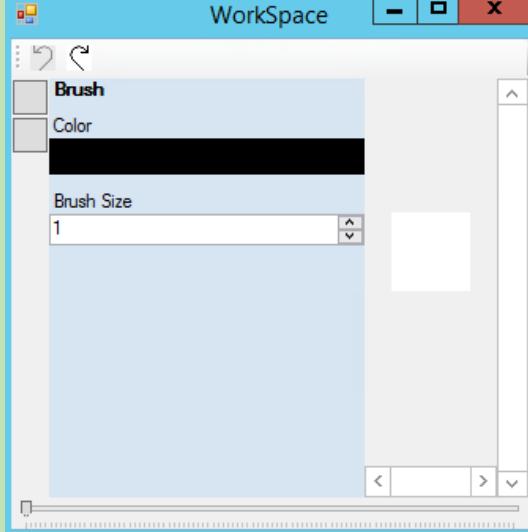
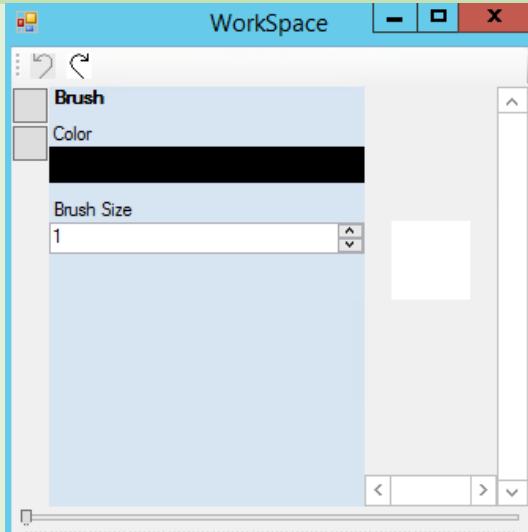
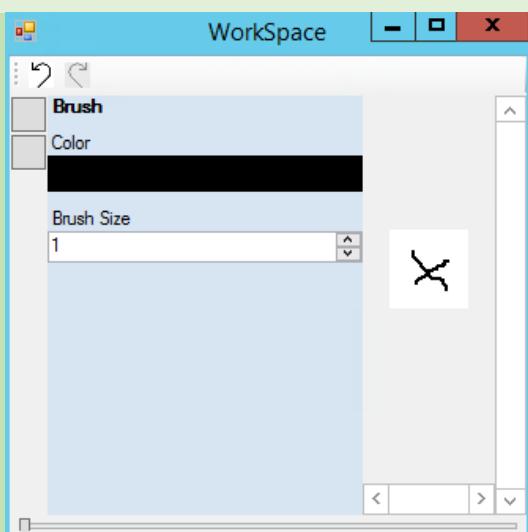


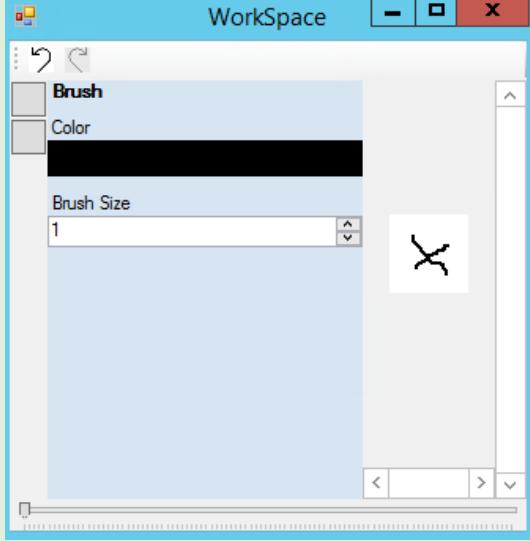
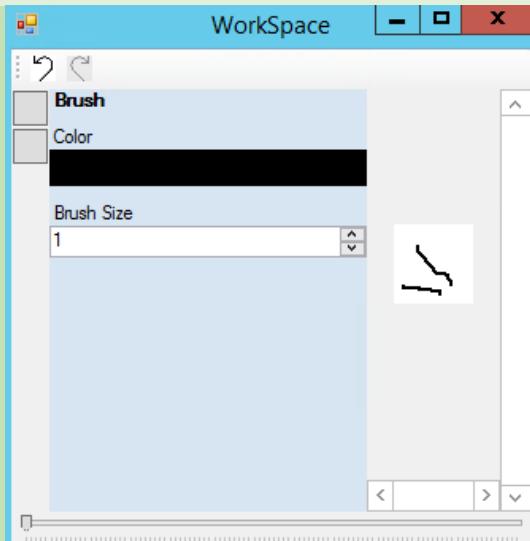
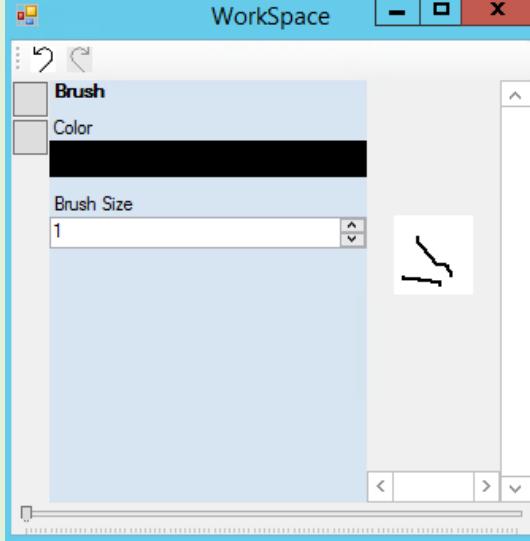
## 23/11/2019 Undo Unit Test

Test ID	Expected Result	Actual Result	Comment
Undo an action (on default image)	1	Cannot be done	
Redo an action (on default image)	2	Cannot be done	

<i>Perform an action</i>	3 Action is performed onto image	 A screenshot of a digital drawing application titled 'WorkSpace'. The interface includes a toolbar with icons for brush, selection, and other tools. A color palette shows 'Brush' and 'Color' swatches. A 'Brush Size' slider is set to 1. On the right, a preview window shows a small black brush stroke. The main canvas area is currently empty.	The action can still be performed and the undo button becomes enabled
<i>Undo the action</i>	4 Action is undone	 A screenshot of a digital drawing application titled 'WorkSpace'. The interface includes a toolbar with icons for brush, selection, and other tools. A color palette shows 'Brush' and 'Color' swatches. A 'Brush Size' slider is set to 1. On the right, a preview window shows a small white square. The main canvas area is empty.	The action is reverted and the redo button is now enabled.
<i>Redo the action</i>	5 Action is redone	 A screenshot of a digital drawing application titled 'WorkSpace'. The interface includes a toolbar with icons for brush, selection, and other tools. A color palette shows 'Brush' and 'Color' swatches. A 'Brush Size' slider is set to 1. On the right, a preview window shows a small black brush stroke. The main canvas area is empty.	The action is redone and the undo button is now enabled

<i>Undo the action</i>	6	Action is undone		The action is undone again
<i>Redo the action</i>	7	Action is redone		The action is redone again
<i>Perform another action</i>	8	Action is done		A new action is performed on top of the old one

<i>Undo both actions</i>	9	Both actions redone		Both actions can be undone by pressing undo twice
<i>Undo an action</i>	10	Cannot be done		Undo is currently disabled after undoing the first two actions so cannot be pressed
<i>Redo both actions</i>	11	Both actions redone		Both actions are put back onto the image

<i>Redo an action</i>	12 Cannot be done		The redo button is disabled after redoing twice
<i>Undo an action, and preform a new action</i>	13 New action is performed on top of old action		The action is preformed successfully
<i>Redo an action</i>	14 Cannot be done		The redo button is disabled as there is now nothing to redo

## 24/11/2019 Layers

### Implementation

The Layer class has been implemented in accordance to Algorithms 3.9, 3.10 and 3.11. The algorithm for drawing a pixel was slightly edited:

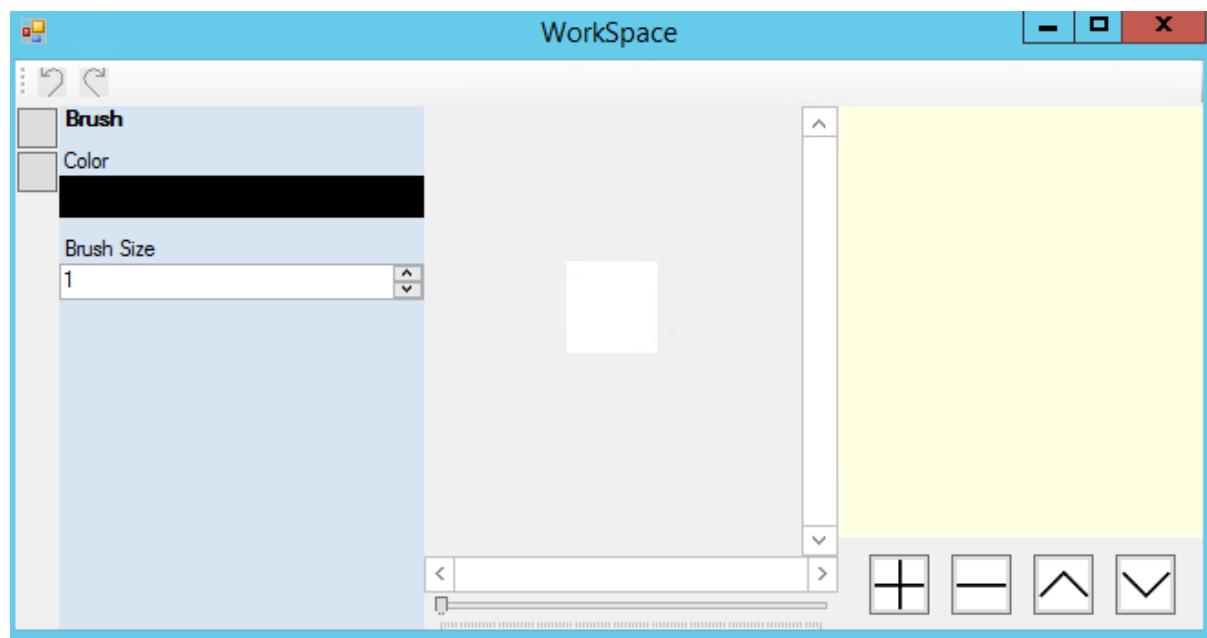
```
DisplayPixel(x,y) {
    FOR EACH layer IN layers
        IF layer.pixels[x,y] is transparent
            STOP REPEAT
        ELSE
            //Draw pixel on image (as before)
        END IF
    NEXT
}
```

```
foreach (Layer layer in layers) {
    // repeats through layers until a solid pixel is found
    if (layer.pixels[x,y].Color != Color.Transparent){  
        // Draws a rectangle using colour of current pixel, X  
        GFX.FillRectangle(layer.pixels[x,y],currentPoint.displ  
        break; // a pixel has been drawn - draw no more here
    }
}
```

It has been changed to stop repeating when a pixel is filled

## Layer Selector Design

A basic design for the layer selector has been added, according to Algorithm 3.12's updated design:



Though the actual layer displaying will be implemented shortly.

## 27/11/2019 Implementing Layer Buttons

### Layer Preview Rendering Optimisation

An implementation for drawing a layer preview has been implemented, in accordance to Algorithm 3.11A:

```
DrawIcon() {
    FOR x = 1 TO iconHeight
        FOR y = 1 TO iconHeight
            imageX = (x * imageWidth) / iconWidth
            imageY = (y * imageHeight) / iconHeight
            DrawRectangle(x,y,1,1,colours[imageX,imageY])
        NEXT
    NEXT
}
```

```
private void UpdateLayersSelector() {
    foreach (Layer layer in image.layers) {
        System.Drawing.Image newImage = new System.Drawing.Bitmap(40,40);
        Graphics GFX = Graphics.FromImage(newImage);
        for (int x = 0; x < 40; x++) {
            for (int y = 0; y < 40; y++) {
                int imageX = (x * image.fileWidth) / 40;
                int imageY = (y * image.fileHeight) / 40;
                GFX.FillRectangle(layer.pixels[imageX,imageY],x,y,1,1);
            }
        }
        previewImages[layer].Image = newImage;
    }
}
```

However this is quite inefficient, as it redraws every layer each time, even when only one layer is updated. This can be resolved by replacing the foreach with just the currentLayer.

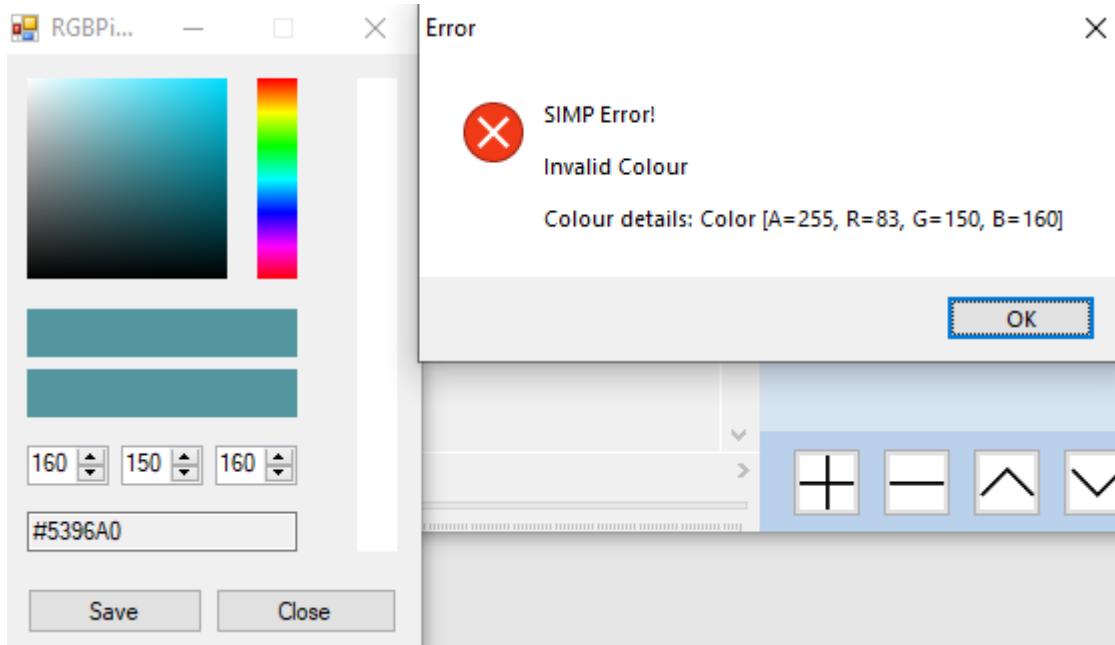
```
private void UpdateLayersSelector(bool full) {
    if (full) {
        foreach (Layer layer in image.layers) {
            UpdateLayer(layer);
        }
    } else {
        UpdateLayer(image.currentLayer);
    }
}
```

Then, during runtime the false Boolean is passed and only the currentlayer is updated. This is much better for performance.

## Colour Interpretation Error

An issue that many stakeholders had reported was a crash when using the colour picker. The error appeared to stem from some colour values in the RGB conversions, however as the program force closed it was never clear what the problem colour was.

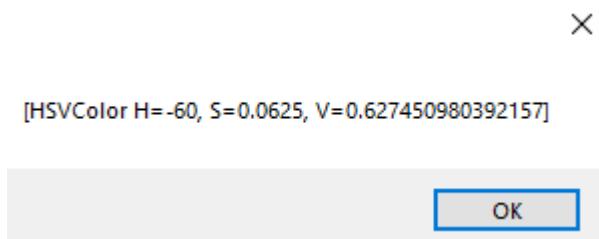
So, to combat this, an error handler was implemented, asking the user to report any such crash causing colours. Soon enough, a problem colour was isolated:



The colour is R160, G150, B160. This was able to be reported due to the error handler. Now this problem can be investigated.

When attempting to run a conversion on this colour from RGB to HSV, the problem becomes apparent:

```
MessageBox.Show(SIMP_RGB.RGBtoHSV(160,150,160).ToString());
```



The Hue is -60, which is clearly invalid.

The introduction of the negative was isolated to these lines of code:

```
if (Δ == 0) {
    h = 0;
} else if (Cmax == R) {
    h = 60 * (((G-B)/6)%6);
} else if (Cmax == G) {
    h = 60 * (((B-R)/6)+2);
} else if (Cmax == B) {
    h = 60 * (((R-G)/6)+4);
}
```

Any of these subtractions could introduce a negative sign, if R > G > B for example.

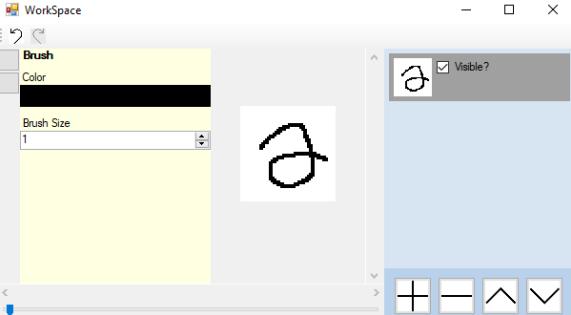
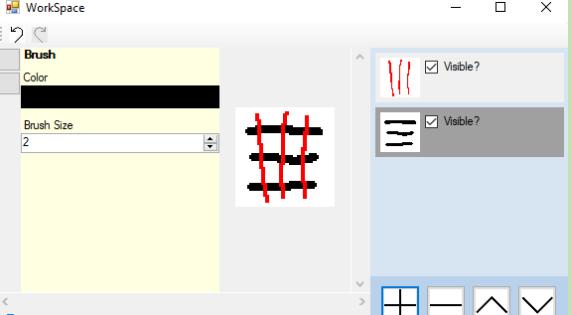
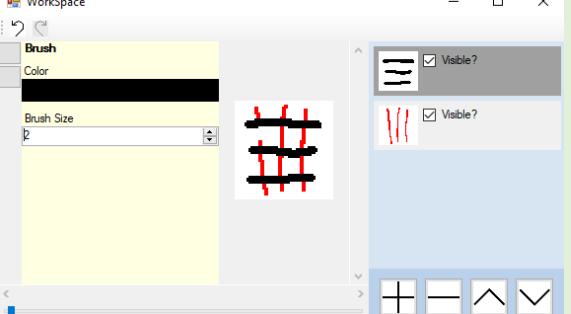
To fix this, as Hue is interpreted as an 'angle' (between 0°-360°), 360° can be added to 'wrap' it around if necessary, similar to how a turn of -40° is equal to a turn of 320°

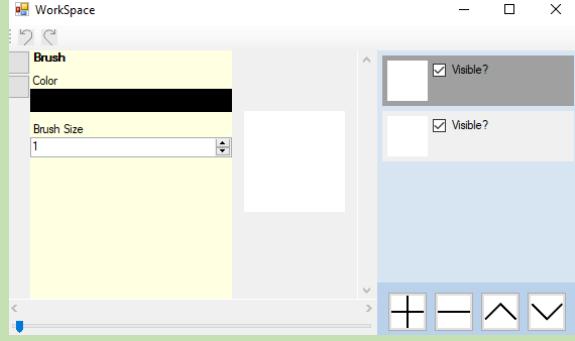
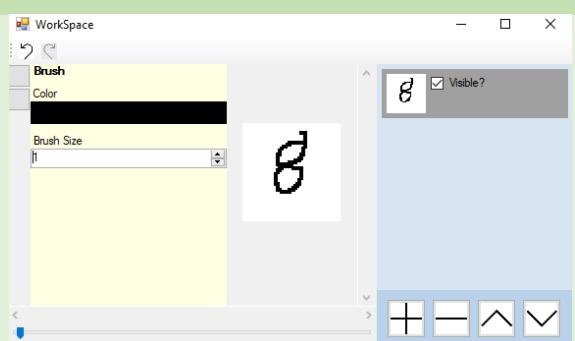
```
if (h<0) {
    h += 360;
}
```

This means the problem colour can now successfully be manipulated:



## 28/11/2019 Layer Unit Test

Test Result	ID	Expected Result	Actual Result	Comment
Select the top layer and draw onto it	1	There is drawing on layer		The layer can be drawn upon and the preview updates
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one	System.ArgumentOutOfRangeException	The program crashes as there is no prevention of removing only layer
Select lower layer and draw onto it at same location as top layer	3	There is drawing on lower layer, but it is obviously below		The program obviously draws the lower layer below the red one
Move lower layer up	4	The lower layers moves on top of previous top layer		The program now puts the thicker lines on top
Move the highest layer up	5	Should be impossible as it cannot go up	System.ArgumentOutOfRangeException	The program does not have any checking for this
Move the lowest layer down	6	Should be impossible as it cannot go down	System.ArgumentOutOfRangeException	The program also does not check for this

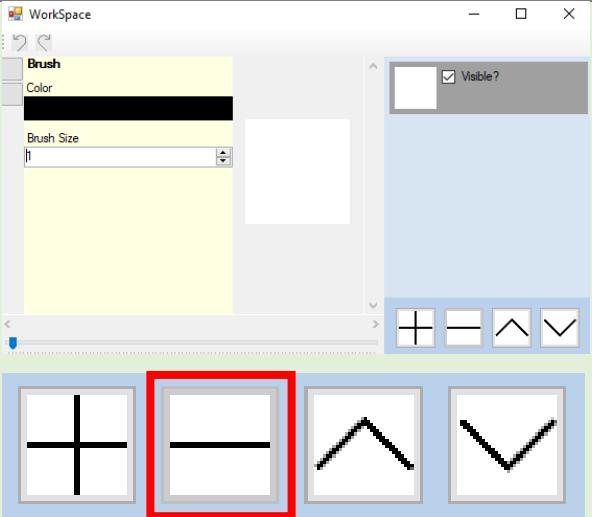
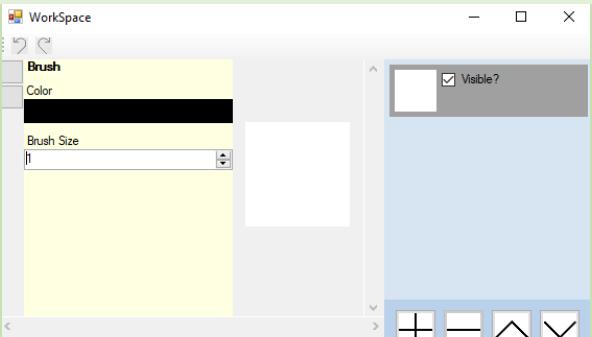
Add a new layer	7 A new (empty) layer is created		The program does this
Add 10 new layers	8 Many new layers are added		The program is able to create a large amount of layers
Layers are labelled then randomly rearranged	9 The layers are moved		The program does not crash when moving about this large amount of layers
The 11 layers are deleted	10 The layers disappear in the order in which they are added in the list		The program is able to then remove all of the layers
The final layer is deleted	11 Prevented as there is only one layer	<p>System.ArgumentOutOfRangeException</p>	There is still no check for this

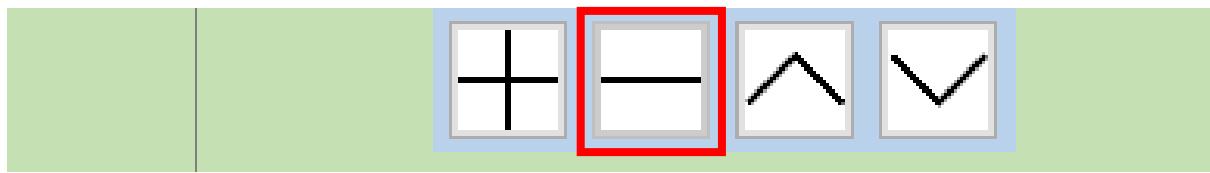
## Fixing Error 2 & 11

Test	ID	Expected Result	Actual Result	Comment
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one	System.ArgumentOutOfRangeException	The program crashes as there is no prevention of removing only layer
The final layer is deleted	11	Prevented as there is only one layer	System.ArgumentOutOfRangeException	There is still no check for this

In order to fix this error, upon refreshing the amount of layers can be checked:

```
btnRemoveLayer.Enabled = image.layers.Count > 1;
```

Test	ID	Expected Result	Actual Result	Comment
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one		The remove button is now disabled
The final layer is deleted	11	Prevented as there is only one layer		The remove button is now disabled

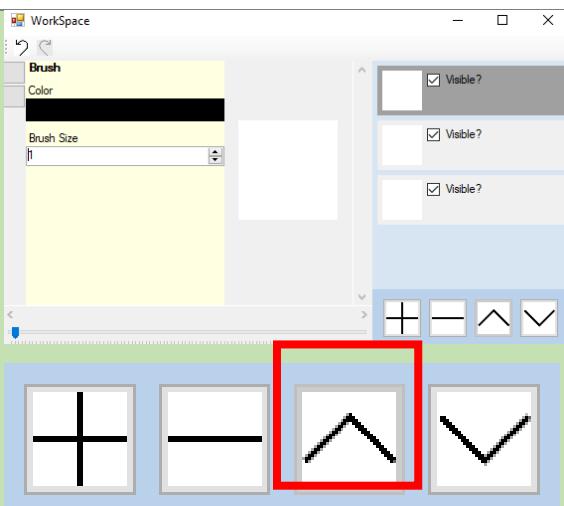


## Fixing Error 5

Test Result	ID	Expected Result	Actual Result	Comment
Move the highest layer up	5	Should be impossible as it cannot go up	System.ArgumentOutOfRangeException	The program does not have any checking for this

To fix this, a line of code can be implemented to check where the current layer is:

```
btnLayerUp.Enabled = image.layers.IndexOf(image.currentLayer) != 0;
```

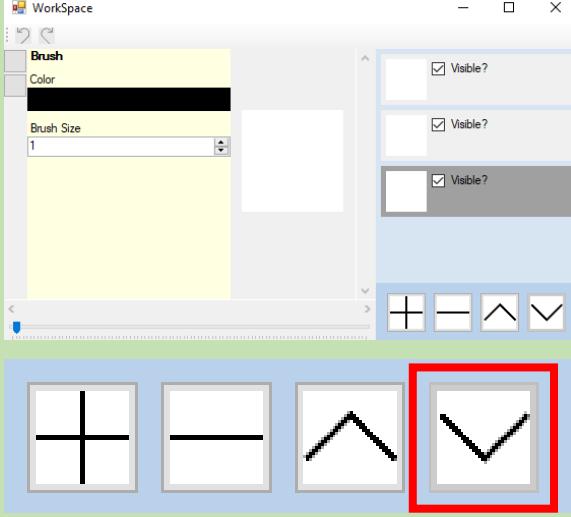
Test Result	ID	Expected Result	Actual Result	Comment
Move the highest layer up	5	Should be impossible as it cannot go up		The program now disables the button when needed

## Fixing Error 6

Test	ID	Expected Result	Actual Result	Comment
Move the lowest layer down	6	Should be impossible as it cannot go down	System.ArgumentOutOfRangeException	The program also does not check for this

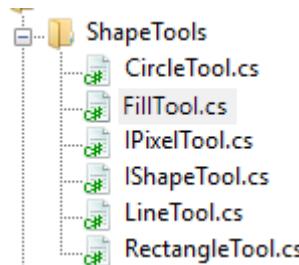
This can be also fixed with a small line of code:

```
btnLayerDown.Enabled = image.layers.IndexOf(image.currentLayer) != (image.layers.Count - 1);
```

Test	ID	Expected Result	Actual Result	Comment
Move the lowest layer down	6	Should be impossible as it cannot go down		The program now disables the button when needed

## 29/11/2019 Shape framework

The necessary classes have now been added to implement the various extra tools:



Thus, following the plan laid out in Algorithm 3.15, each class is given its duties. This means that the IShapeTool class handles everything generic to any shape:

```

public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation)...
public override void HandleMouseClick(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)...
public override void HandleMouseUp(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)...
public override void HandleMouseDown(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)...
public override void AddShapePoint(int x, int y)...
  
```

So that the specific shape (in this case LineTool) only needs to provide code for drawing itself, everything else has already been implemented:

```

public override void GenShape()
{
    throw new NotImplementedException();
}
  
```

Then, the algorithm for resolving points was implemented in accordance to 3.15A:

```

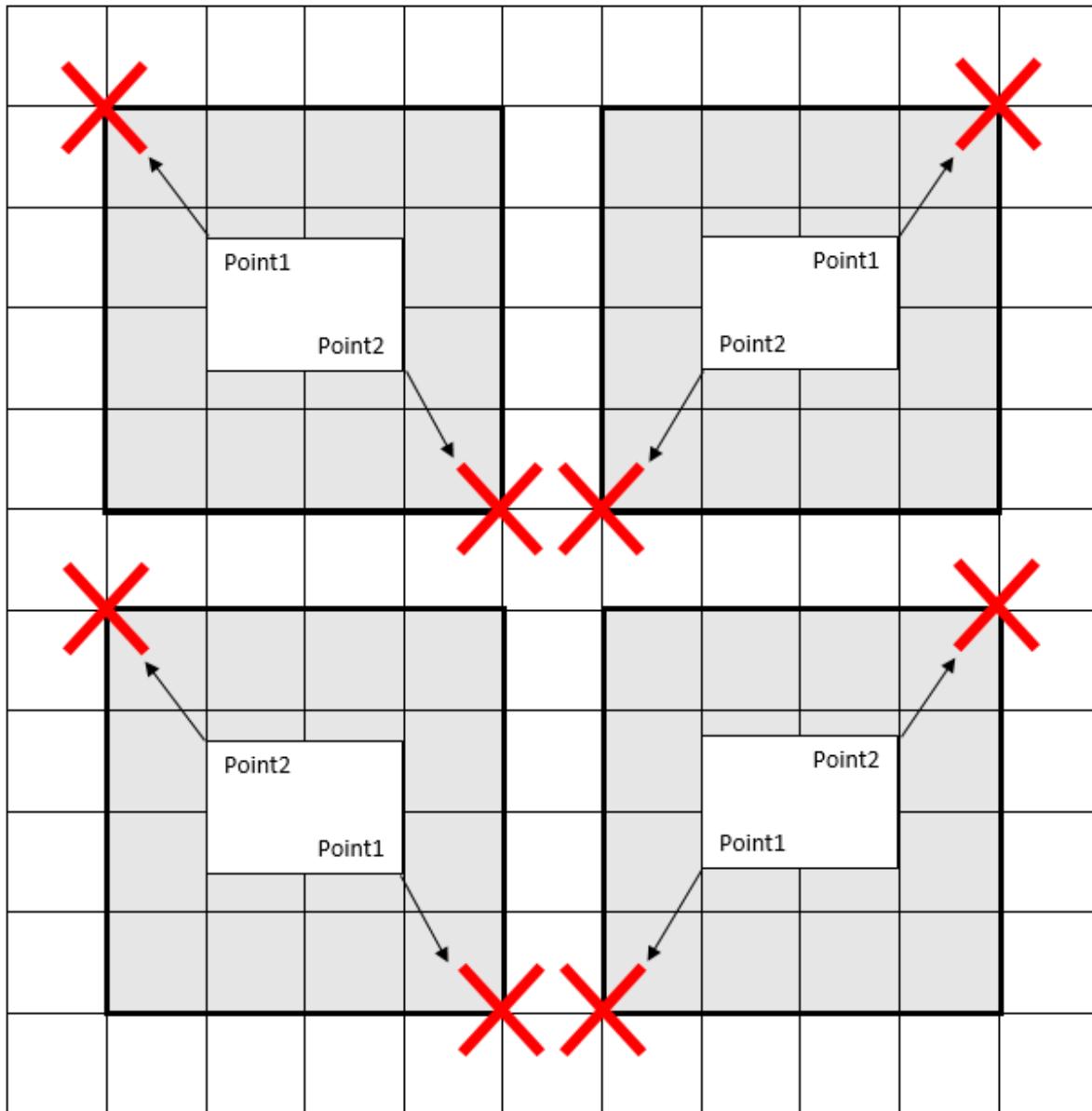
internal void ResolvePoints() {
    int highX, highY, lowX, lowY = 0;

    if (point1.fileX > point2.fileX) {
        highX = point1.fileX;
        lowX = point2.fileX;
    } else {
        highX = point2.fileX;
        lowX = point1.fileX;
    }

    if (point1.fileY > point2.fileY) {
        highY = point1.fileY;
        lowY = point2.fileY;
    } else {
        highY = point2.fileY;
        lowY = point1.fileY;
    }

    point1 = new FilePoint(lowX, lowY);
    point2 = new FilePoint(highX, highY);
}
  
```

However, this leads to a problem. The `ResolvePoint` algorithm is meant to collapse these four scenarios into a single scenario:



However this does not apply for the `LineTool`, as the line drawing algorithm will already resolve points by itself. So thus the `LineTool` class needs to **override** the existing method in order to disable it:

```
internal override void ResolvePoints() {
    // do nothing
}
```

Then, after this, adding code for each specific tool is simply. It need only override one method. For example the rectangle tool:

```
internal override void GenShape() {
    for (int x = point1.fileX; x < point2.fileX; x++) {
        for (int y = point1.fileY; y < point2.fileY; y++) {
            shapePoints.Add(new FilePoint(x,y));
        }
    }
}
```

Then, when adding the diamond tool, a formula for generating diamonds was needed. Thankfully this is very similar to the circle formula:

$$\frac{(x - (x' + 0.5))^2}{w^2} + \frac{(y - (y' + 0.5))^2}{h^2} \leq 1$$

To make this into the diamond formula, the square is switched for an Absolute operation:

$$\frac{|x - (x' + 0.5)|}{w} + \frac{|y - (y' + 0.5)|}{h} \leq 1$$

Thus the shape tools all have formulas, and can be implemented.

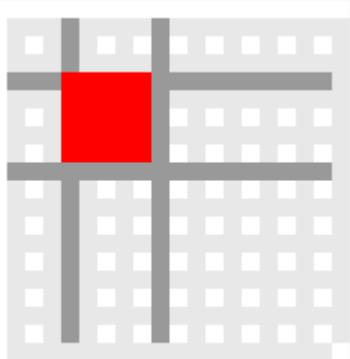
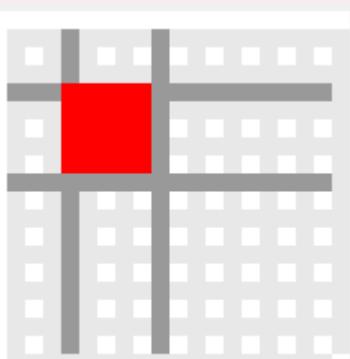
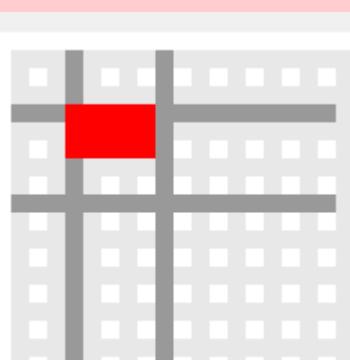
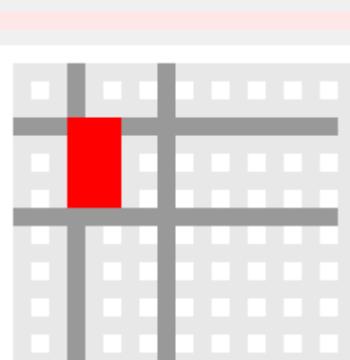
After this, it is now time to do the full section testing.

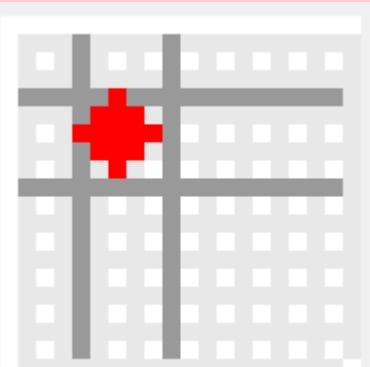
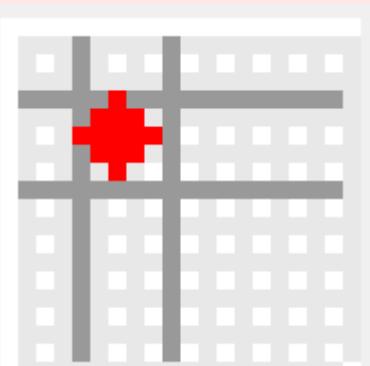
## 02/12/2019 Shape Unit Test

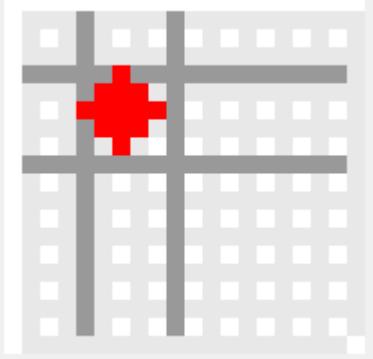
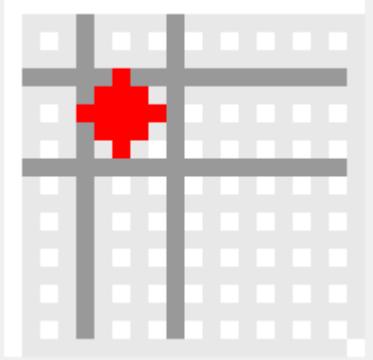
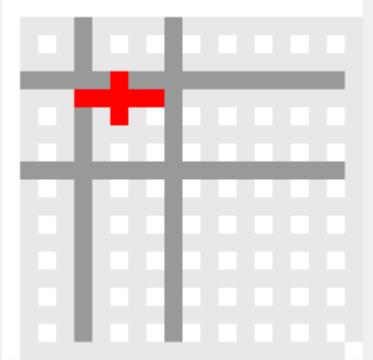
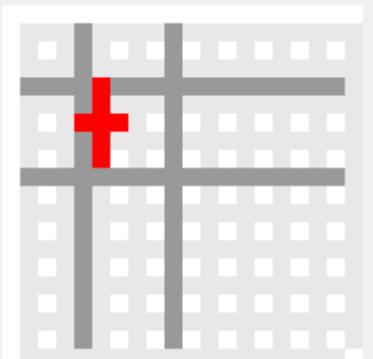
Test	ID	Expected Result	Actual Result	Comment
Create line with points (5,5) & (10,10)	1	A line is drawn from (5,5) to (10,10)		The line does not include the two starting points
Create line with points (5,10) & (10,5)	2	A line is drawn from (5,10) to (10,5)		The line does not include the two starting points

<i>Create line with points (10,5) &amp; (5,10)</i>	3	A line is drawn from (10,5) to (5,10)		The line does not include the two starting points
<i>Create line with points (10,10) &amp; (5,5)</i>	4	A line is drawn from (10,10) to (5,5)		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (10,8)</i>	5	A shorter line is drawn		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (8,10)</i>	6	A thinner line is drawn		The line does not include the two starting points

<i>Create line with points (5,5) &amp; (10,5)</i>	7	A single row is drawn		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (5,10)</i>	8	A single column is drawn		The line does not include the two starting points
<i>Create rectangle with points (5,5) &amp; (10,10)</i>	9	A rectangle is drawn from (5,5) to (10,10)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,10) &amp; (10,5)</i>	10	A rectangle is drawn from (5,10) to (10,5)		The rectangle has not included the coords of the lower point in its calculations

<i>Create rectangle with points (10,5) &amp; (5,10)</i>	11 A rectangle is drawn from (10,5) to (5,10)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (10,10) &amp; (5,5)</i>	12 A rectangle is drawn from (10,10) to (5,5)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (10,8)</i>	13 A shorter rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (8,10)</i>	14 A thinner rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations

<i>Create rectangle with points (5,5) &amp; (10,5)</i>	15 A single row is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (5,10)</i>	16 A single column is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (10,10)</i>	17 A circle is drawn from (5,5) to (10,10)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,10) &amp; (10,5)</i>	18 A circle is drawn from (5,10) to (10,5)		The circle has not included the coords of the lower point in its calculations

<i>Create circle with points (10,5) &amp; (5,10)</i>	19	A circle is drawn from (10,5) to (5,10)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (10,10) &amp; (5,5)</i>	20	A circle is drawn from (10,10) to (5,5)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (10,8)</i>	21	A shorter circle is drawn		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (8,10)</i>	22	A thinner circle is drawn		The circle has not included the coords of the lower point in its calculations

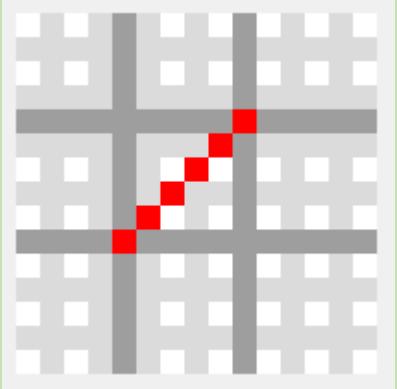
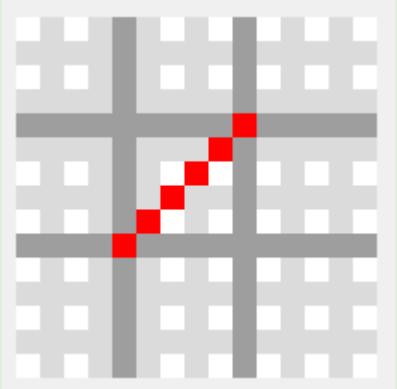
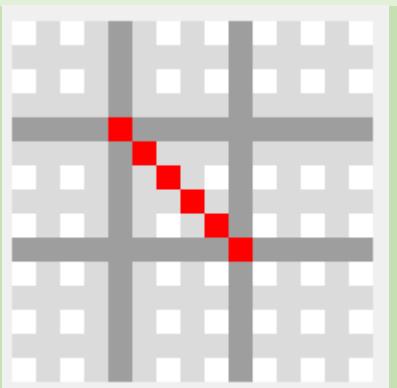
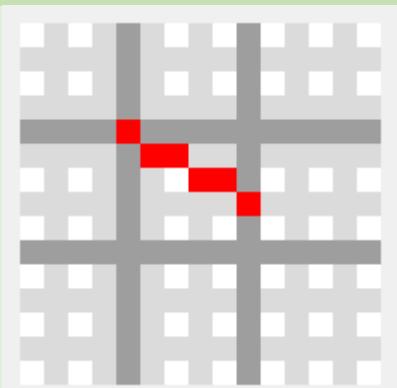
Create circle with points (5,5) & (10,5)	23	A single row is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (5,10)	24	A single column is drawn		The circle has not included the coords of the lower point in its calculations

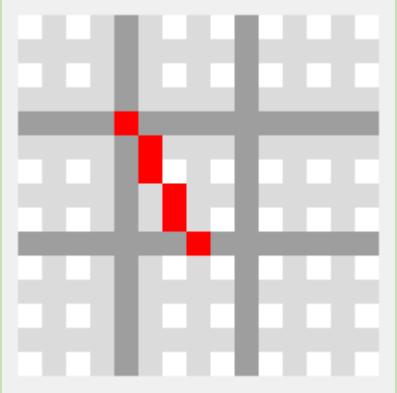
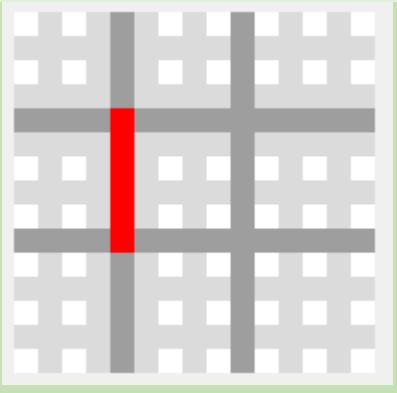
## Fixing Error #1-#8

To fix these errors, the program can be made to forcefully set the start and end points of the line:

```
// makes the the start and end points are part of the line
AddShapePoint(point1.labelX,point1.fileY);
AddShapePoint(point2.labelX,point2.fileY);
. . .
```

Test	ID	Expected Result	Actual Result	Comment
Create line with points (5,5) & (10,10)	1	A line is drawn from (5,5) to (10,10)		The line does not include the two starting points

<i>Create line with points (5,10) &amp; (10,5)</i>	2	A line is drawn from (5,10) to (10,5)		The line does not include the two starting points
<i>Create line with points (10,5) &amp; (5,10)</i>	3	A line is drawn from (10,5) to (5,10)		The line does not include the two starting points
<i>Create line with points (10,10) &amp; (5,5)</i>	4	A line is drawn from (10,10) to (5,5)		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (10,8)</i>	5	A shorter line is drawn		The line does not include the two starting points

<i>Create line with points (5,5) &amp; (8,10)</i>	6	A thinner line is drawn		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (10,5)</i>	7	A single row is drawn		The line does not include the two starting points
<i>Create line with points (5,5) &amp; (5,10)</i>	8	A single column is drawn		The line does not include the two starting points

## Fixing Error #9-#24

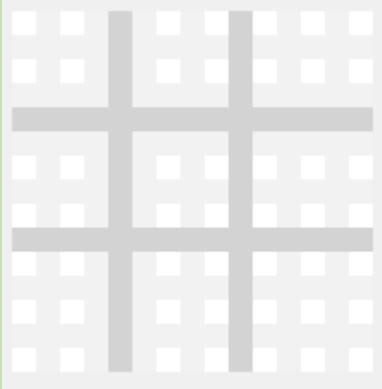
To fix this error, the terminator for the for loop can now become `>=` instead of `>`

<i>Test Result</i>	<i>ID</i>	<i>Expected Result</i>	<i>Actual Result</i>	<i>Comment</i>
<i>Create rectangle with points (5,5) &amp; (10,10)</i>	9	A rectangle is drawn from (5,5) to (10,10)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,10) &amp; (10,5)</i>	10	A rectangle is drawn from (5,10) to (10,5)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (10,5) &amp; (5,10)</i>	11	A rectangle is drawn from (10,5) to (5,10)		The rectangle has not included the coords of the lower point in its calculations

<i>Create rectangle with points (10,10) &amp; (5,5)</i>	12	A rectangle is drawn from (10,10) to (5,5)		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (10,8)</i>	13	A shorter rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (8,10)</i>	14	A thinner rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create rectangle with points (5,5) &amp; (10,5)</i>	15	A single row is drawn		The rectangle has not included the coords of the lower point in its calculations

<i>Create rectangle with points (5,5) &amp; (5,10)</i>	16	A single column is drawn		The rectangle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (10,10)</i>	17	A circle is drawn from (5,5) to (10,10)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,10) &amp; (10,5)</i>	18	A circle is drawn from (5,10) to (10,5)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (10,5) &amp; (5,10)</i>	19	A circle is drawn from (10,5) to (5,10)		The circle has not included the coords of the lower point in its calculations

<i>Create circle with points (10,10) &amp; (5,5)</i>	20	A circle is drawn from (10,10) to (5,5)		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (10,8)</i>	21	A shorter circle is drawn		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (8,10)</i>	22	A thinner circle is drawn		The circle has not included the coords of the lower point in its calculations
<i>Create circle with points (5,5) &amp; (10,5)</i>	23	A single row is drawn		The circle has not included the coords of the lower point in its calculations

<i>Create circle with points (5,5) &amp; (5,10)</i>	24 A single column is drawn		The circle has not included the coords of the lower point in its calculations
---	-----------------------------	--	---

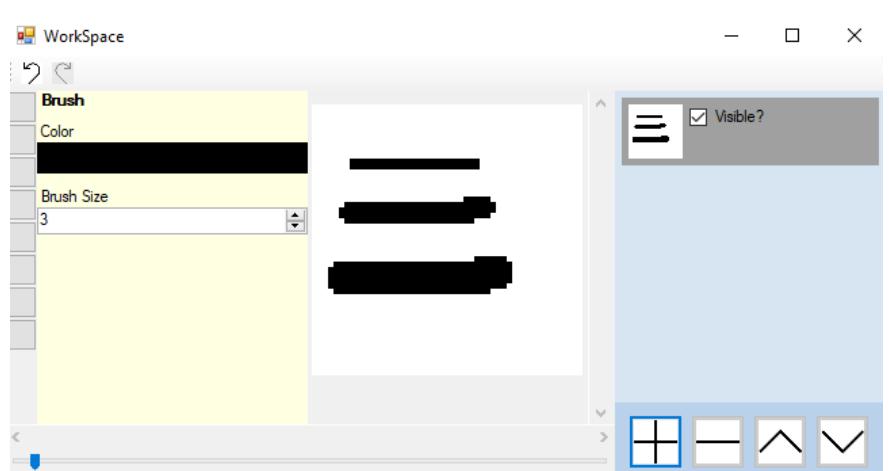
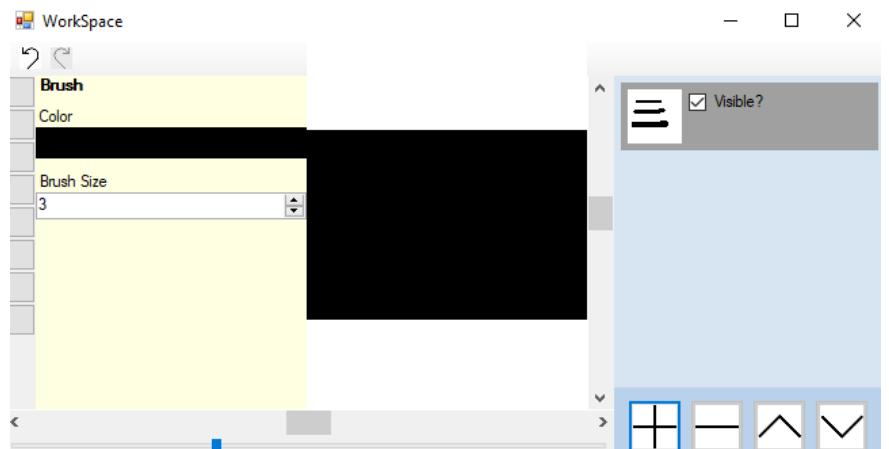
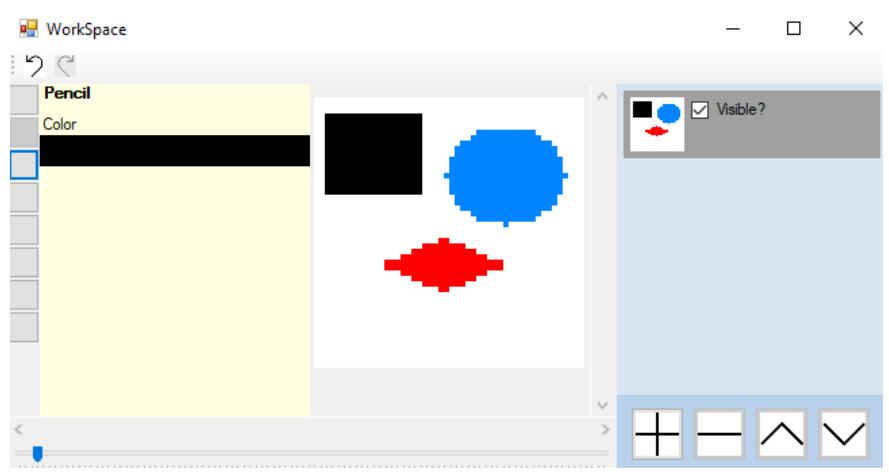
### 3.2.1 Stakeholder Feedback Drawings

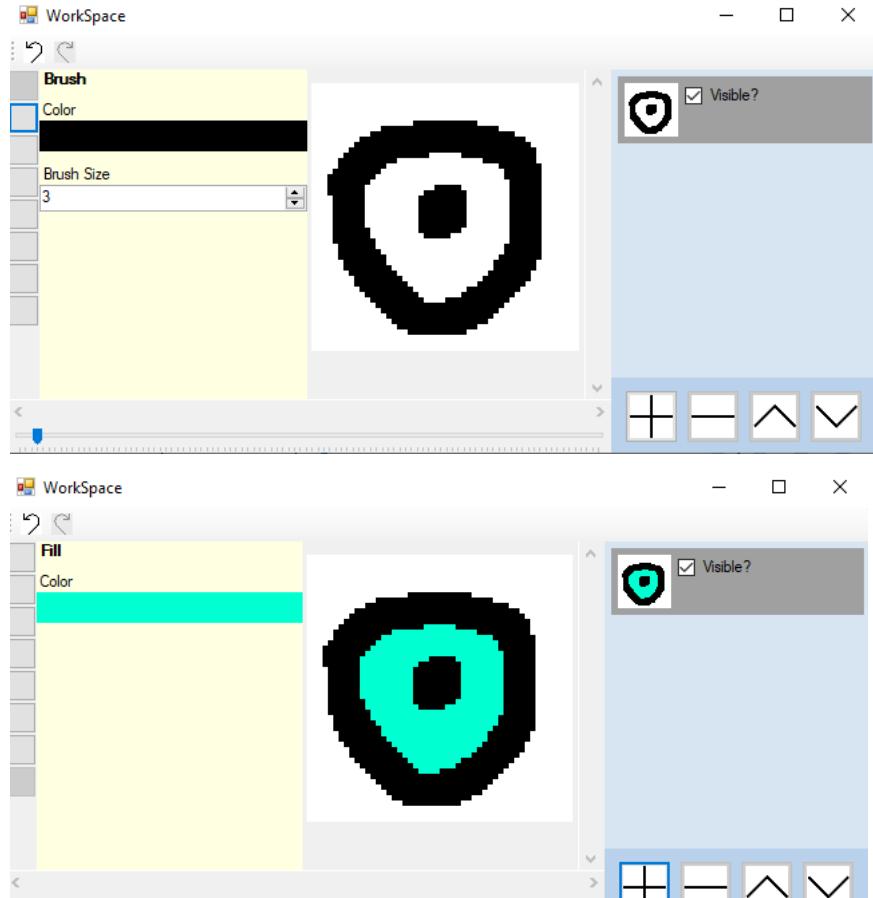
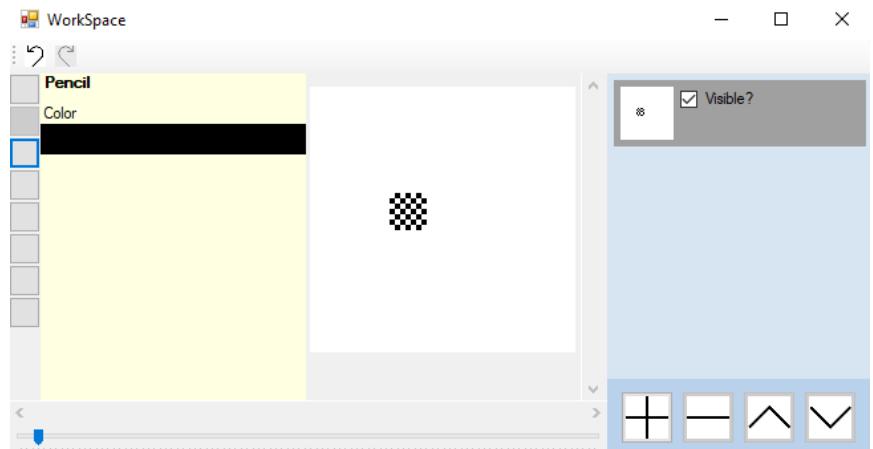
My stakeholders each drew images using SIMP, with the latest tools.

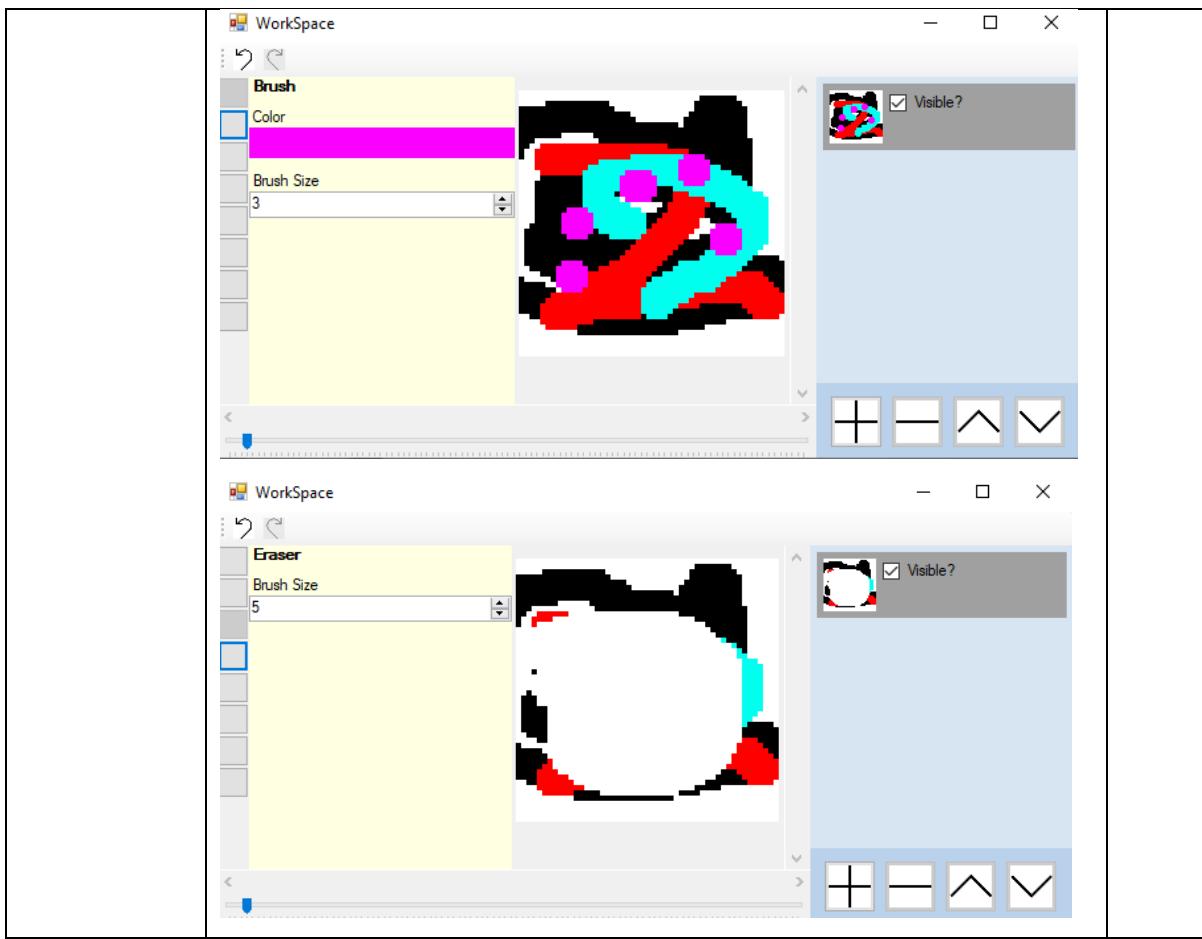
Dhespreet



### 3.2.2 Success Criteria Evaluation

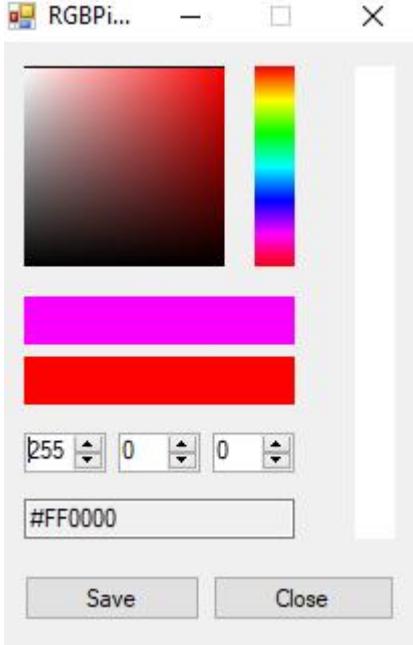
Feature	Proof	Code
Section A - Brushes		
Variable brush width	<p>Screenshot of strokes of the same brush showing different widths</p> 	A1
Hard brushes	<p>Screenshot showing the hard edge of the brush (colour to no colour)</p> 	A2
Shape creation tools	<p>Screenshot showing the shape toolbar and a small selection of drawn shapes</p> 	A3

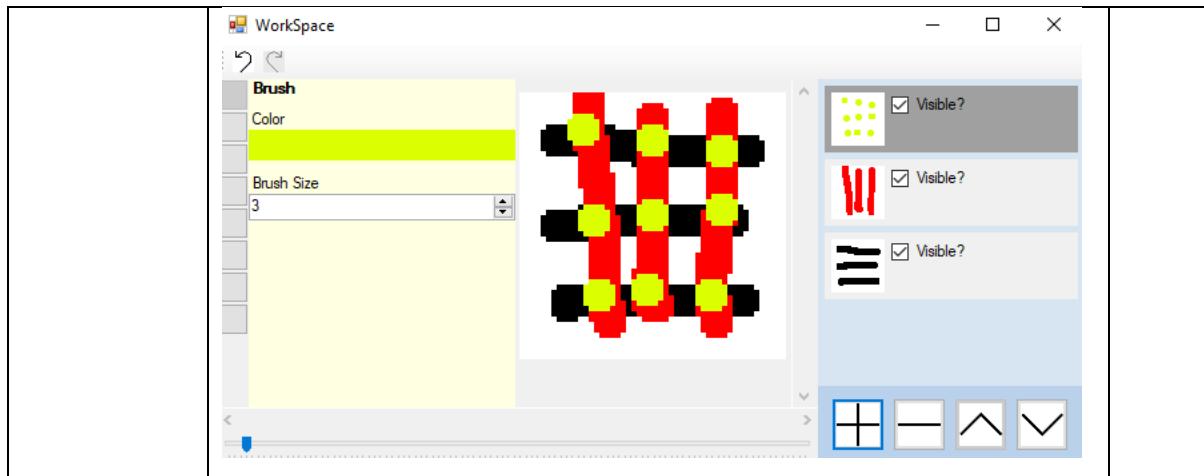
Fill (bucket) tool	<p>Screenshot showing a before and after of filling a large area</p> 	A4
Single pixel pencil	<p>Screenshot showing a stroke of the single pixel brush</p> 	A5
Rubber	<p>Screenshot showing a densely packed picture being rubbed out</p>	A6



### Section B – Other editing tools

RGB colour picker	<p>Screenshot showing a system for entering an RGB colour</p>	B3
RGB direct input	<p>Screenshot showing the user entering "FF0000" (or equivalent) and the programming outputting red</p>	B4

		
Layer system	Screenshot of layer navigator	B5
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8



Section C – File System

Creating a new image		C1
----------------------	---	----

So this makes the current success criteria:

Not completed

Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6

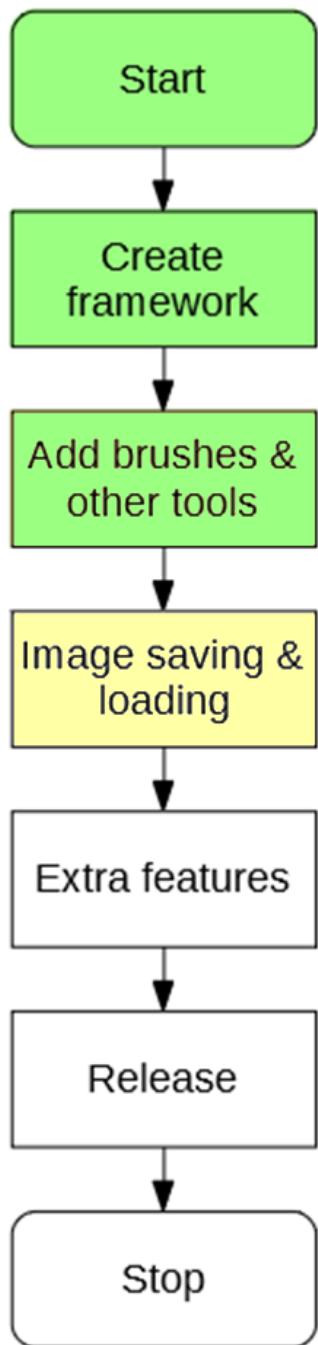
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11
<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5

Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

# Section 4 – Saving & Loading

## 4.1 Design

This section will contain the systems needed to save, load, import and export images. This will be able to export the image created by the previous section:



## 4.1.1 Success Criteria Fulfilment Plan

In this section the following success criteria are planned to be completed:

Not completed

To be done this section

Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11

<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

## 4.1.2 Saving a file

---

Before loading must come saving. This is where all of the necessary contents of the image are exported onto a file on disk, which can be opened again by the program and contain **no loss of necessary information**. This means that some info (such as the last brush colour used) is not important and can be lost.

### File Plan

To save the file, the program must save into a file:

HEADER:

IMAGE HEIGHT  
IMAGE WIDTH  
NUMBER OF LAYERS

BODY:

TOP LAYER:

COLOUR OF (0,0)  
COLOUR OF (1,0)  
...  
COLOUR OF (width-1,height-1)

SECOND LAYER:

COLOUR OF (0,0)  
COLOUR OF (1,0)  
...  
COLOUR OF (width-1,height-1)

...

LAST LAYER:

COLOUR OF (0,0)  
COLOUR OF (1,0)  
...  
COLOUR OF (width-1,height-1)

This file plan contains two main parts, the Header and Body. This is similar to the form of a HTML page.

- Header will be fixed in size and contain the width and height of the image.
  - The dimensions are needed so that when the program is reading from the file, it knows that for each layer it must load width \* height pixels.
  - The number of layers are needed so that when the program is reading from the file, it knows how many times to load a layer.
- Body will be variable in size and will contain the information about each layer of the image.

## Header Pseudocode

Writing to the header should be a reasonably simple affair.

```
PROCEDURE WriteHeader(file) {
    file.Write(image.width)
    file.Write(image.height)
    file.Write(image.layers.count)
}
```

## Body Pseudocode

Writing the body should be also simple, though a lot more data will be written:

```
PROCEDURE WriteBody(file) {
    FOR EACH layer IN image.layers
        FOR x = 0 TO image.width
            FOR y = 0 to image.height
                color = layer.GetColor(x,y)
                file.Write(color.R)
                file.Write(color.G)
                file.Write(color.B)
            NEXT y
        NEXT x
    NEXT layer
}
```

This makes saving to the file a much simpler task

Writes R, G, and B as  
three separate values

### 4.1.3 Loading a file

---

Loading from the file may be a more difficult task, as there is only raw binary to work with. However with reference to the [file plan](#) it should be obvious how to load the file.

#### Header Pseudocode

Assuming there are three values in the file, loading should be a simple affair:

```
PROCEDURE LoadHeader(file) {
    imageWidth = file.Read()
    imageHeight = file.Read()
    layerCount = file.Read()
}
```

#### Body Pseudocode

After reading the values from the header, it becomes possible load the next of the file like so:

```
PROCEDURE LoadBody(file) {
    FOR i = 0 TO layerCount
        layer = new Layer(imageWidth, imageHeight)
        FOR x = 0 TO imageWidth
            FOR y = 0 TO imageHeight
                R = file.Read()
                G = file.Read()
                B = file.Read()
                layer.SetPixel(new Color(R,G,B))
            NEXT y
        NEXT x
    NEXT i
}
```

## Handling Invalid Files

The previous code assumes that the file has not been corrupted at any point. Potential errors that may be thrown (and thus handled) are:

- `EndOfStreamException`, for if the file has ended prematurely.
- `InvalidParameterException`, if R, G or B is incorrect when creating the new colour.

So then, to make the file safe to load, the following **try-catch** blocks can be implemented to handle these sorts of errors, in the `LoadBody` procedure:

```
PROCEDURE LoadBody(file) {
    FOR i = 0 TO layerCount
        layer = new Layer(imageWidth, imageHeight)
        FOR x = 0 TO imageWidth
            FOR y = 0 TO imageHeight
                try {
                    R = file.Read()
                    G = file.Read()
                    B = file.Read()
                } catch EndOfStreamException {
                    break; } // As soon as the end of the
                    // file is reached, stop reading
                }
                try {
                    layer.SetPixel(new Color(R,G,B))
                } catch InvalidParameterException {
                    layer.SetPixel(new Color(0,0,0))
                }
            NEXT y
        NEXT x
    NEXT i
}
```

#### 4.1.4 Exporting to a PNG

---

This is quite a simple operation. In C# there exists a method of the `Bitmap` class (which is created when the display requires an image) which saves an image to a file. This will be used to avoid **reinventing the wheel**, and as the algorithms behind many popular image formats are elaborate and out of the scope of the project.

The user must also be able to tell the program where to save the image. This can be done by using an existing part of the .NET library, the file browser. This is used instead of a proprietary browser as the file system can be complex, and the Windows browser can use its own features such as favourites.

SIMP is an image editing program, not a file browsing program.

So thus the code for exporting is:

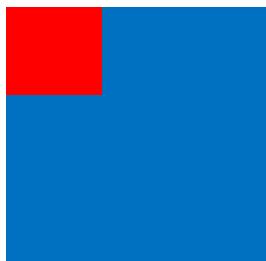
```
PROCEDURE export() {
    Prompt user for where to save file
    saveBitmap = image.RequestBitmap()
    saveBitmap.save()
}
```

#### 4.1.5 Importing from a PNG

---

Importing as much the same as exporting, as C# also contains inbuilt classes for this purpose. However once the image has been encapsulated into a bitmap, it must be placed onto the image. This can be done by creating a new layer, and transposing each pixel of the file onto the image, or check each pixel of the image. This is because if the file is much bigger than the image, then it would be expensive to iterate through each pixel in the image if it is mostly invisible:

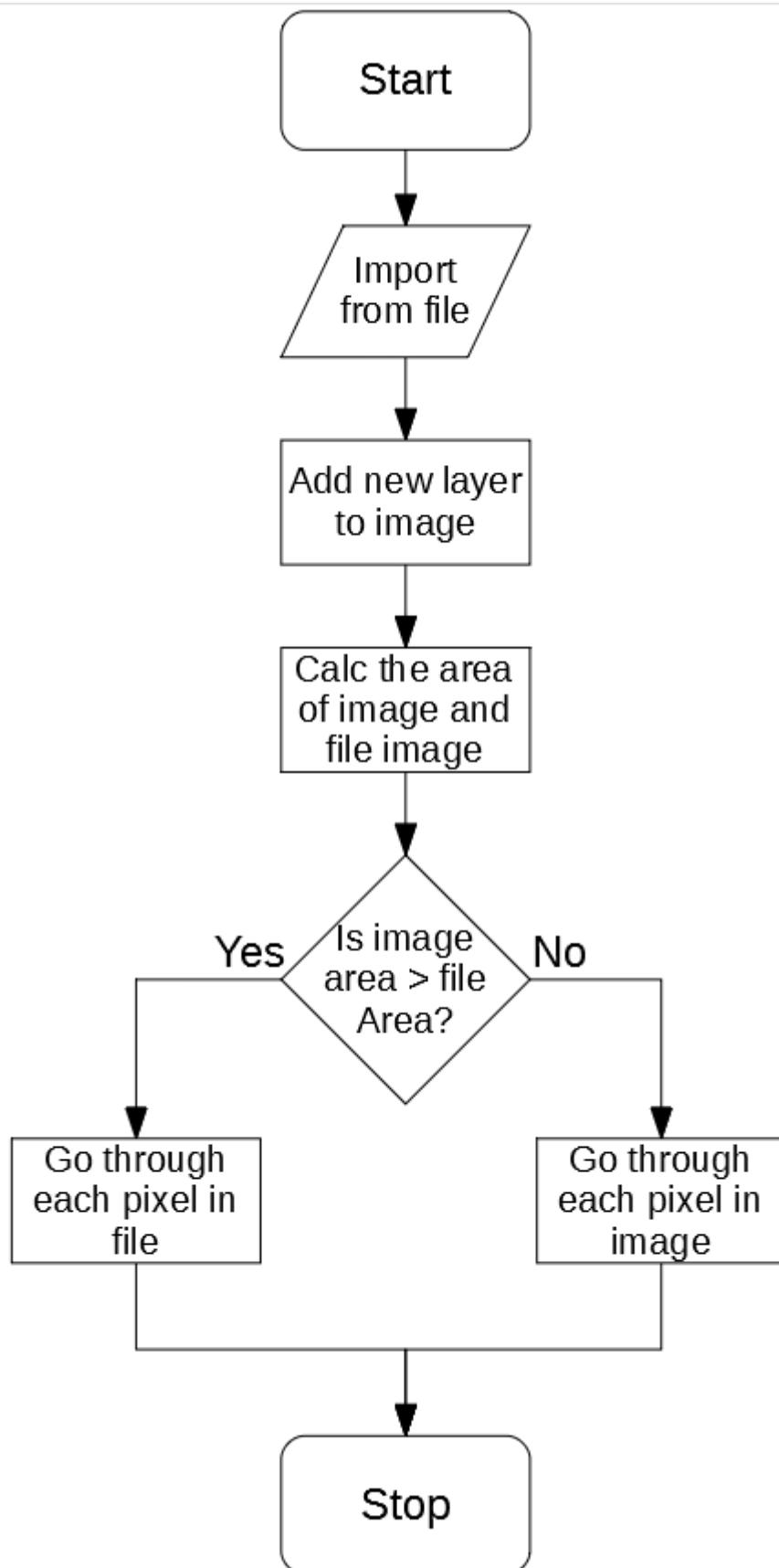
For example, if the red rectangle is the image, and the blue rectangle is the file trying to be imported:



Then it is clearly impractical to loop through each pixel of the file.

To decide which way around to iterate, a **heuristic** can be used. By finding the area of the image and file, a good guess can be made as to which one to iterate through (the one with the smaller area). This can be used to make sure that file loading is fast for larger images.

Thus the algorithm becomes:



## 4.1.6 New GUI design

---

When creating images like this, some new GUIs may be needed. The most important one being the dialogue to create an image:

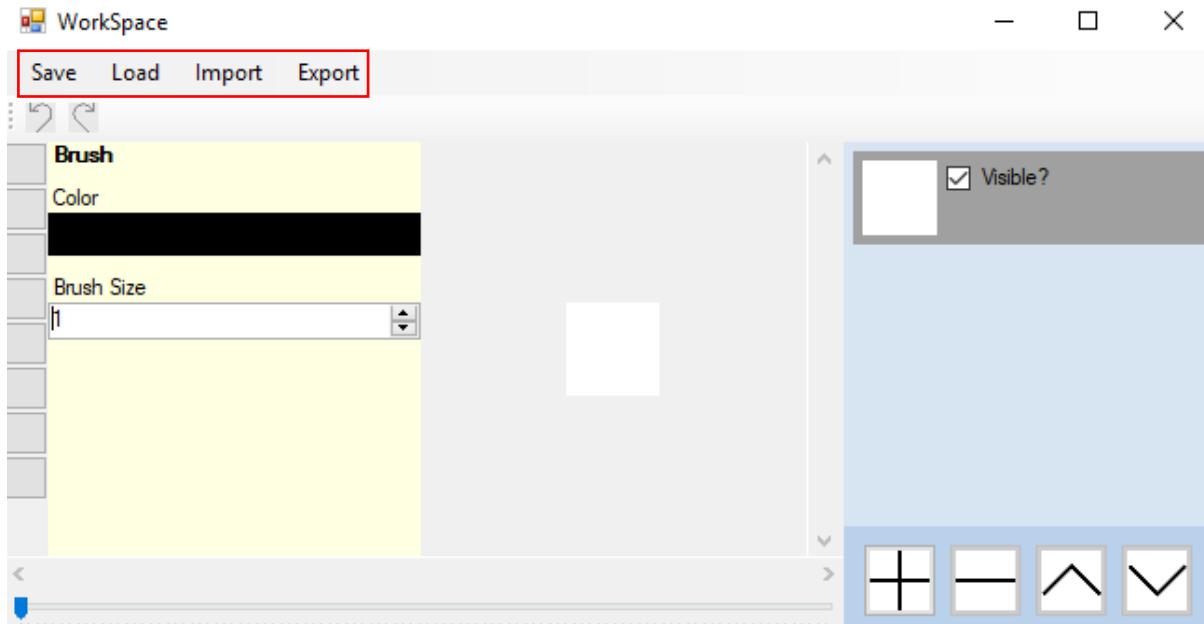


This will be started whenever a new image is needed to be made.

## 4.2 Development

### 04/12/2019 Form Design

The buttons necessary for the designed functions have been added to the form:



Stakeholders agreed that the top was a reasonable place to put those buttons, similar to other programs.

### 06/12/2019 Saving File

#### Writing algorithm

When saving the file, there are two potential classes that can be used for saving to a file; `StreamWriter` or `FileStream`.

`StreamWriter` allows saving of ASCII code to a file, `FileStream` saves raw binary. `StreamWriter` produces more easily readable files, but is less efficient as all numbers are encoded in ASCII. I **have decided** to use the `FileStream` class to reduce the file size of SIMP files and increase loading speed.

After this, the file saving algorithm can be implemented:

```

PROCEDURE WriteBody(file) {
    FOR EACH layer IN image.layers
        FOR x = 0 TO image.width
            FOR y = 0 to image.height
                color = layer.GetColor(x,y)
                file.Write(color.R)
                file.Write(color.G)
                file.Write(color.B)
            NEXT y
        NEXT x
    NEXT layer
}

void SaveFile(FileStream stream) {
    stream.WriteByte((byte)image.fileWidth);
    stream.WriteByte((byte)image.fileHeight);
    stream.WriteByte((byte)image.layers.Count);

    Color currentColor;
    foreach (Layer layer in image.layers) {
        for (int x = 0; x < image.fileWidth; x++) {
            for (int y = 0; y < image.fileHeight; y++) {
                currentColor = layer.pixels[x,y].Color;
                stream.WriteByte(currentColor.R);
                stream.WriteByte(currentColor.G);
                stream.WriteByte(currentColor.B);
            }
        }
    }
}

```

## Header Error

After programming the header, an error was detected. It occurred when one of the dimensions of the image was greater than 255. As this could not be saved into one byte, an error was thrown.

```

stream.WriteByte((byte)image.fileWidth);
stream.WriteByte((byte)image.fileHeight);
stream.WriteByte((byte)image.layers.Count);

```

This can be fixed by using two bytes each to store the width and height. This means that maximum size is increased from 255 to  $255^2$  (65,025), which is much larger than the maximum size. Whilst this may waste space for smaller images, the extra space needed is so minute that this is unlikely to be an issue.

It's assumed that 255 layers will never be created, so the same sort of error checking is not needed there.

This makes the new code for saving:

```

stream.WriteByte((byte)(image.fileWidth / 256)); // saves into two bytes
stream.WriteByte((byte)(image.fileWidth % 256));
stream.WriteByte((byte)(image.fileHeight / 256));
stream.WriteByte((byte)(image.fileHeight % 256));
stream.WriteByte((byte)image.layers.Count);

```

## 9/12/2019 More robust importing

---

Image importing has now been implemented according to 4.1.2, but it is currently not very robust. This especially happens when a non-SIMP file is attempted to be loaded.

This can be resolved by attaching the values 0x53, 0x49, 0x4D, 0x50 to the start of the file. This is the ASCII code for 'SIMP'. Thus when the file is attempting to be loaded, the program will check the first four bytes to see if they are 'SIMP'.

```
void SaveFile(FileStream stream) {
    //SIMP check digits
    stream.WriteByte((byte)'S');
    stream.WriteByte((byte)'I');
    stream.WriteByte((byte)'M');
    stream.WriteByte((byte)'P');

    void OpenFile(FileStream stream) {
        string checkString = "";
        for (int i = 0; i < 4; i++) {
            checkString += (char)stream.ReadByte();
        }
        if (!checkString.Equals("SIMP")) {
            MessageBox.Show("File cannot be loaded. \n\nSIMP data could not be found within this file.", "Loading Error",
                           MessageBoxButtons.OK);
            return;
        }
    }
}
```

This means non SIMP files will not be loaded.

## 10/12/2019 Importing & Exporting

---

Image importing and exporting can now be introduced.

These both use C# base classes for importing & exporting, as the algorithms tied with PNG, JPEG and BMP are complex. Implementing these would be far outside the scope of the project, so it is suitable to use them here.

## 11/12/2019 Stakeholder feedback

After giving the program to my stakeholders, they had the following feedback:

“Images should be imported to be centred in the image”  
“Exporting while the image is zoomed doesn’t work”  
“Image should be the same size as the canvas”  
“You should be able to open an image from the start page”

### Fixing Exporting

When the image is exported whilst the image is zoomed in, a zoomed in portion of the image was exported instead:

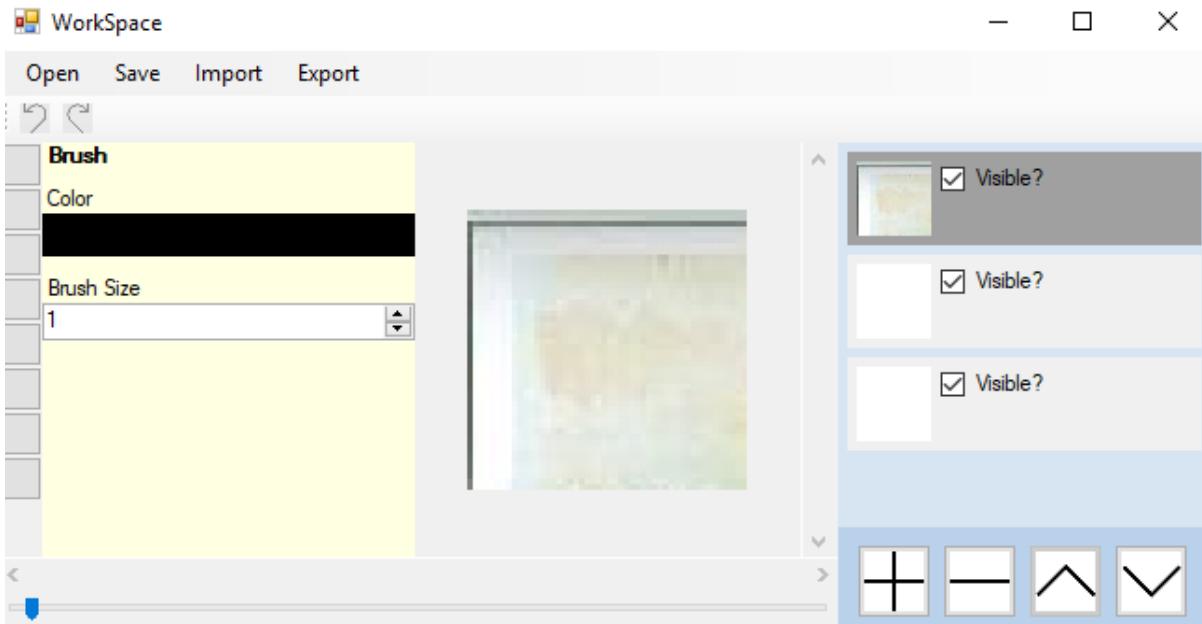


To fix this, the zoom of the image is temporarily set to 1, then exported. This means the proper image is always exported:

```
DialogResult result = diaExport.ShowDialog();
int tempZoom = image.zoomSettings.zoom;
image.zoomSettings.zoom = 1;
if (result == DialogResult.OK) {
    System.Drawing.Image saveImage = image.GetDisplayImage(image.fileWidth,image.fileHeight,true);
    saveImage.Save(diaExport.FileName);
}
image.zoomSettings.zoom = tempZoom;
```

## Fixing Importing

The importing however turned out to be quite flawed, for most images the result would be:



So, a different importing system needs to be used. A similar system to the one used for drawing the layer preview, where a foreach is used on each pixel of the image.

```
DrawIcon() {
    FOR x = 1 TO iconHeight
        FOR y = 1 TO iconHeight
            imageX = (x * imageWidth) / iconWidth
            imageY = (y * imageHeight) / iconHeight
            DrawRectangle(x,y,1,1,colours[imageX,imageY])
    NEXT
    NEXT
}
```

Changing the code for importing to:

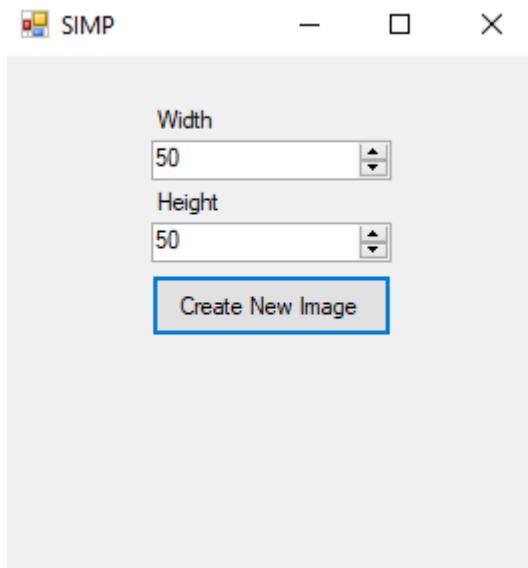
```
for (int x = 0; x < image.fileWidth; x++) {
    for (int y = 0; y < image.fileHeight; y++) {
        int imageX = (x * file.Width) / image.fileWidth;
        int imageY = (y * file.Height) / image.fileHeight;
        newLayer.pixels[x,y] = new SolidBrush(file.GetPixel(imageX,imageY));
    }
}
```

This means that images can now be properly imported:

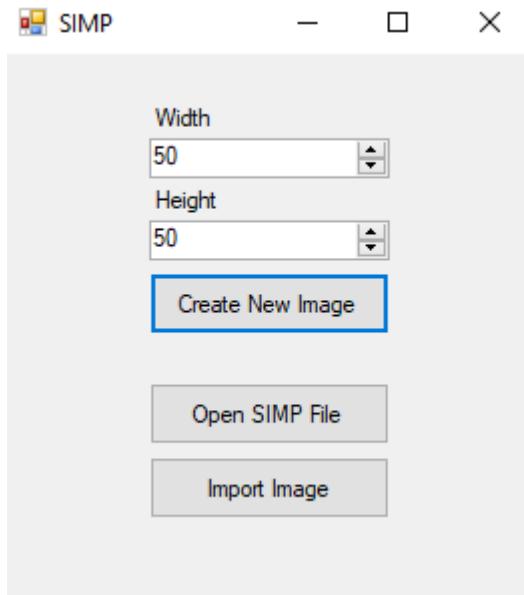


### Improving Start page

Currently the start page for SIMP is very basic, only allowing to create an image, and without any options to import or open an existing SIMP file:



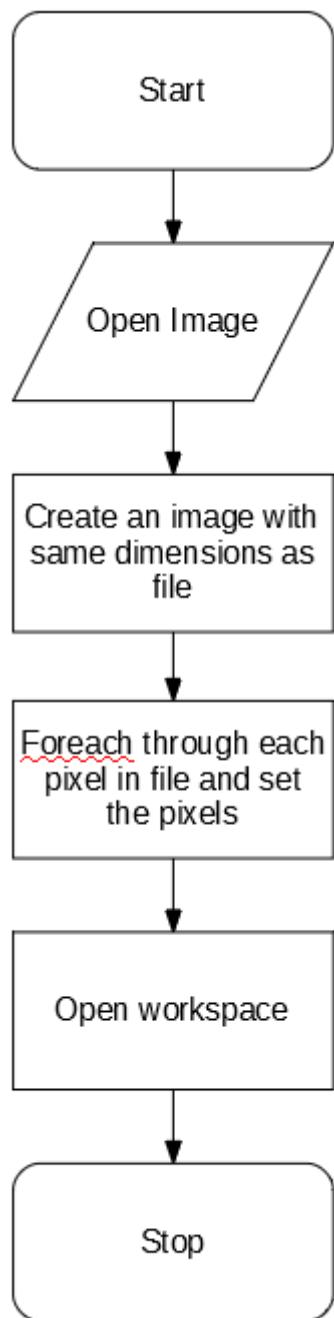
So, a few more buttons have been requested by the stakeholders:



Opening an existing SIMP file is reasonably simple, it can use the existing open file function designed before.

```
void BtnOpenClick(object sender, EventArgs e)
{
    DialogResult result = diaOpen.ShowDialog();
    if (result == DialogResult.OK) {
        using (FileStream stream = new FileStream(diaOpen.FileName, FileMode.Open)) {
            Workspace.OpenFile(stream);
        }
    }
}
```

However importing an image will need to be designed slightly differently



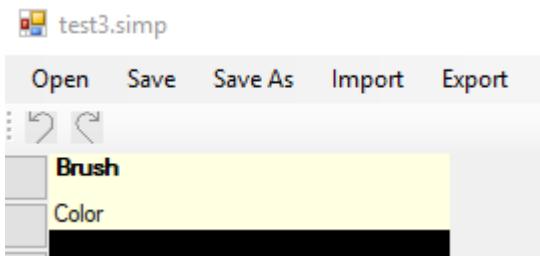
After doing this, the start page is fully functional.

## 14/12/2019 Improvements to saving

---

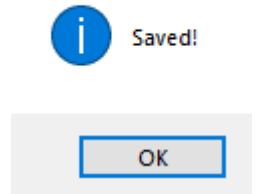
### Implementing Save As

The top bar currently only contains a Save button, which always makes a new file. This is not always useful if the user wants to save updated changes. The top bar has been updated to now include Save and Save As buttons:



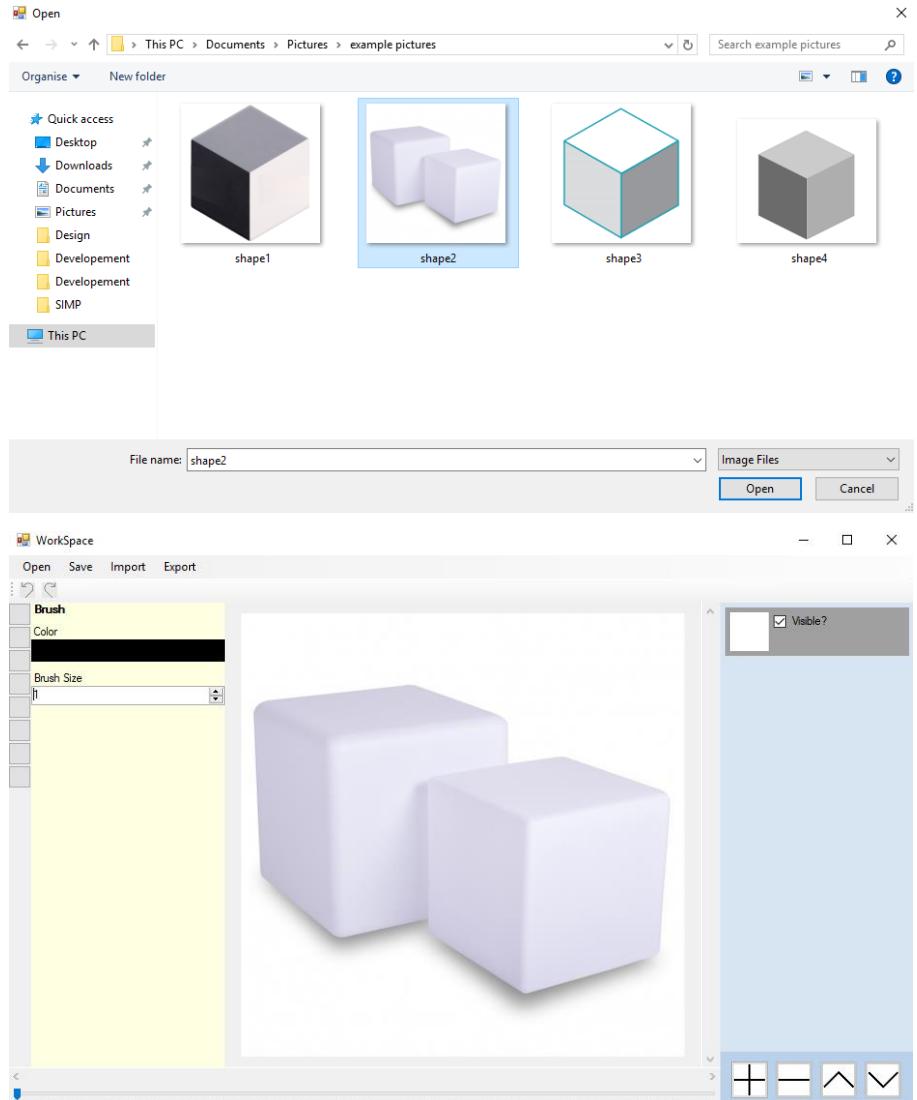
Notification boxes have also been added upon saving or exporting an image:

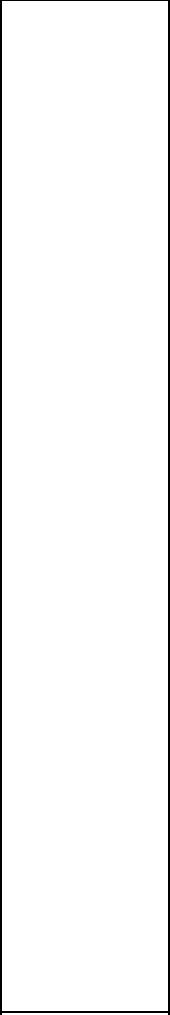
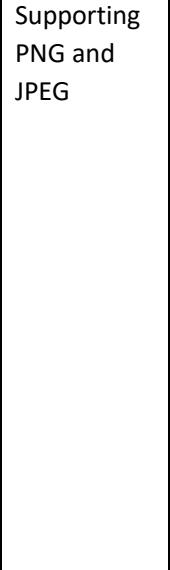
Success! 

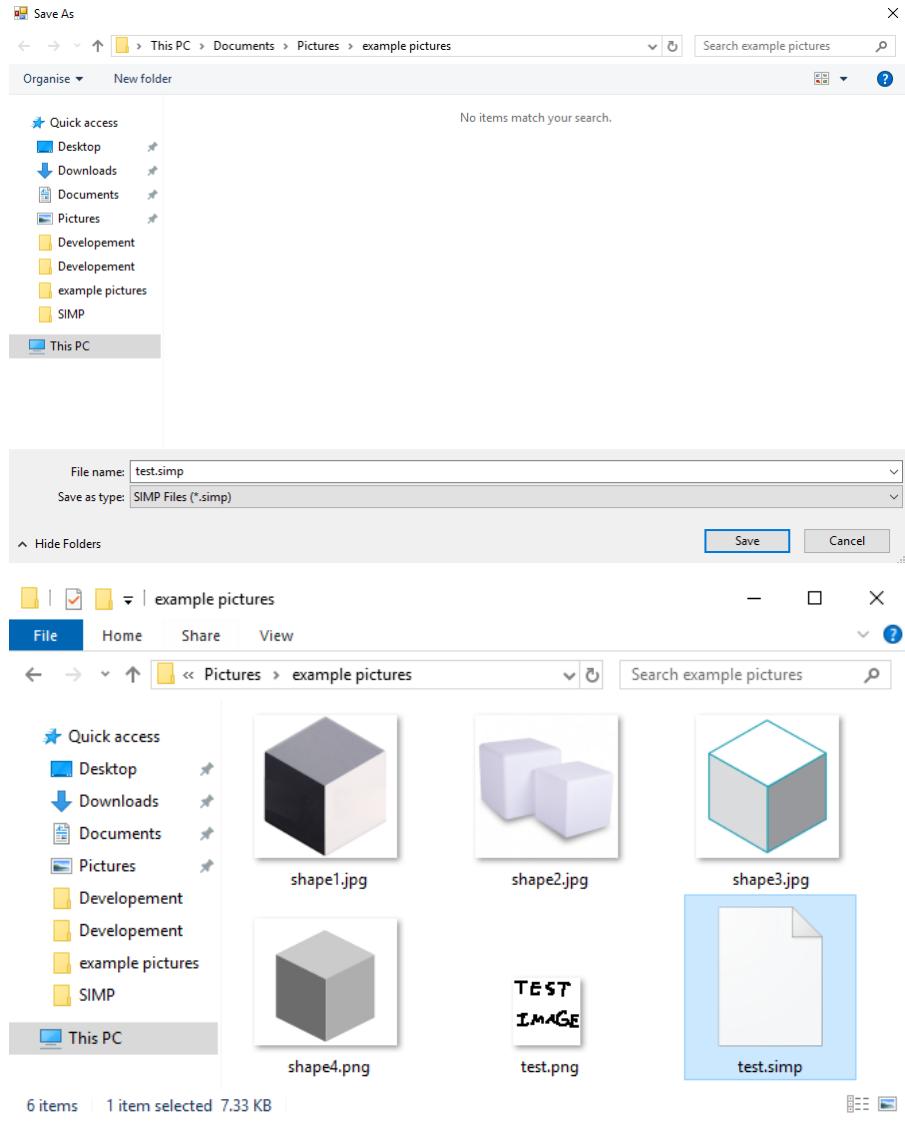


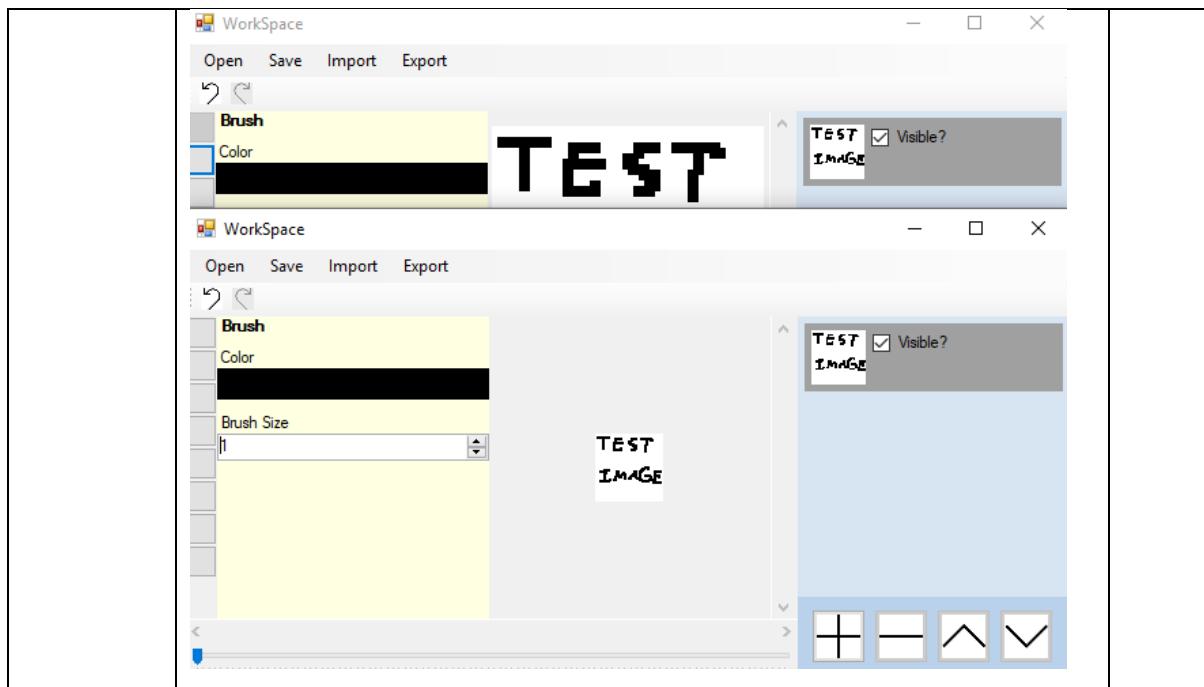
## 4.2.1 Success Criteria Evaluation

After these developments, Section 4 is now complete. The success criteria can now be evaluated

Feature	Proof	Code
Section C – File System		
Importing images	<p>Screenshot of the file browser showing an image preview, and screenshot showing the image in the program</p> 	C2
Exporting images	<p>Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder</p>	C3

	<p>Screenshot showing the file browser which accepts both PNG and JPEG images</p> <p>Supporting PNG and JPEG</p>	C4
	<p>Screenshot showing the file browser which accepts both PNG and JPEG images</p> <p>Supporting PNG and JPEG</p>	C4

Saving and loading from a proprietary format	<p>Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded</p>  <p>The screenshot illustrates a workflow for saving and loading images. At the top, a 'Save As' dialog box is open, showing the file name 'test.simp' and the save type 'SIMP Files (*.simp)'. Below it, a file browser window displays a folder structure under 'example pictures'. Inside this folder are several image files: 'shape1.jpg', 'shape2.jpg', 'shape3.jpg', 'shape4.png', and 'test.png'. The file 'test.png' contains the text 'TEST IMAGE'. The file 'test.simp' is highlighted with a blue selection box. The status bar at the bottom of the browser window indicates '6 items 1 item selected 7.33 KB'.</p>	C5
--	--	----



Not completed

Completed

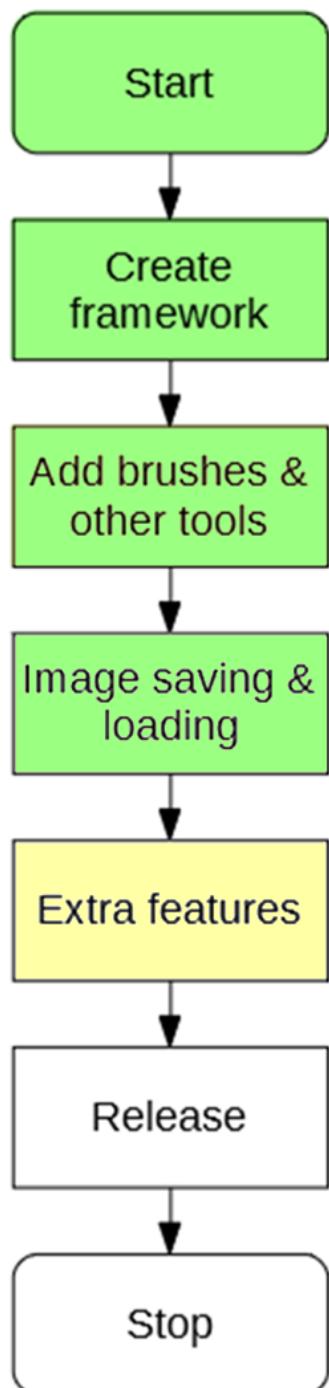
Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering "FF0000" (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5

Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text "Hello World" on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11
<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
<b>Section C – File System</b>		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
<b>Section D – Usability</b>		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders "Did you encounter any errors while using the program?"	D1
Program should be easy to use	75% yes response to asking stakeholders "Did you find the program easy to use?"	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

# Section 5 – Extra Features

## 5.1 Design

This section will contain the systems needed to save, load, import and export images. This will be able to export the image created by the previous section:



## 5.1.1 Success Criteria Fulfilment Plan

In this section the following success criteria are planned to be completed:

Not completed

To be done this section

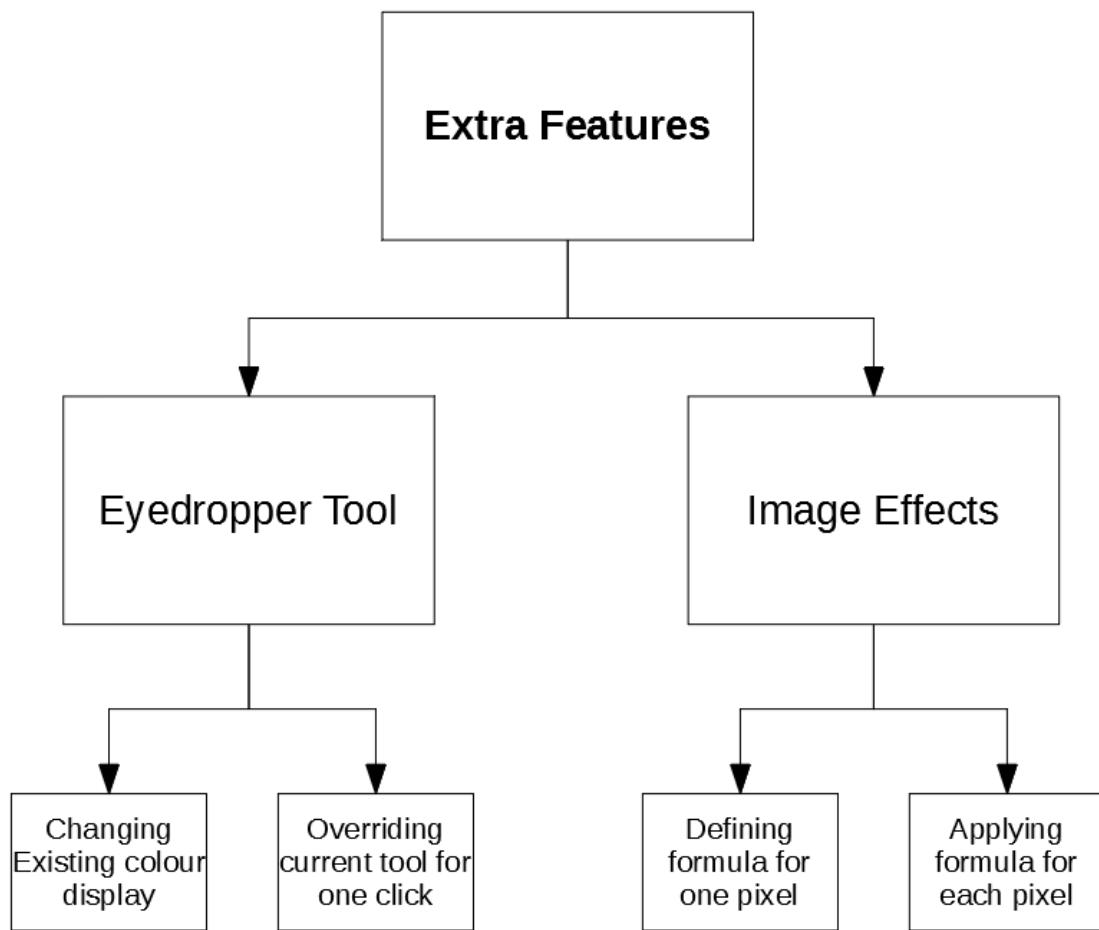
Completed

Feature	Proof	Code
Section A - Brushes		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6
Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering "FF0000" (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text "Hello World" on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11

<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5
Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

## 5.1.2 Decomposition

---

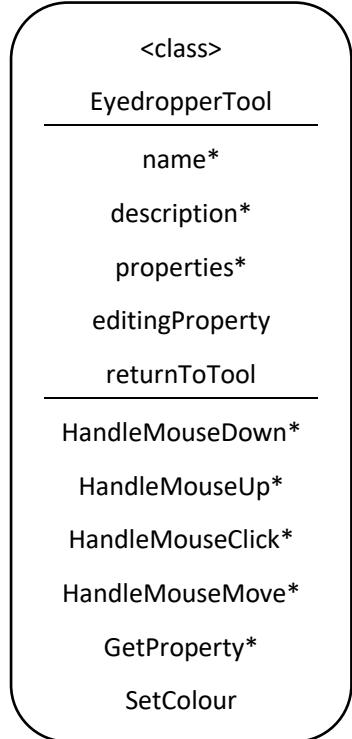


## 5.1.3 Class Design

### 5.1.3.1 EyedropperTool

The EyeDropperTool will be a special case, as it is not a normally accessible tool on the sidebar.

Class Diagram



Properties

Property	Datatype	Justification
name	String	Inherited from ITool
description	String	Inherited from ITool
properties	List of ToolProperties	Inherited from ITool
editingProperty	ColorProperty	The property that the eyedropper tool has been set to change
ReturnToTool	ITool	The tool that was selected before the eyedropper was

Methods

Method	Params	Return type	Justification
HandleMouseDown	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseUp	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseClick	FilePoint clickLocation, MouseButton button	None	Inherited from ITool

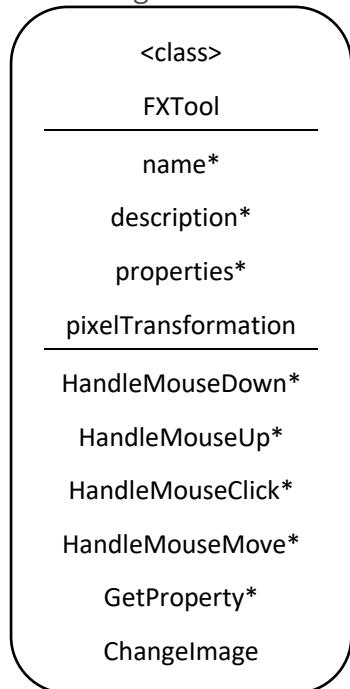
HandleMouseMove	FilePoint oldLocation, FilePoint newLocation	None	Inherited from ITool
GetProperty	String propertyName	ToolProperty	Inherited from ITool
SetColour	ColorProperty property	None	When complete, will set the respective colourproperty to the new colour.

### 5.1.3.2 FXTool

This tool will be responsible for the image effects and will contain the effects of:

- Grayscale
- Black & White
- Invert

Class Diagram



Properties

Property	Datatype	Justification
name	String	Inherited from ITool
description	String	Inherited from ITool
properties	List of ToolProperties	Inherited from ITool
pixelTransformation	<a href="#">PixelTransformation</a>	A formula for how each pixel will be changed

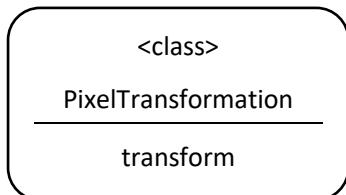
Methods

Method	Params	Return type	Justification
HandleMouseDown	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseUp	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseClick	FilePoint clickLocation, MouseButton button	None	Inherited from ITool

HandleMouseMove	FilePoint oldLocation, FilePoint newLocation	None	Inherited from ITool
GetProperty	String propertyName	ToolProperty	Inherited from ITool
ChangeImage	None	None	Applies the current pixelTransformation onto each pixel in the image

### 5.1.3.3 PixelTransformation

This class will contain a **delegate method** for what transformation should be applied to each pixel. The delegate is included in a class to impose proper encapsulation and readability.



#### Properties

Property	Datatype	Justification
transform	Delegate: params: (colour) returns: (color)	Contains the algorithm to apply to each pixel in the image

#### Static Properties

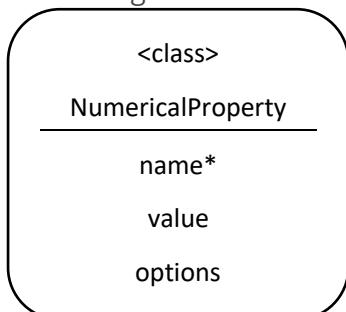
To access these transformations, the PixelTransformation class will include a few static methods that return each sort of transformation:

Static Property	Datatype	Justification
Grayscale	PixelProperty	Contains the algorithm to convert a colour into grayscale
BlackAndWhite	PixelProperty	Contains the algorithm to convert a colour to black & white
Invert	PixelProperty	Contains the algorithm to invert a colour

### 5.1.3.4 ComboProperty

The user will need a way to select what image transformation they want to apply. This needs a new sort of property – one where an option can be selected from a few choices.

#### Class Diagram



#### Properties

Property	Datatype	Justification
name*	String	Inherited from IToolProperty
value	String	Stores the value of the property

options	Array of String	Stores the possible options to choose from.
---------	-----------------	---

## 5.1.4 Algorithm Design

### Algorithm 5.1 Changing existing colour display

The current display for a colour looks like:



This should be changed to look like:



Where the eyedropper button initiates the tool.

### Algorithm 5.2 Overriding current tool for one click

When created, the `ReturnToTool` property of the `EyeDropper` tool can be set to the tool that was selected. Then, upon clicking:

```
HandleMouseClick(clickLocation) {  
    colour = image.GetColour(clickLocation)  
    editingProperty.value = colour  
    workspace.currentTool = returnToTool  
    workspace.displayTool() } // Redisplays the tool as the colour has changed
```

Redisplays the tool as the colour has changed

## Algorithm 5.1 &amp; Algorithm 5.2 Unit Test

Test	ID	Expected Result	Comment
<i>Opening a tool with a color property</i>	1	Color has option to use the eyedropper tool on it.	This makes sure that the eyedropper tool button is correctly visible.
<i>Pressing the eyedropper button</i>	2	Button indicates that it is activated and eyedropper is active	This makes sure that the user is aware the tool is active
<i>Selecting a point on the image</i>	3	Color is changed to the color at the point pressed and eyedropper deselects	This is to make sure that the cycle is completed fully
<i>Selecting the eyedropper tool again</i>	4	The same process can be done, same as first time eyedropper is pressed	This makes sure the eyedropper tool is reset correctly
<i>Selecting a point not on the image</i>	5	The tool is still active and no colour is changed	Tests that the program handles clicking at a point not on the image
<i>Pressing the eyedropper button again whilst it is selected</i>	6	The tool deactivates and normal execution resumes	Tests that the eyedropper can be deselected if user wants to
<i>Selecting another tool with eyedropper active</i>	7	Eyedropper deactivates and new tool is selected	Tests that the program handles deselecting eyedropper when a different tool is selected
<i>Selecting a transparent pixel</i>	8	White is returned instead	Tests that the eyedropper returns the correct displayed colour as white is displayed at completely transparent points

### Algorithm 5.3 Defining formula for one pixel

In order to do this, formulas must be defined for the three planned transformations:

- Grayscale
- Black & White
- Invert

#### Algorithm 5.3A Grayscale

This can be achieved by finding the average of the three colours:

```
Grayscale(colour) {
    total = colour.R + colour.G + colour.B
    average = total / 3
    return new Color(average,average,average)
}
```

R, G and B all use the same average value

#### Algorithm 5.3B Black & White

This can be achieved by finding the average, and checking if it is above a certain threshold:

```
BlackAndWhite(colour) {
    total = colour.R + colour.G + colour.B
    average = total / 3
    IF average < 127 THEN
        return White
    ELSE
        return Black
    END IF
}
```

#### Algorithm 5.3C Invert

This can be achieved by subtract each of R, G and B from 255 (the maximum)

```
Invert(colour) {
    newR = 255 - colour.R
    newG = 255 - colour.G
    newB = 255 - colour.B
    return new Colour(newR,newG,newB)
}
```

### Algorithm 5.4 Applying formula for each pixel

To then apply the formula for each pixel, the image can be iterated through and the current transformation's algorithm will execute on each pixel:

```
ChangeImage() {
    FOR x = 0 TO width
        FOR y = 0 TO height
            oldColor = image.GetPixel(x,y)
            newColor = currentTransformation.transform(oldColor)
            image.SetPixel(x,y,newColor)
        NEXT
    NEXT
}
```

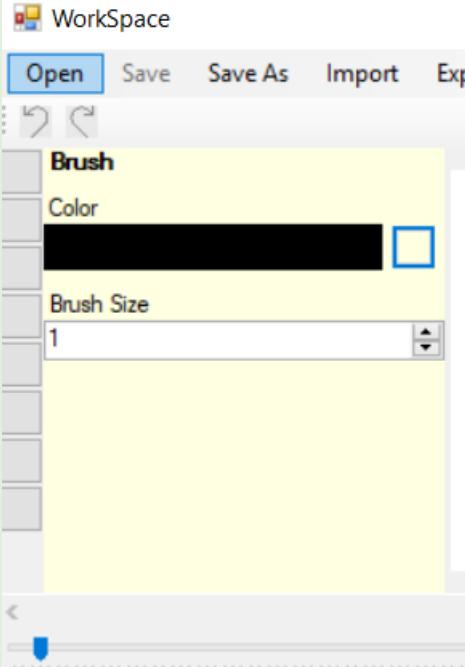
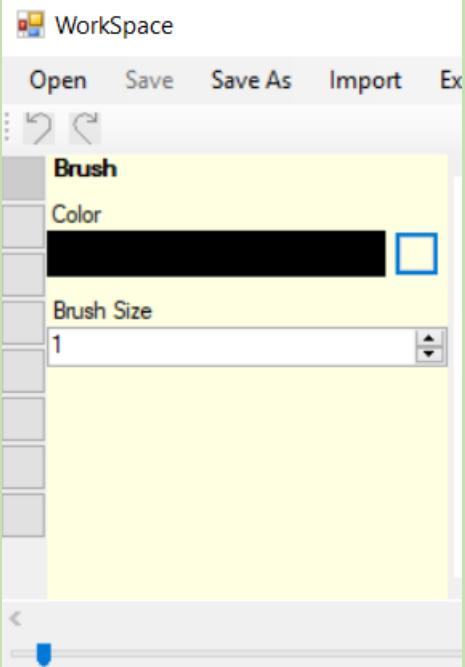
### Algorithm 5.3 & Algorithm 5.4 Unit Test

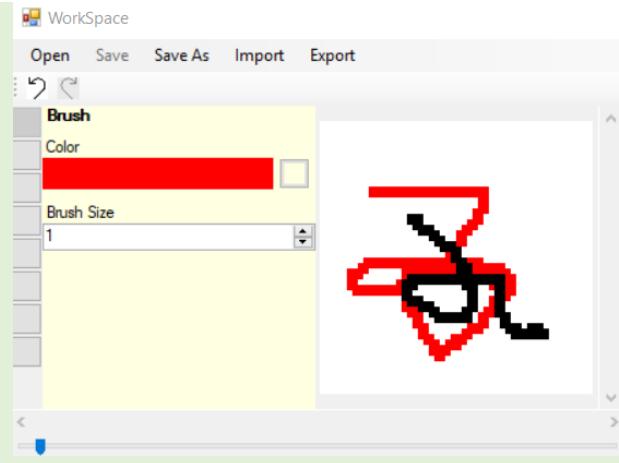
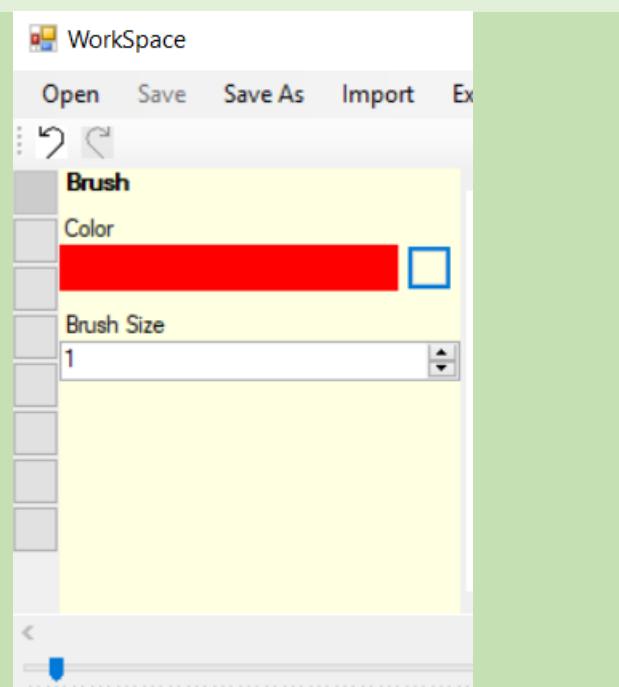
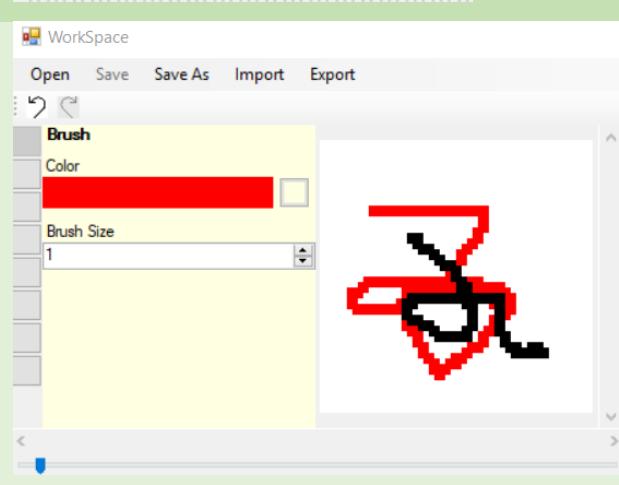
	<i>Test ID</i>	<i>Expected Result</i>	<i>Comment</i>
<i>Apply grayscale transformation</i>	1	The image becomes shades of grey	Tests that the grayscale transformation functions correctly
<i>Apply black &amp; white transformation</i>	2	The image becomes black and white	Tests that black and white transformation functions correctly
<i>Apply invert transformation</i>	3	The colours of the image invert (e.g. green -> purple)	Tests that the invert transformation functions correctly
<i>Apply invert transformation twice</i>	4	The colours return to normal	Tests that the invert is reversible

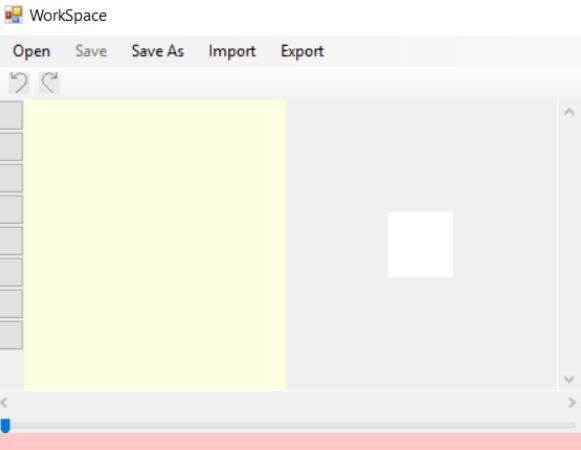
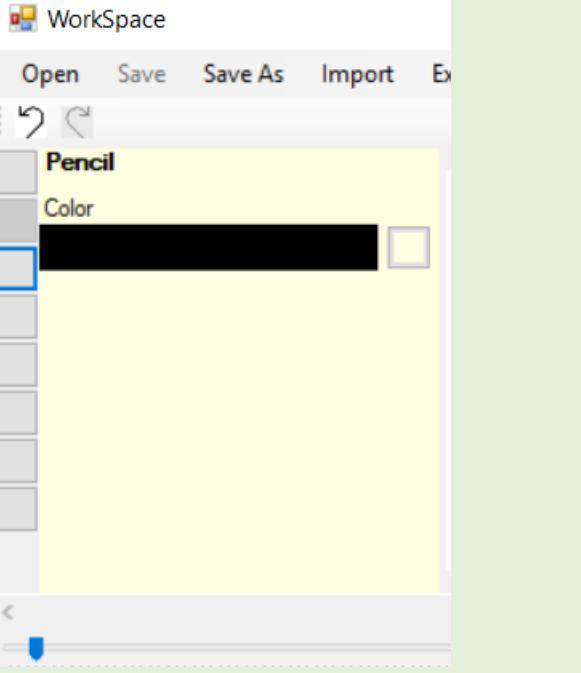
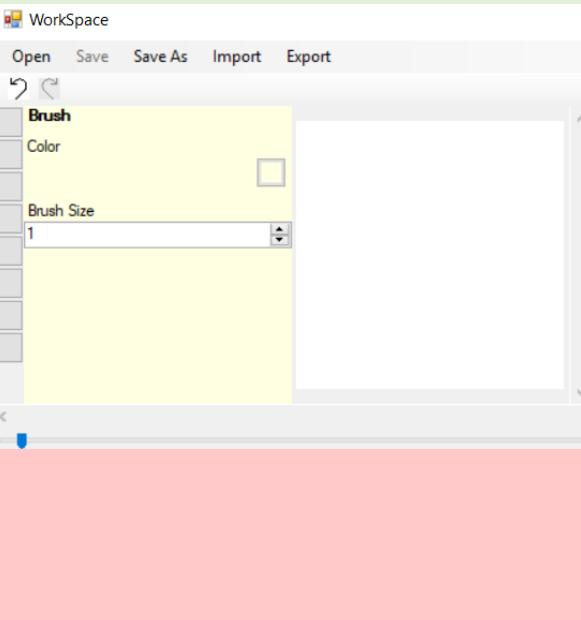
## 5.2 Development

19/12/2019 Adding Eyedropper

Algorithm 5.1 & Algorithm 5.2 Unit Test

Test	ID	Expected Result	Actual Result	Comment
<i>Opening a tool with a color property</i>	1	Color has option to use the eyedropper tool on it.		This makes sure that the eyedropper tool button is correctly visible.
<i>Pressing the eyedropper button</i>	2	Button indicates that it is activated and eyedropper is active		This makes sure that the user is aware the tool is active

<p><i>Selecting a point on the image</i></p>	<p>3 Color is changed to the color at the point pressed and eyedropper deselects</p>		<p>This is to make sure that the cycle is completed fully</p>
<p><i>Selecting the eyedropper tool again</i></p>	<p>4 The same process can be done, same as first time eyedropper is pressed</p>		<p>This makes sure the eyedropper tool is reset correctly</p>
<p><i>Selecting a point not on the image</i></p>	<p>5 The tool is still active and no colour is changed</p>		<p>Tests that the program handles clicking at a point not on the image</p>

<i>Pressing the eyedropper button again whilst it is selected</i>	<b>6</b> The tool deactivates and normal execution resumes		The eyedropper tool is selected again, causing a crash when trying to return to the previous tool
<i>Selecting another tool with eyedropper active</i>	<b>7</b> Eyedropper deactivates and new tool is selected		Tests that the program handles deselecting eyedropper when a different tool is selected
<i>Selecting a transparent pixel</i>	<b>8</b> White is returned instead		This case has not been handled, so returns a transparent colour

## Fixing Error #6

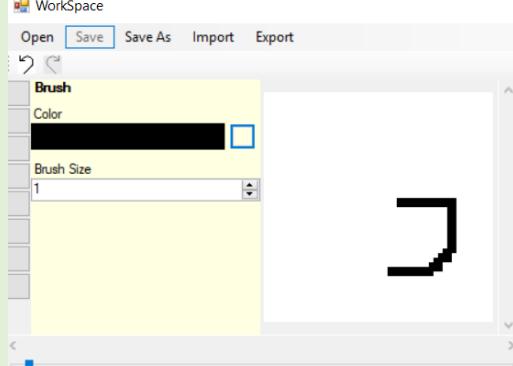
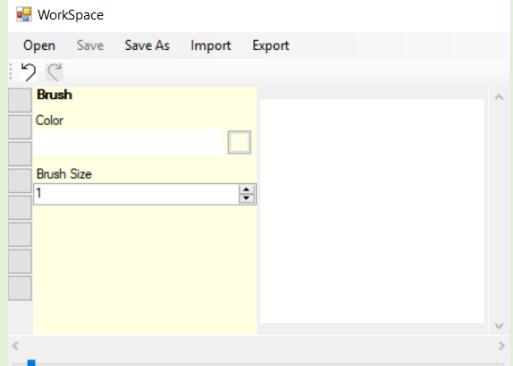
The error is caused by the program not checking whether the eyedropper is currently selected. It can be fixed by adding this check:

```
if (currentTool is EyedropperTool) {
    currentTool = ((EyedropperTool)currentTool).returnToTool;
} else {
    EyedropperTool newTool = new EyedropperTool("", "", this, (ColorProperty)((Button)sender).Tag, currentTool);
    currentTool = newTool;
}
```

## Fixing Error #8

This can be fixed with a single line, replacing the colour with white if it is transparent

```
color = (color == Color.Transparent) ? Color.White : color;
```

Test	ID	Expected Result	Actual Result	Comment
<i>Pressing the eyedropper button again whilst it is selected</i>	6	The tool deactivates and normal execution resumes		The eyedropper tool now deselects when pressed again
<i>Selecting a transparent pixel</i>	8	White is returned instead		The new line returns white

## 20/12/2019 Adding FX

---

### Implementing Delegate System

The delegate system is implemented as a part of the PixelTransformation class. It is defined, then PixelTransformation is given it as a property:

```
public class PixelTransformation
{
    public delegate Color Transformation(Color input);
    public Transformation transform;
```

Then, when creating the static members, the transform member can be set to the necessary code:

```
public static PixelTransformation GrayScale {
    get {
        PixelTransformation transformation = new PixelTransformation();
        transformation.transform = delegate(Color input) {
            return Color.White;
        };
        return transformation;
    }
}
```

In this case, the colour is returned as white always. Thus when applying this transformation the code is very simple, the delegate is called and passed the colour it needs:

```
for (int x = 0; x < myWorkspace.image.fileWidth; x++) {
    for (int y = 0; y < myWorkspace.image.fileHeight; y++) {
        Color oldColor = myWorkspace.image.GetPixel(x,y);
        Color newColor = currentTransformation.transform(oldColor);
        myWorkspace.image.SetPixel(x,y,newColor);
    }
}
```

This closely follows the previously laid out design:

```
ChangeImage() {
    FOR x = 0 TO width
        FOR y = 0 TO height
            oldColor = image.GetPixel(x,y)
            newColor = currentTransformation.transform(oldColor)
            image.SetPixel(x,y,newColor)
        NEXT
    NEXT
}
```

## Implementing Grayscale

The grayscale can be implemented in accordance to the designed algorithm:

```
Grayscale(colour) {
    total = colour.R + colour.G + colour.B
    average = total / 3
    return new Color(average,average,average)
}

transformation.transform = delegate(Color input) {
    int total = input.R + input.G + input.B;
    int average = total / 3;
    return Color.FromArgb(average,average,average);
};
```

return new Color(average,average,average)

R, G and B all use the same average value

## Implementing Black & White

```
BlackAndWhite(colour) {
    total = colour.R + colour.G + colour.B
    average = total / 3
    IF average < 127 THEN
        return White
    ELSE
        return Black
    END IF
}
```

However, this algorithm can be improved as the total does not need to be divided by 3, the total can be compared to  $127 * 3$ :

```
transformation.transform = delegate(Color input) {
    int total = input.R + input.G + input.B;
    if (total < 381) {
        return Color.Black;
    } else {
        return Color.White;
    }
};
```

## Implementing Invert

```
Invert(colour) {
    newR = 255 - colour.R
    newG = 255 - colour.G
    newB = 255 - colour.B
    return new Colour(newR,newG,newB)
}
```

## 17/01/2020 Final Changes

---

### Transparency Saving error

While doing Beta Testing with clients, an error was discovered that was not found in Alpha Testing. When saving layers, the Red, Green and Blue values were saved, meaning that transparency was **not** saved. This should be resolved so that transparency is saved.

So the saving code has been changed to save the alpha (transparency) channel.

```
currentColor = layer.pixels[x,y].Color;
stream.WriteByte(currentColor.R);
stream.WriteByte(currentColor.G);
stream.WriteByte(currentColor.B);
stream.WriteByte(currentColor.A);
```

However this means that the file format has been changed during beta testing – however my clients would like their older SIMP files to remain compatible.

### Introducing File Versioning

Now that the file saving has been changed, the header has also been changed to denote that it is a different type of SIMP file. (SIM2 = SIMP2)

```
//SIMP check digits
stream.WriteByte((byte)'S');
stream.WriteByte((byte)'I');
stream.WriteByte((byte)'M');
stream.WriteByte((byte)'2');
```

Then, when loading, the image, there is a switch case to decide what sort of file it is:

```
switch (checkString) {
    case "SIMP": //simp format v1 loading

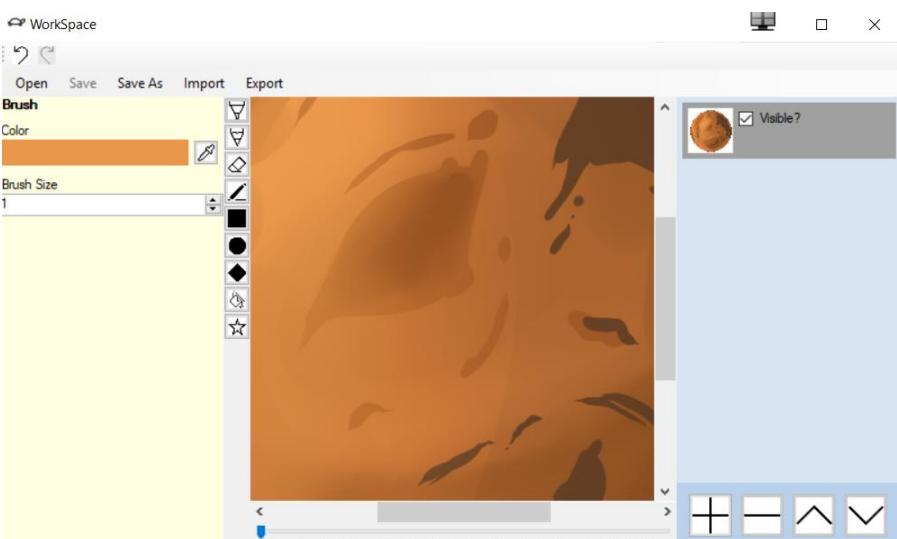
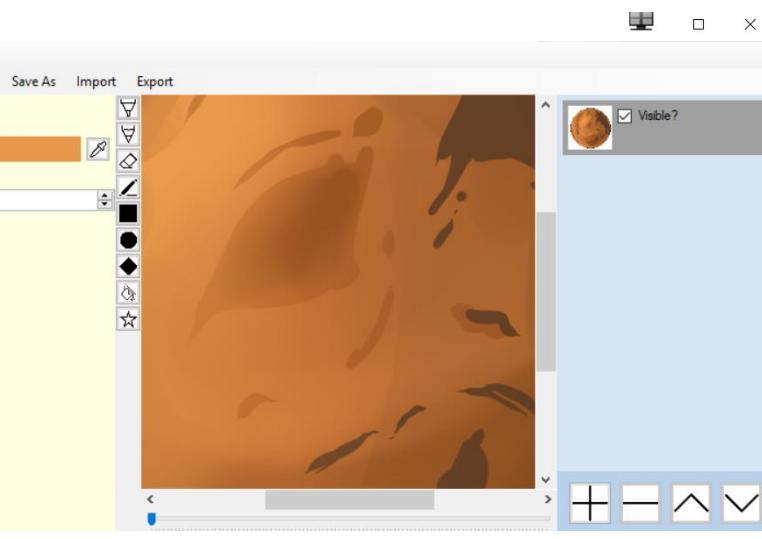
    case "SIM2": //simp format v2 loading

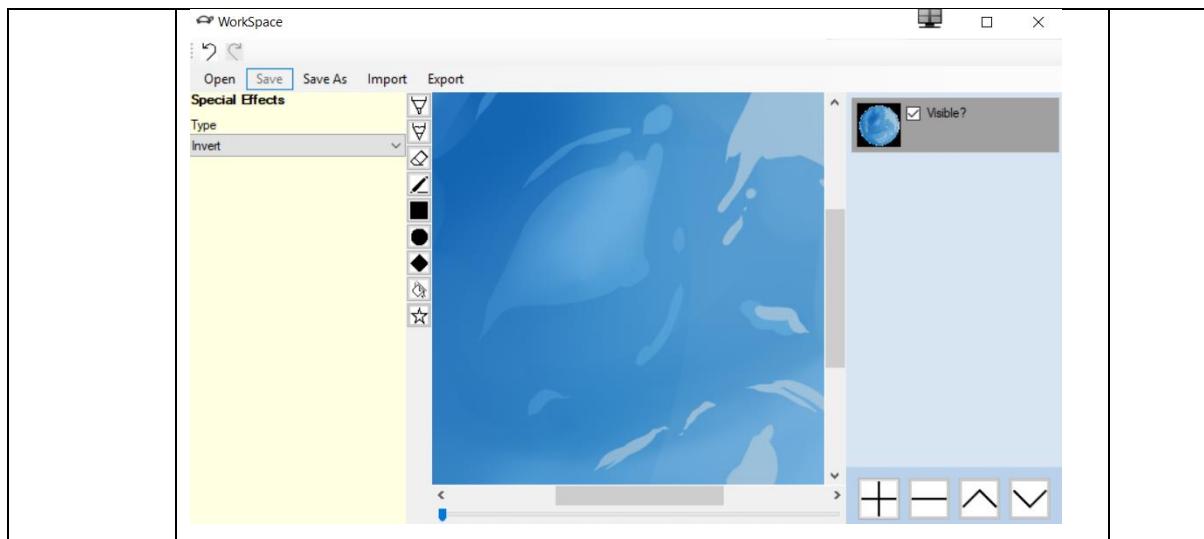
    default:
        MessageBox.Show("File cannot be loaded. \n\nSIMP data could not be found within this file.");
        return null;
```

Then, in the SIM2 loading, the alpha channel is loaded.

```
R = SafeRead(stream);
G = SafeRead(stream);
B = SafeRead(stream);
A = SafeRead(stream);
newLayer.pixels[x,y] = new SolidBrush(Color.FromArgb(A,R,G,B));
```

## Success Criteria Evaluation

Feature	Proof	Code
Section B – Other Editing Tools		
Eyedropper tool	<p>Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it</p> 	B11
Image effects	<p>Screenshot of an image before and after an effect is applied</p> 	B12



#### Section D – Usability

Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

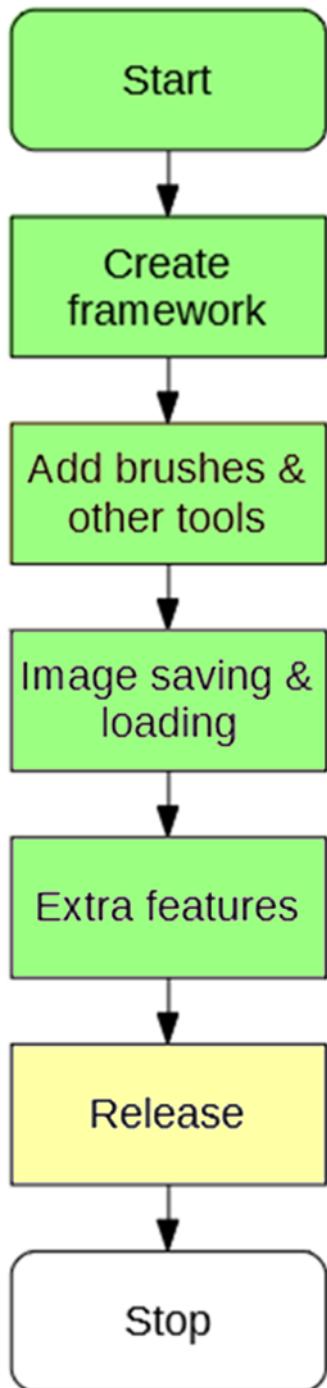
Feature	Proof	Code
<b>Section A - Brushes</b>		
Variable brush width	Screenshot of strokes of the same brush showing different widths	A1
Hard brushes	Screenshot showing the hard edge of the brush (colour to no colour)	A2
Shape creation tools	Screenshot showing the shape toolbar and a small selection of drawn shapes	A3
Fill (bucket) tool	Screenshot showing a before and after of filling a large area	A4
Single pixel pencil	Screenshot showing a stroke of the single pixel brush	A5
Rubber	Screenshot showing a densely packed picture being rubbed out	A6

Section B – Other editing tools		
Image viewer	Screenshot of a currently being viewed image	B1
Bitmap image editor	Screenshot of a zoom in on the image showing the pixels	B2
RGB colour picker	Screenshot showing a system for entering an RGB colour	B3
RGB direct input	Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red	B4
Layer system	Screenshot of layer navigator	B5
Rectangle selection tool	Screenshot showing a rectangle selection on the image	B6
Magic selection tool	Screenshot showing a complex selection around non-linear shape	B7
Transparent pixels	Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required	B8
Zoom in (no zoom out)	Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller	B9
Text	Screenshot of the text “Hello World” on the image	B10
Eyedropper tool	Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it	B11
<i>Image effects</i>	<i>Screenshot of an image before and after an effect is applied</i>	B12
<i>Rotating Images</i>	<i>Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°</i>	B13
<i>Clipping masks</i>	<i>Screenshot of an image being clipped onto a complex selection</i>	B14
Section C – File System		
Creating a new image	Screenshot of a blank 300x300 square image	C1
Importing images	Screenshot of the file browser showing an image preview, and screenshot showing the image in the program	C2
Exporting images	Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder	C3
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images	C4
Saving and loading from a proprietary format	Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded	C5

Section D – Usability		
Program should be stable and not crash.	A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?”	D1
Program should be easy to use	75% yes response to asking stakeholders “Did you find the program easy to use?”	D2
Features should be easily accessible	From the default state of the program, any feature will need to be activated by no less than 4 clicks	D3

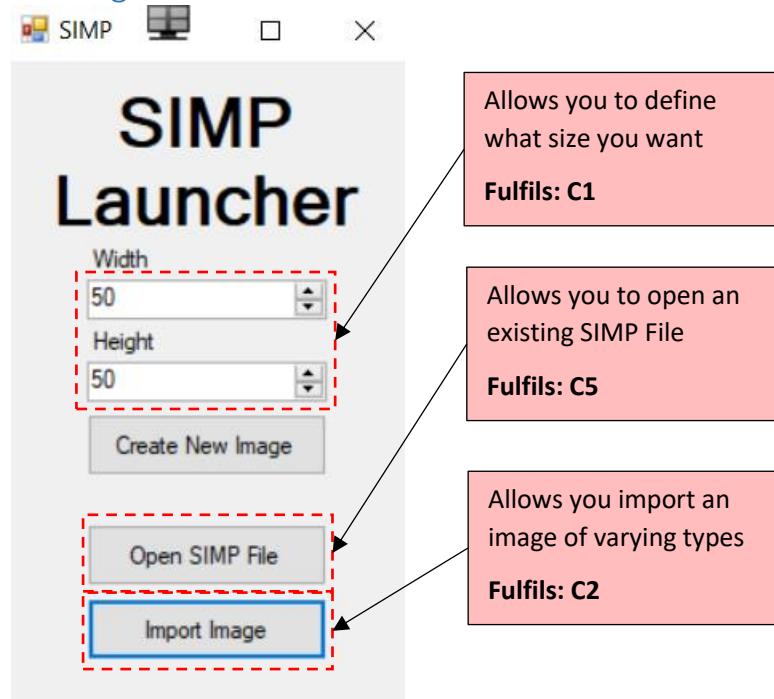
# Section 6 – Evaluation

SIMP is now finished, there will now be final evaluation on the program

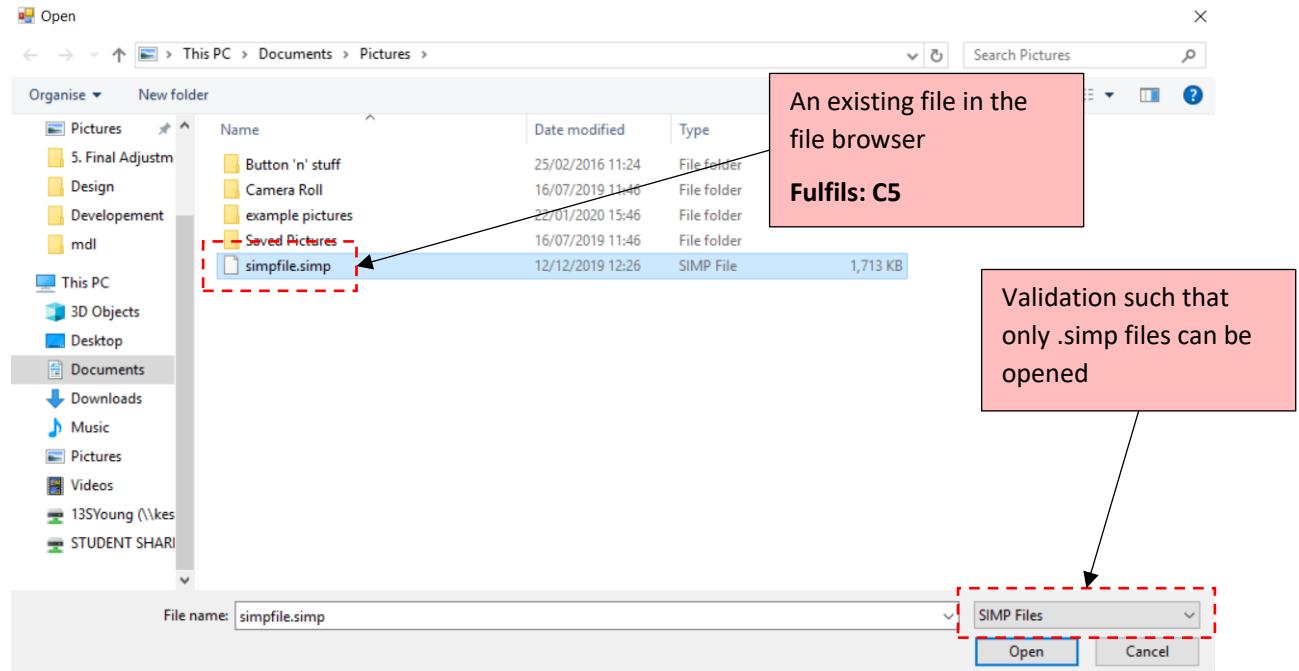


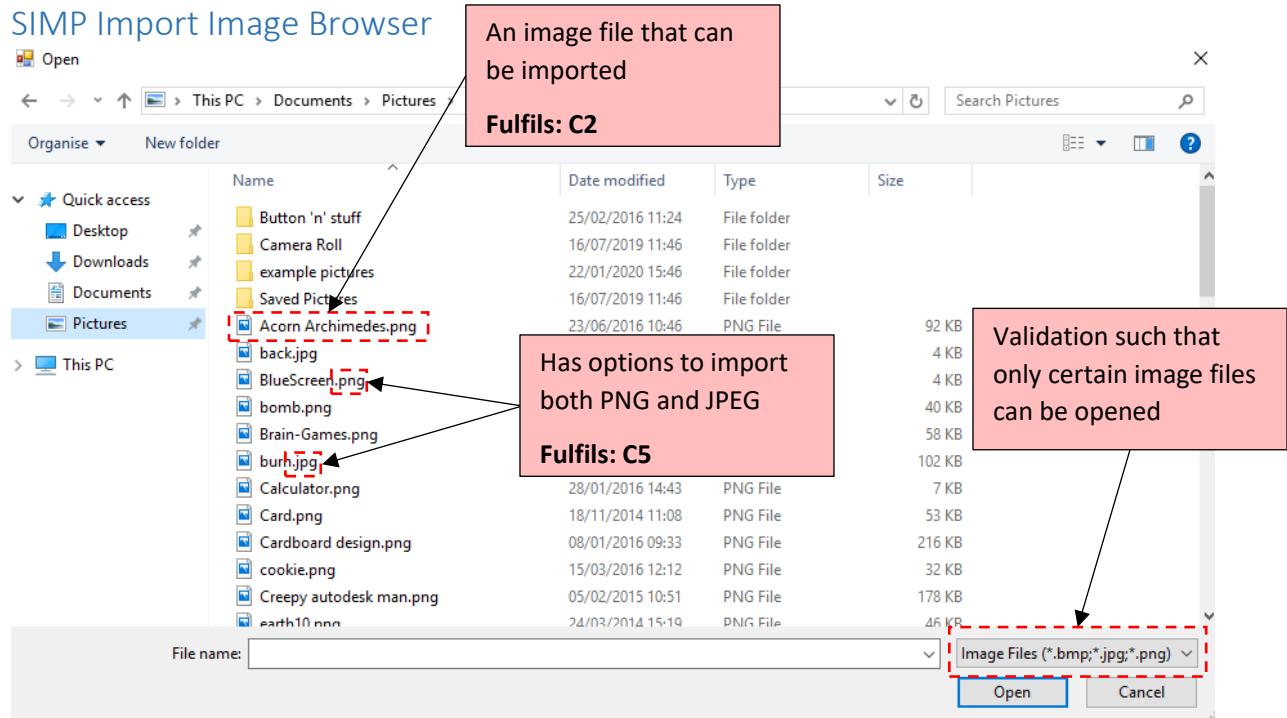
## 6.1 Criteria Fulfilment

### Starting Window

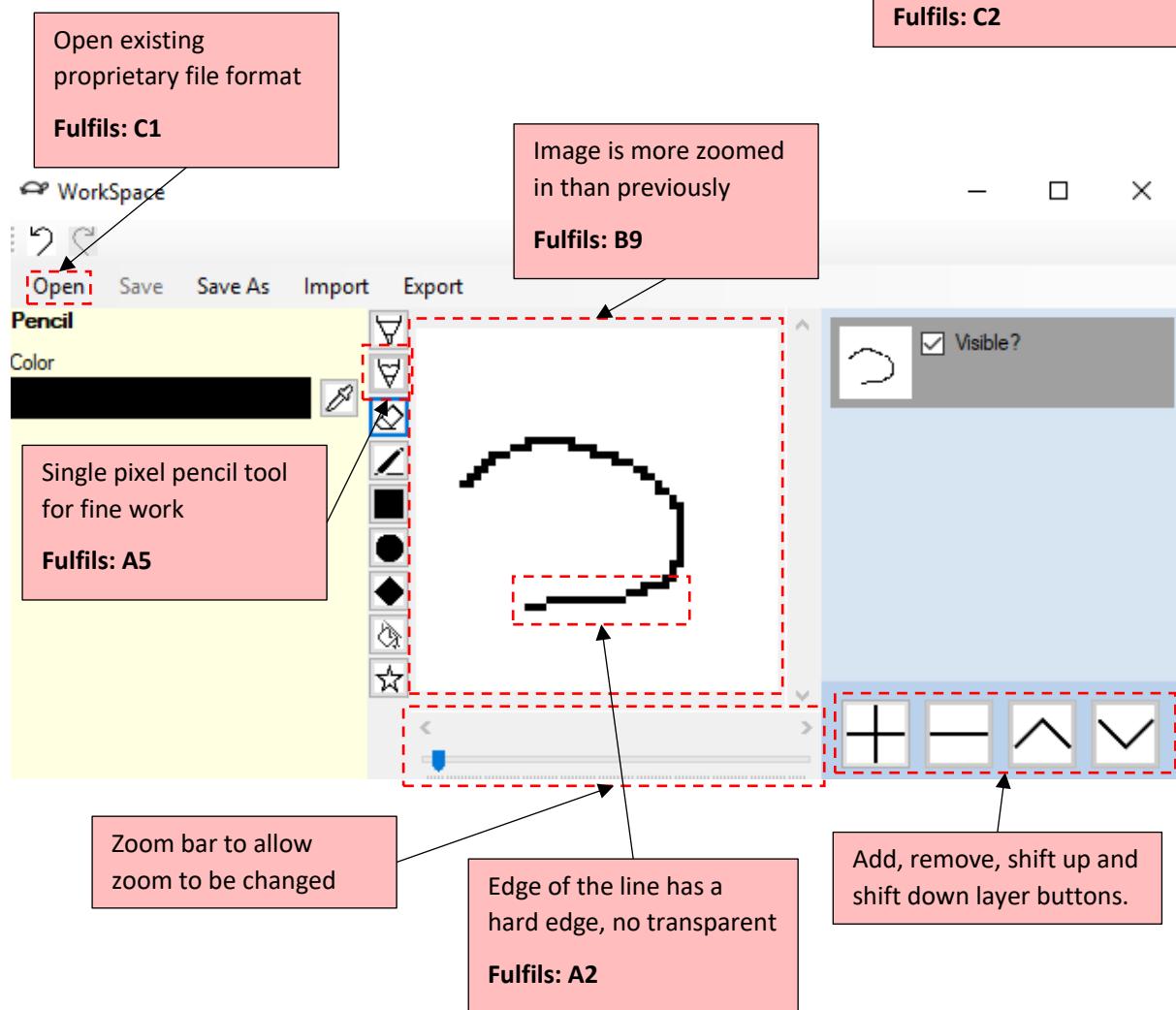
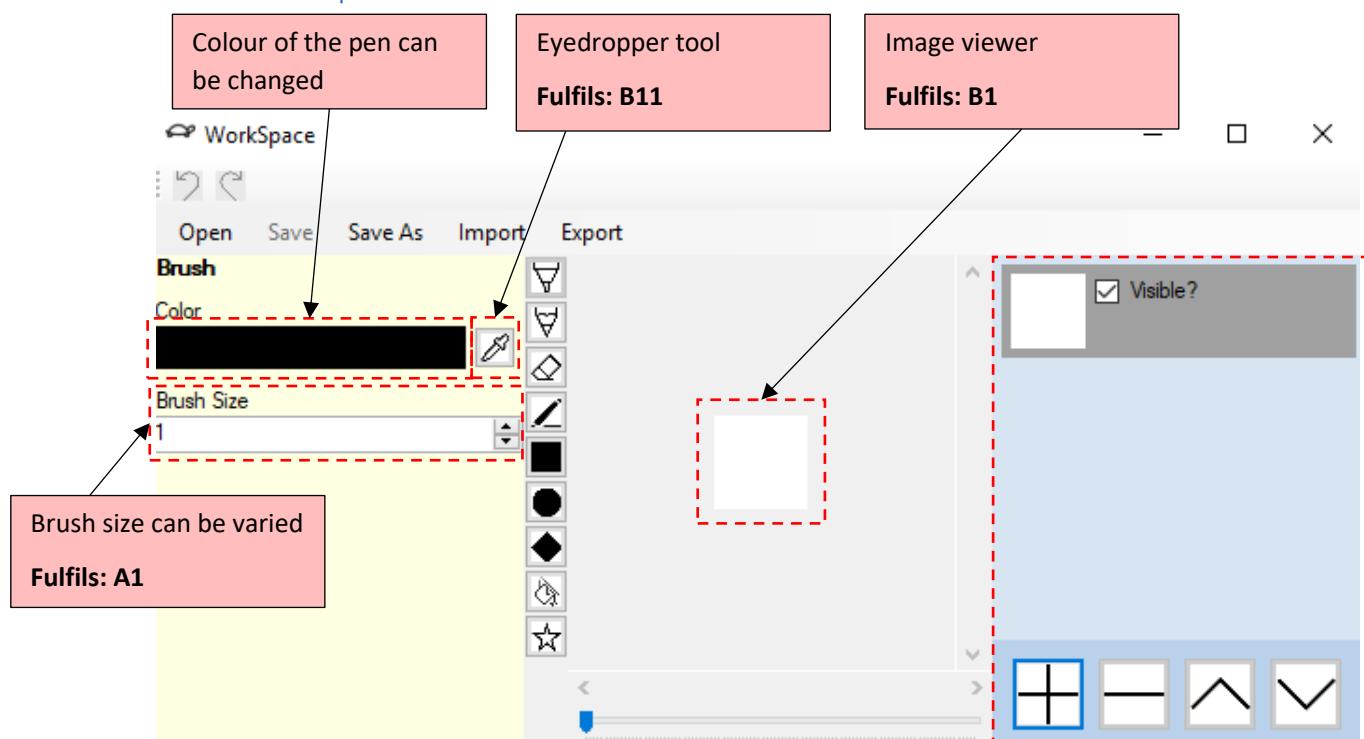


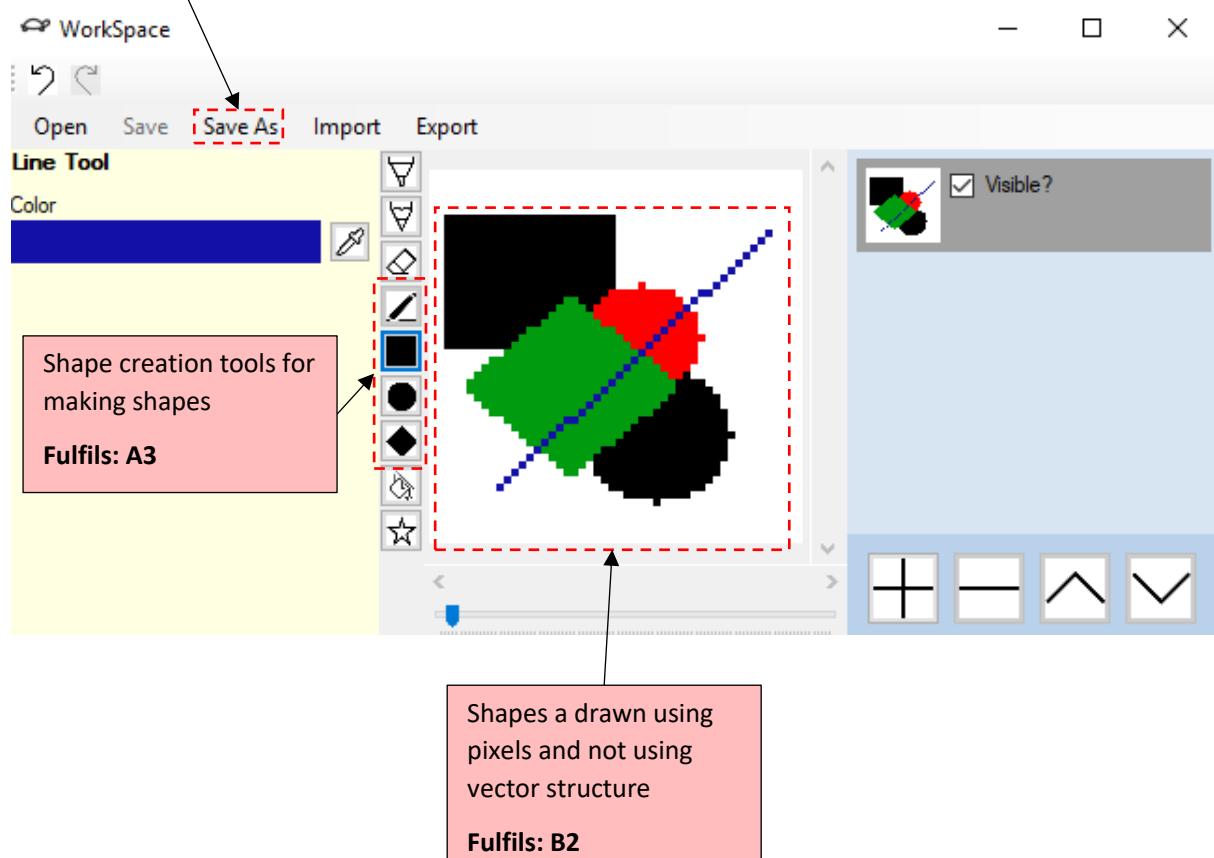
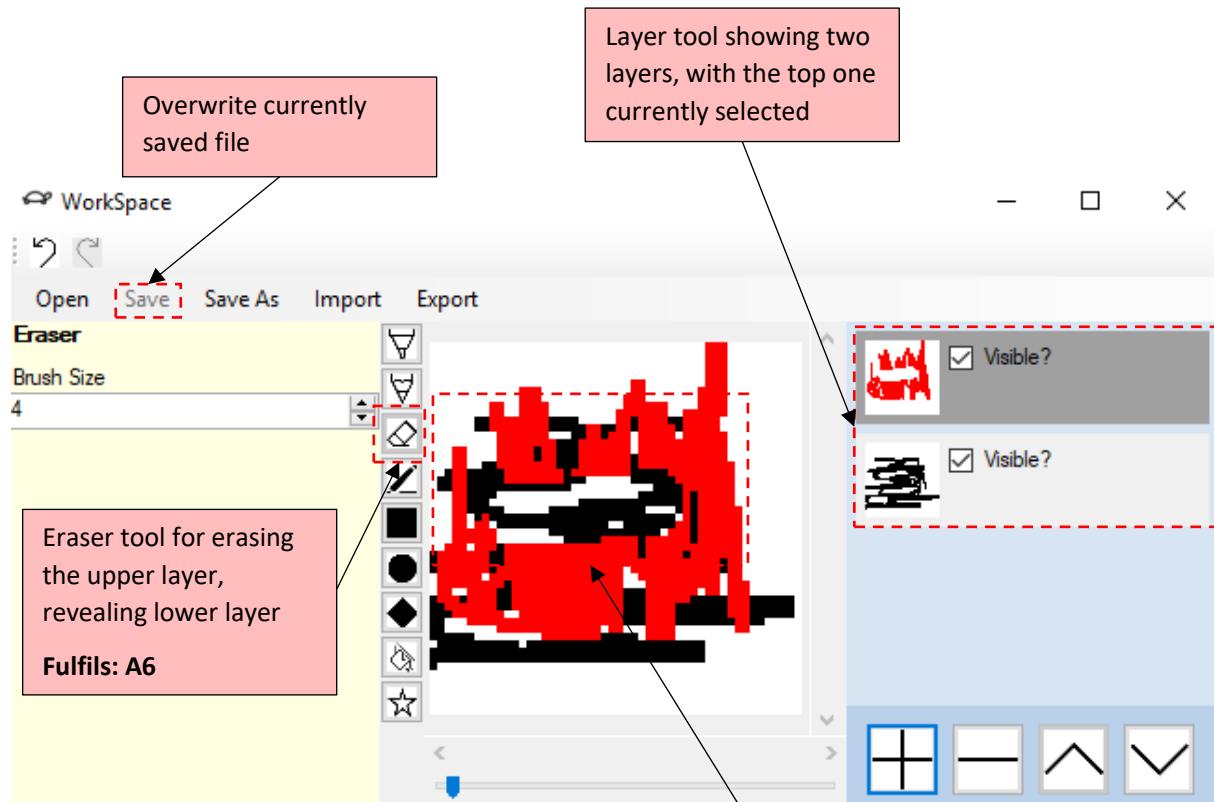
### SIMP Open File Browser

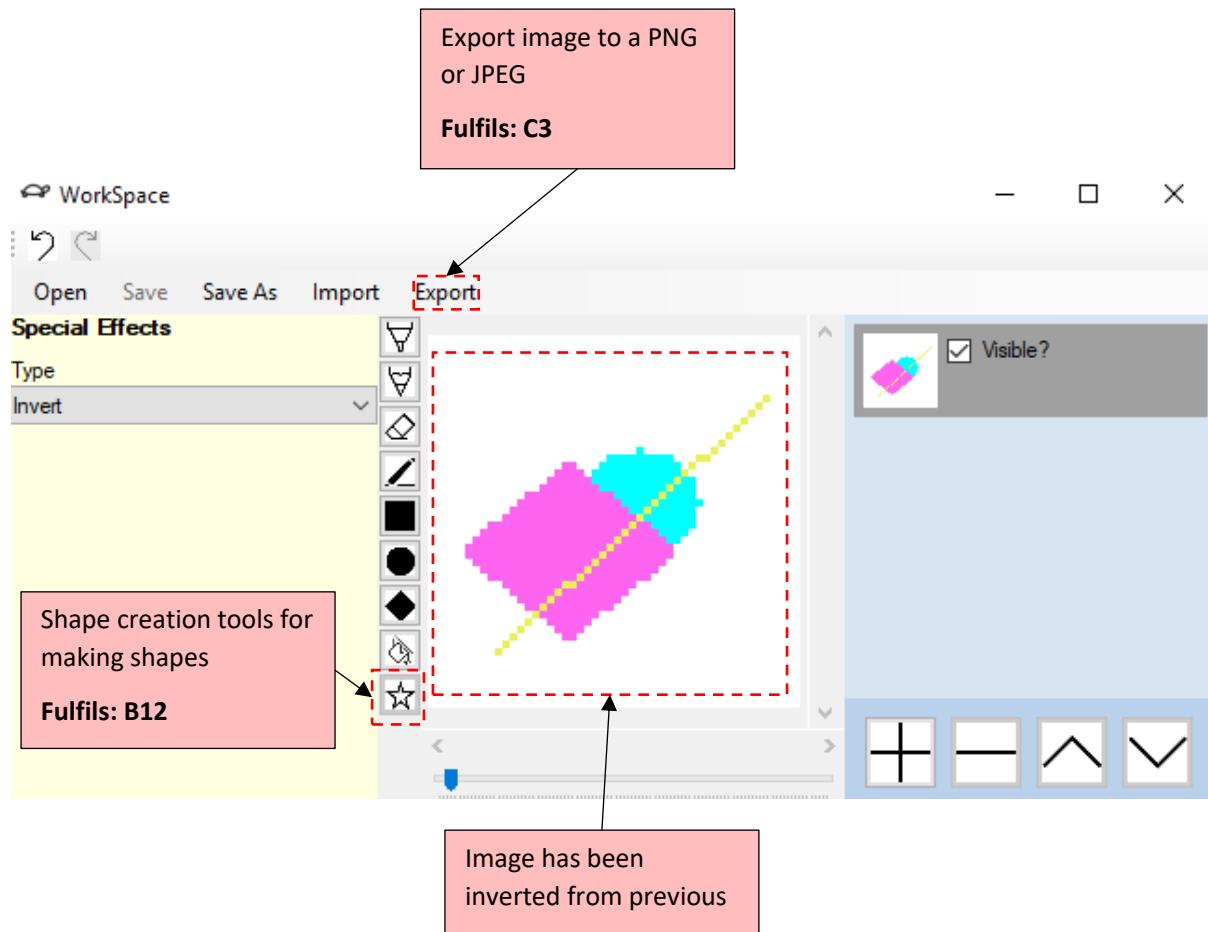
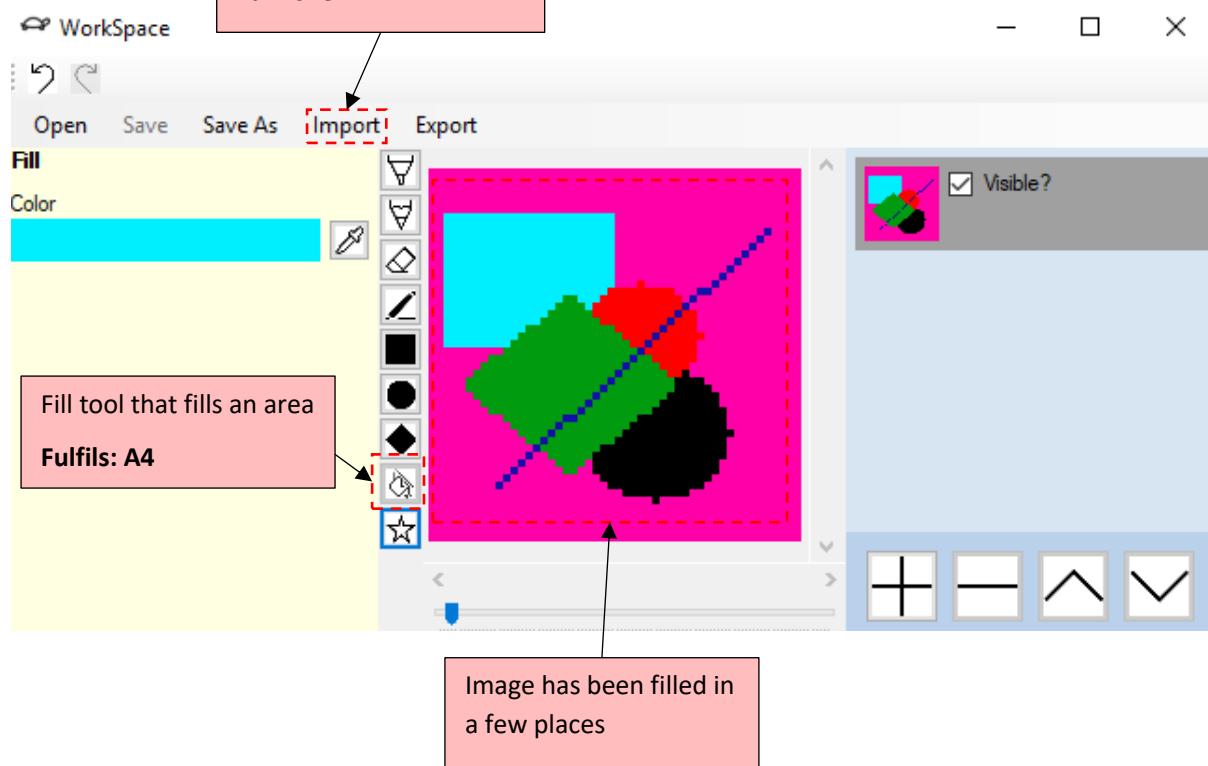




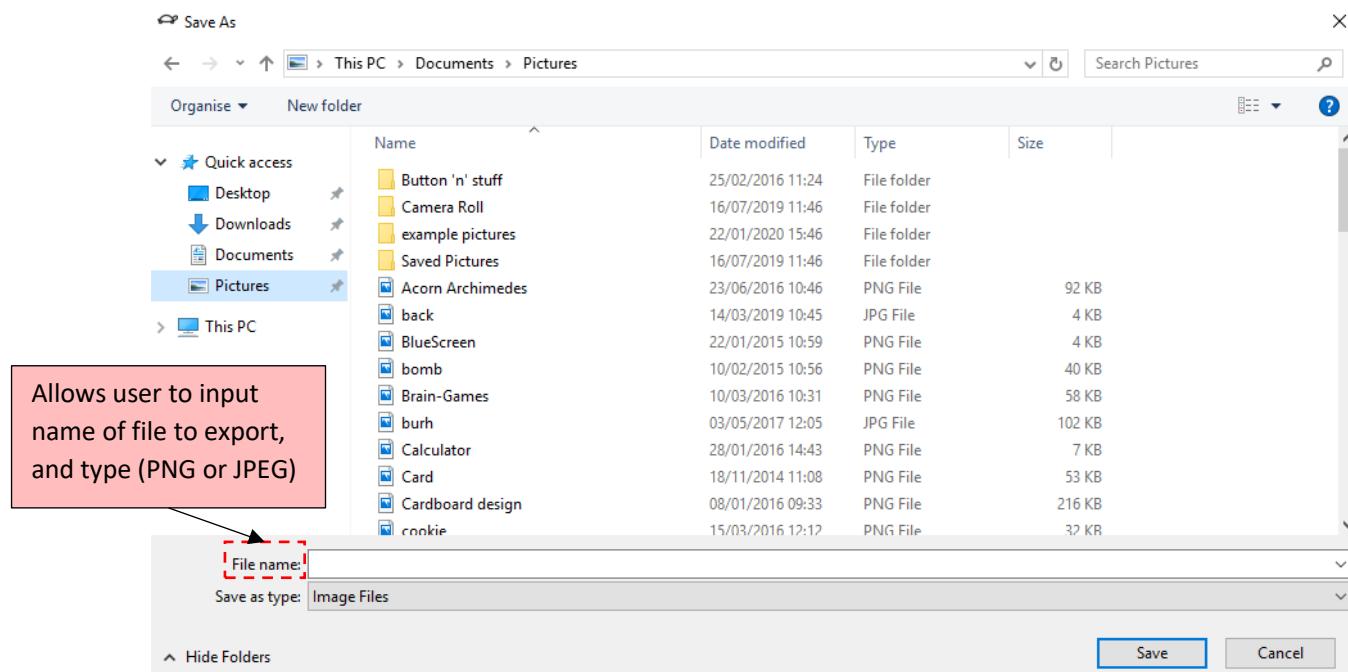
## Main Workspace



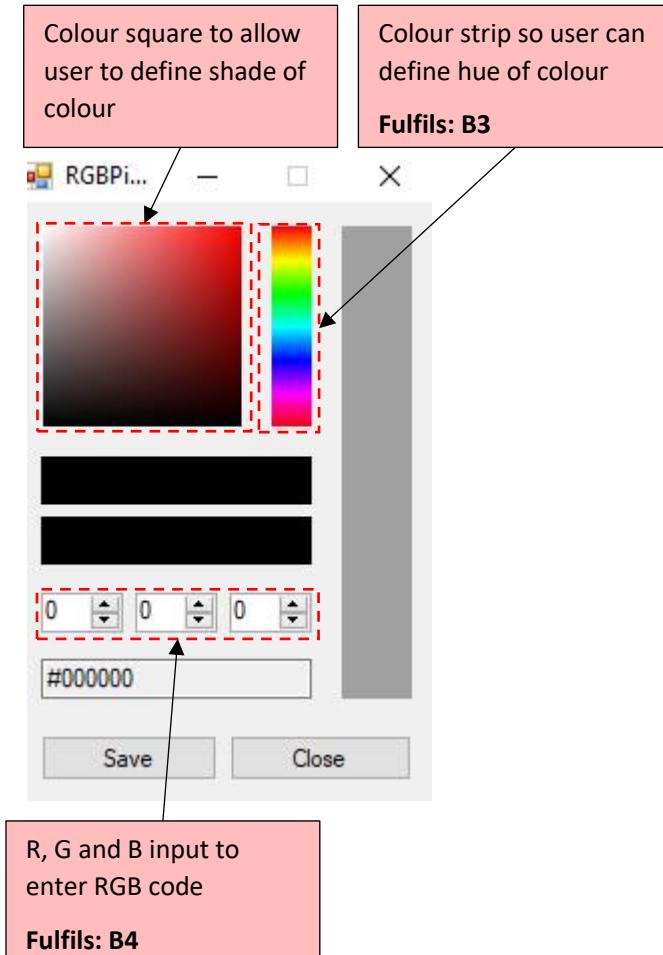


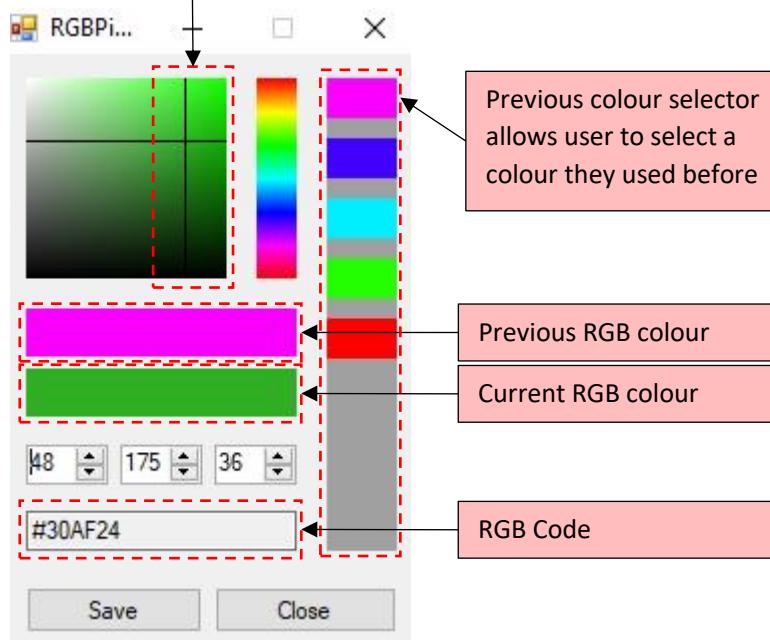


## SIMP Export Image Browser



## SIMP RGB Picker





## Full Criteria Fulfilment Diagram

Feature	Fulfilled?	Code
<b>Section A - Brushes</b>		
Variable brush width	Yes	A1
Hard brushes	Yes	A2
Shape creation tools	Yes	A3
Fill (bucket) tool	Yes	A4
Single pixel pencil	Yes	A5
Rubber	Yes	A6
<b>Section B – Other editing tools</b>		
Image viewer	Yes	B1
Bitmap image editor	Yes	B2
RGB colour picker	Yes	B3
RGB direct input	Yes	B4
Layer system	Yes	B5
Rectangle selection tool	No	B6
Magic selection tool	No	B7
Transparent pixels	Yes	B8
Zoom in (no zoom out)	Yes	B9
Text	No	B10
Eyedropper tool	Yes	B11
<i>Image effects</i>	Yes	B12
<i>Rotating Images</i>	No	B13
<i>Clipping masks</i>	No	B14
<b>Section C – File System</b>		
Creating a new image	Yes	C1
Importing images	Yes	C2
Exporting images	Yes	C3
Supporting PNG and JPEG	Yes	C4
Saving and loading from a proprietary format	Yes	C5
<b>Section D – Usability</b>		
Program should be stable and not crash.	Yes	D1
Program should be easy to use	Yes	D2
Features should be easily accessible	Yes	D3

