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:** Dheshpreet

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

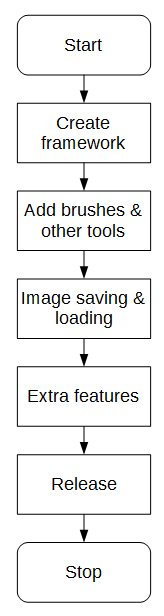
**Would benefit from:** More advanced tools, and support for larger images.

# 1.3 Computational Justification

The problems 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



Start of the project.

The internals of the software need to be initially created, involving features for setting pixels, creating an image and displaying it consistently.

The extra brushes and tools need to be added on top of the original framework. It should be noted to be preferable to, after the framework is finished it should need to be reliable and functional enough to not need to be edited at any other point in development

After images can be created, there need to be ways to bring them in and out of the program, either using popular image formats or a proprietary format for the program.

Any extra features that have been requested by the stakeholders can now be added (time permitting), however there will not be feature creep (features being added for the sake of being added), any added features will serve a purpose

Program is released to the stakeholders for open testing and formal feedback (whilst small prototypes would be released throughout development). Any minor tweaks and fixes can be added now.

Work ceases.

# 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 as intentionally very broad. This is 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 direly 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 images 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 overs 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.

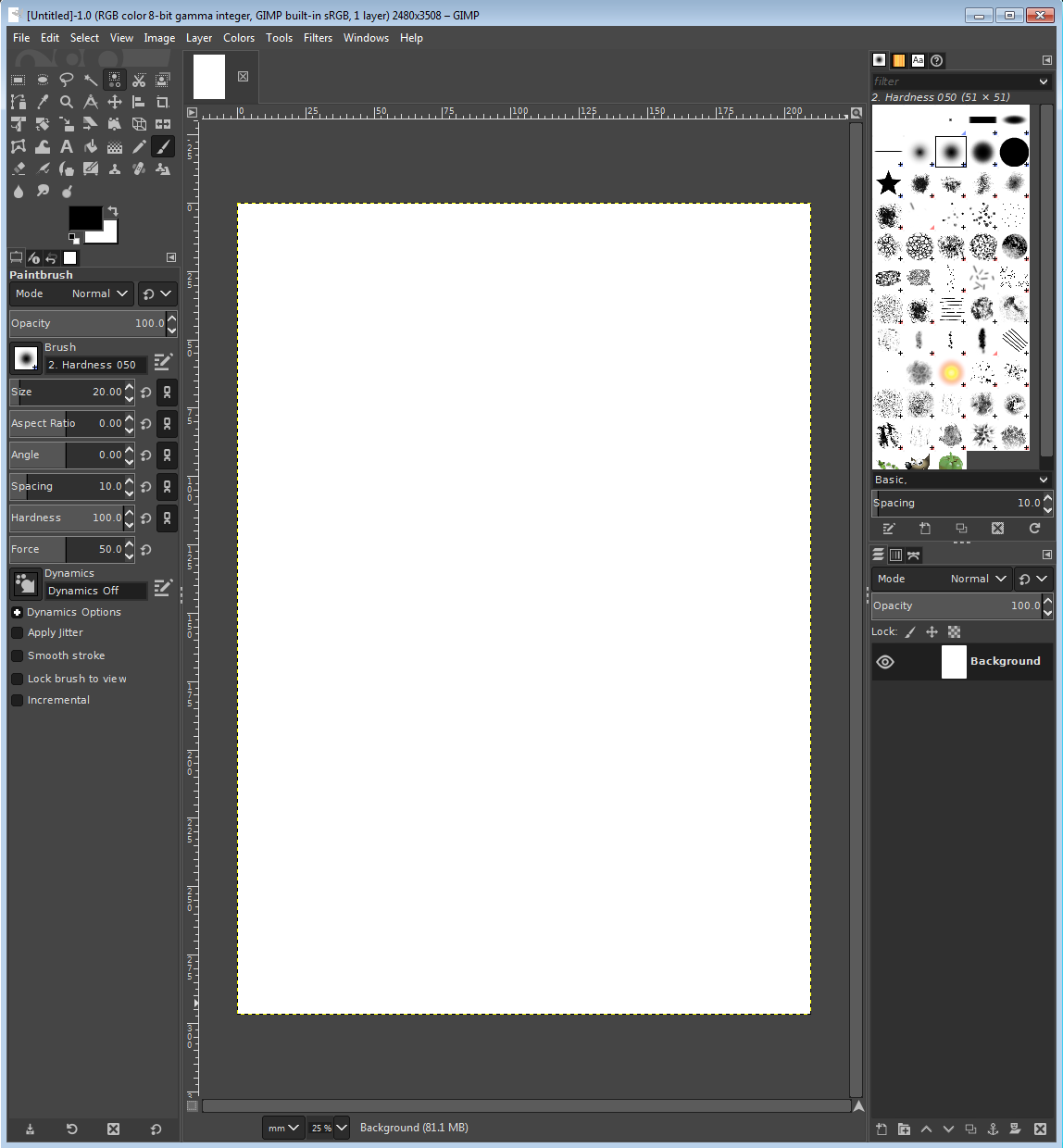
## Image result for gimp1.5.1 GIMP

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

Toolbox



Colour

Picker

Brush

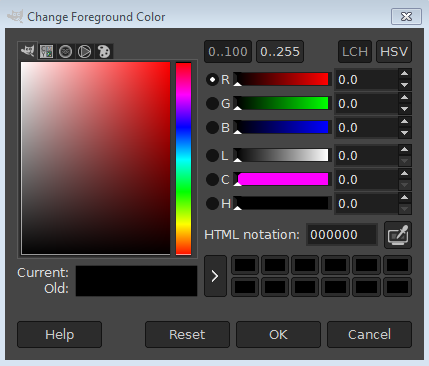
Properties

Picture display

Layer Selector

Sel

*A screenshot taken showing GIMP 2.10*



*The RGB picker from GIMP.*

### 1.5.1b Features

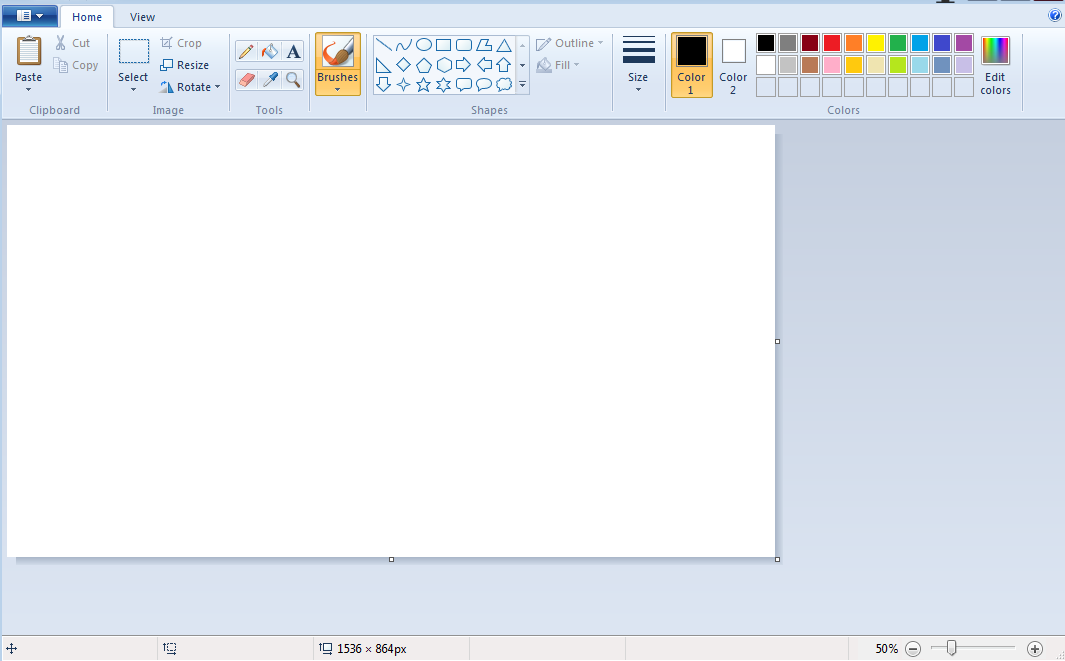
|  |  |
| --- | --- |
| **Feature** | **Description** |
| Rectangle Selection Tool | Allows a rectangle to be selected on the image, to be manipulated |
| Magic Selection Tool | Selects a region based on similar colours, used for quickly selecting an object with a complex border |
| RGB Selector | RGB 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

Colour Picker

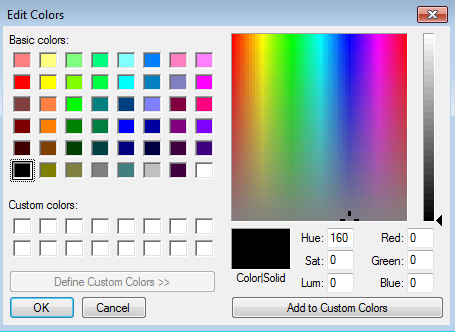
Toolbox

### 1.5.2a Screenshots



Page Display

*A screenshot taken showing Paint.*



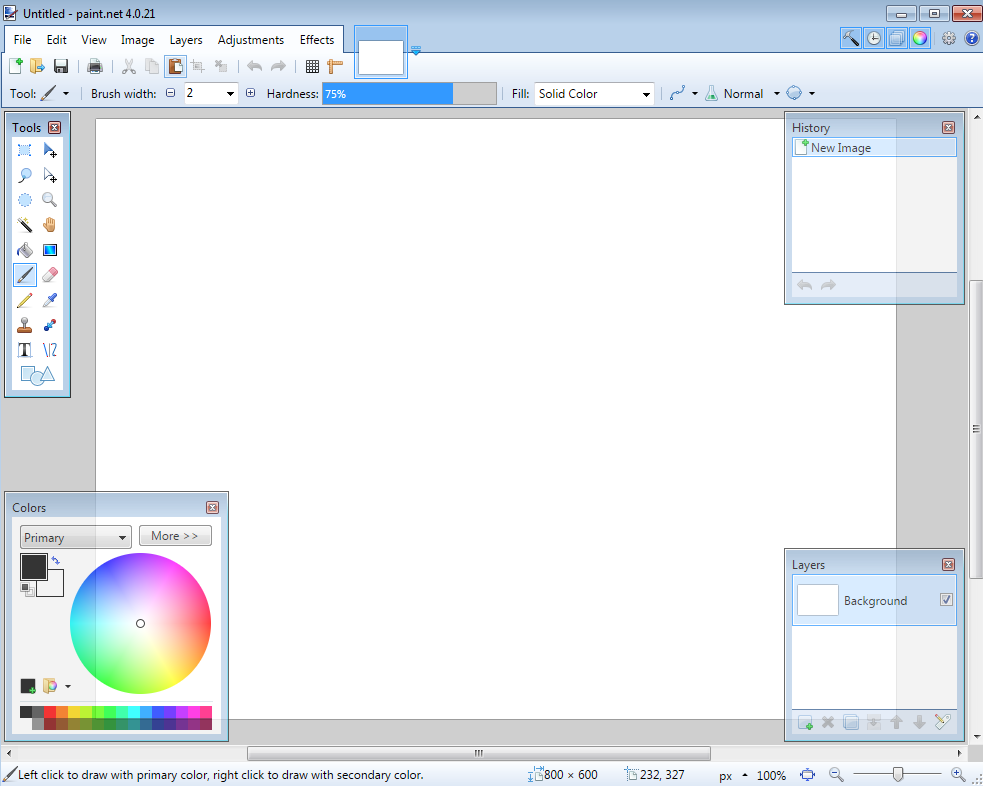
*The RGB picker from Paint.*

### 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 |

## Image result for paint.net logo1.5.3 Paint.NET

### 1.5.3a Screenshots



Page View

History

Viewer

Layer

Tool

Colour

Picker

Toolbox

### 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 allow numeric widths 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 | ✓ | ✗ | ✗ | 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 | ✗ | ✗ | ✓ | 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 GIM 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

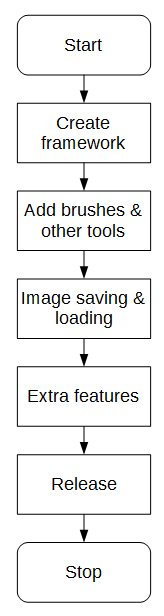
The ‘code’ field denotes a shortening that will be used to refer to that criteria, for example “Criteria A1”. *Italics denote stretch goals – optional features added if there is enough time.*

|  |  |  |
| --- | --- | --- |
| **Feature** | **Proof** | **Code** |
| 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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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.

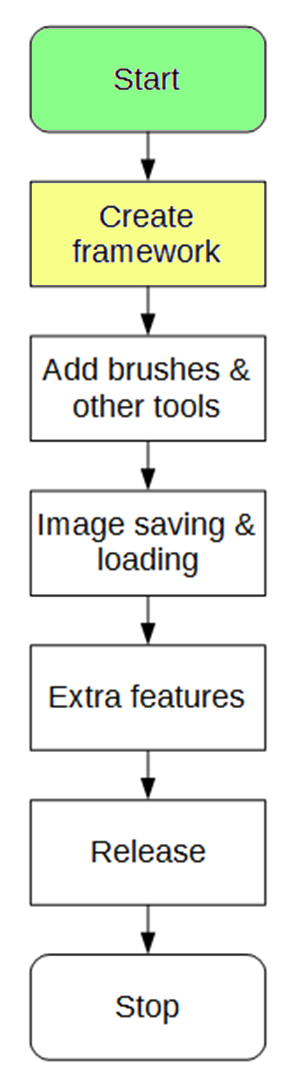
Section 2 – Framework

# General Contents

* [Decomposition Diagram](#_2.1.1_Project_Decomposition)
* [Usability Diagram](#_2.1.2_Usability)
* [IPO Diagrams](#_2.1.3_Inputs,_Processing,)
* [Class Design #1](#_2.1.4_Class_Design)
* [Algorithm Design](#_2.1.5_Algorithm_Design)
  + Contains smaller Unit Tests
* [Class Design #2](#_2.1.6_Class_Design)
  + This supersedes the previous Class Design
* [Class Relation Diagrams](#_2.1.7_Class_Relation)
* [Other key data structures](#_2.1.8_Key_Data)
* [Testing Table](#_2.1.9_Full_Section)
* [Testing Plan](#_2.1.10_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

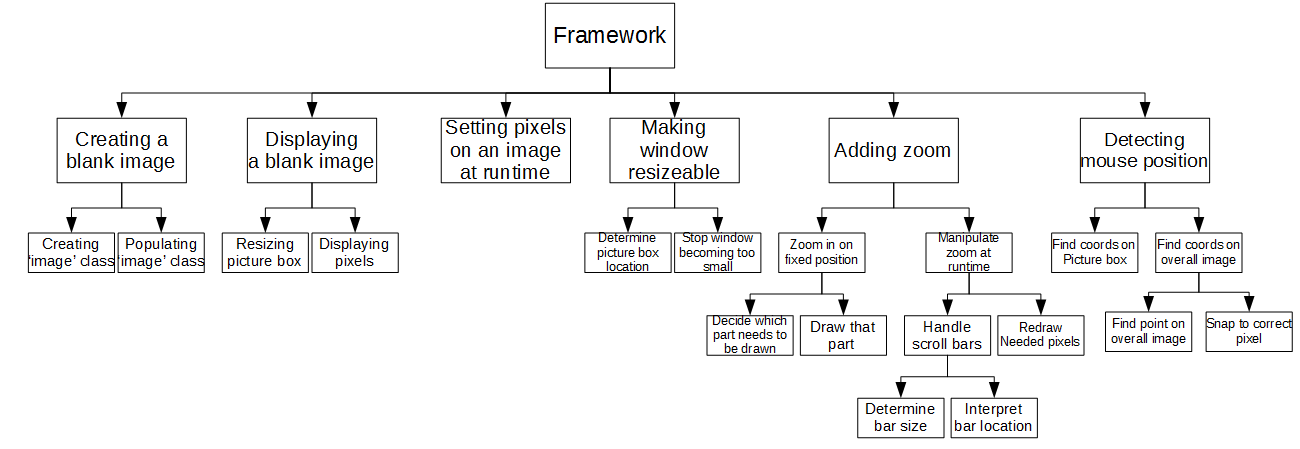
To be done this section

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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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



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

These algorithms will then be combined together.

[5](#_Algorithm_2.5_Setting)

[15](#_Algorithm_2.14_+)

[14](#_Algorithm_2.14_+)

[13](#_Algorithm_2.13_Determining)

[12](#_Algorithm_2.12_Redrawing)

[11](#_Algorithm_2.11_Interpreting)

[10](#_Algorithm_2.10_Determining)

[9](#_Algorithm_2.9_Draw)

[8](#_Algorithm_2.8_Deciding)

[7](#_Algorithm_2.7_Stopping)

[6](#_Algorithm_2.6_Determining)

[4](#_Algorithm_2.4_Displaying)

[3](#_Algorithm_2.3_Resizing_1)

[2](#_Algorithm_2.2_Populating)

[1](#_Algorithm_2.1_Creating)

## 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 *must* 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.

Canvas

## 2.1.3.1 UI Design

Scroll bars

Zoom bar

## 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**](#_2.1.6_Class_Design)

## 2.1.5.1 Image

<class>

Image

colours[,]

width

height

zoomSetting

GetPixel()

SetPixel()

GetDisplayImage()

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 | [ZoomSettings](#_2.1.4.2_ZoomSettings) | 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

<class>

ZoomSettings

centreLocation

zoomAmount

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

The default zoom will be none - 1x.

zoomAmount = 1

}

}

## 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](#_2.1.4.2_ZoomSettings) class

<class>

Workspace

image

width

height

form

displayBox

IsScrollBarVisible()

GetScrollBarMax()

GetScrollBarValue()

SetScrollBarValue()

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| image | [Image](#_2.1.4.1_Image) | 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 interraction |
| displayBox | PictureBox | Links the workspace to the Picture Box that it will use for displaying |

### Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Params** | **Return type** | **Justification** |
| IsScrollBar-Visible | axis, [Axis](#_2.1.4.2_Scrollbar), 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, [Axis](#_2.1.4.2_Scrollbar), the X or Y bar. | Integer | Calculates what the maximum for the selected scroll bar needs to be set to. |
| GetScroll-BarValue() | axis, [Axis](#_2.1.4.2_Scrollbar), the X or Y bar. | Integer | Calculates what the current value for the selected scroll bar needs to be set to. |
| SetScroll-BarValue() | axis, [Axis](#_2.1.4.2_Scrollbar), 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 **abstraction** 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

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

form = \_form

width = width of form

height = height of form

displayBox = new PictureBox

add displayBox to form

}

}

## 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.

<enum>

Axis

X

Y

|  |  |
| --- | --- |
| **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 |
| new Image(10,10) | Valid | An image of size 10fpx, 10fpx | This tests if a normal image can be created |
| new Image(10,6) | Valid | A 10fpx, 5fpx image | This tests if a non-rectangular image can be created |
| new Image(10,5) | Valid | A 10fpx, 2fpx image | This tests when an odd dimension is entered |
| new Image(10,1) | Extreme Valid | A 10fpx, 1fpx image | This tests if a very short image can be created |
| new Image(1,10) | Extreme Valid | A 1fpx, 10fpx image | This tests if a very thin image can be created |
| new Image(1,1) | Extreme Valid | A 1fpx, 1fpx image | This tests when a very small image is created |
| new Image(10,0) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no height is rejected |
| new Image(0,10) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no width is rejected |
| new Image(0,0) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no width or height is rejected |
| new Image(10000,10) | Extreme Invalid | The parameters are rejected and no image is created | This tests if a width which is much too large is rejected |
| new Image(10,10000) | Extreme Invalid | The parameters are rejected and no image is created | This tests if a height which is much too large is rejected |
| new Image(-1,10) | Invalid | The parameters are rejected and no image is created | This tests if a negative width is rejected. |
| new Image(10,-1) | Invalid | The parameters are rejected and no image is created | This tests if a negative height is rejected. |
| new Image(10) | Erraneous | The parameters are rejected and no image is created | This tests if an image missing a height is rejected. |
| new Image(“10”,”10”) | 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

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

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*

}

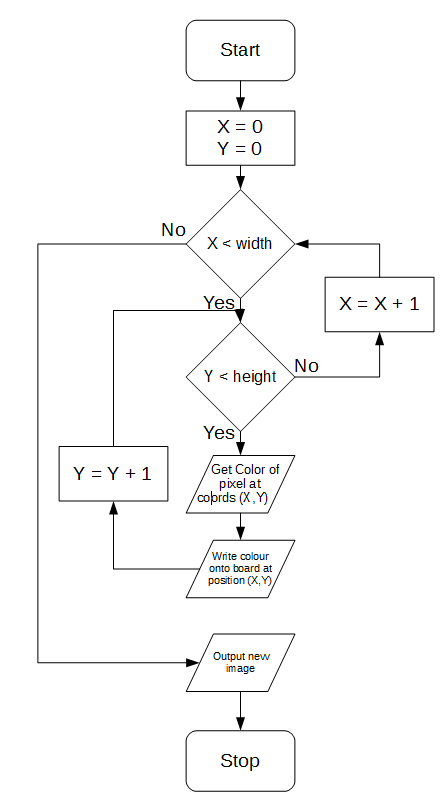
}

### Unit Testing

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

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

...

public Display {

ResizePictureBox(displayBox)

displayBox.DisplayImage = GetDisplayImage()

}

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

private [ResizePictureBox](#_Algorithm_2.3_Resizing)(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](#_Algorithm_2.4_Displaying)() {

// 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)

}

}

}

## Algorithm 2.5 Setting Pixels at Runtime

### Pseudocode

class Image {

...

SetPixel(x,y,colour) {

pixels[x,y] = colour

[Display()](#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

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

Form

### Diagram

imageHeight

formWidth

formHeight

imageWidth

Canvas

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

½formWidth  
 ½imageWidth

So this to find the X, the formula is ½formWidth - ½imageWidth or ½(formWidth – 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

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Test Type | Expected Output | Description |
| Form is started at normal size | Valid | Image Displays in the middle of the form | This tests that the program can start and display the image |
| Form is maximized | Extreme Valid | Image displays in the middle of the large form | This tests if the image is displayed when the form is very large |
| Form is minimized | Extreme Valid | No image is displayed (as form is currently invisible) | This tests that the program can be minimized safely |
| Form is resized to smallest possible | Extreme Valid | The form cannot be made smaller than the image | This tests that the form can never be made smaller than the image |
| Form is resized to minimum width maximum height | Extreme Valid | A very thin form displays the image | This tests if a form with an extreme height but minimum width is accepted |
| Form is resized to minimum height maximum width | Extreme Valid | A very short form displays the image | This tests if a form with an extreme width but minimum height is accepted |
| The form’s size is rapidly changed. | 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 users 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](#_Algorithm_2.8A_Converting).

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 10fpx10fpx, centred on the pixel at 5fpx,5fpx, then the zoomed area will be:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| formHeight |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | formWidth |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Key: Fully Drawn

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | formHeight  formWidth |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Key: Fully Drawn

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | formWidth | formHeight |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Form |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Thus, the form’s view will be:

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

(12fpx, 7fpx)

(9dpx, 9dpx)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | (8fpx, 3fpx)  (1dpx, 1dpx) |  |  |  |  |
|  |  |  |  |  |  |
|  |  | (10fpx, 5fpx)  (5dpx, 5dpx) |  |  |  |
|  |  |  |  |  |  |

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

(13fpx, 8fpx)

(9dpx, 9dpx)

(9fpx, 4fpx)

(1dpx, 1dpx)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | (11fpx, 6fpx)  (5dpx, 5dpx) |  |  |  |
|  |  |  |  |  |  |

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

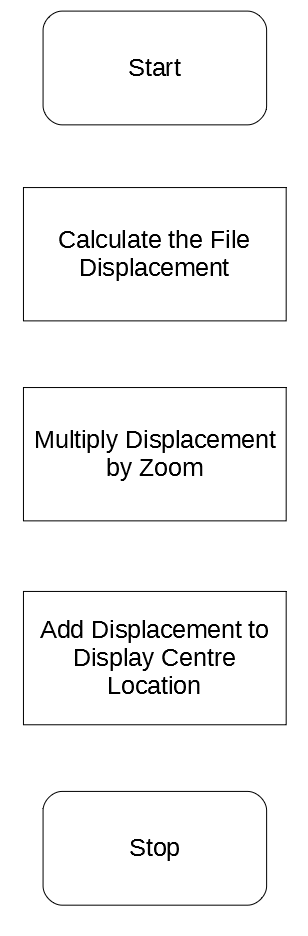
From these diagrams some conclusions can be drawn:

* 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)

(12fpx, 7fpx)

(9dpx, 9dpx)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | (8fpx, 3fpx)  (1dpx, 1dpx) |  |  |  |  |
|  |  |  |  |  |  |
|  |  | (10fpx, 5fpx)  (5dpx, 5dpx) |  |  |  |
|  |  |  |  |  |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Test Type | Expected Output | Description |
| DisplayPoint(10,5) | 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 |
| DisplayPoint(8,3) | Valid | (1,1) | This tests if a point to the top left of the centre location is positioned appropriately in the image |
| DisplayPoint(12,7) | Valid | (9,9) | This tests if a point to the bottom right of the centre location is positioned appropriately in the image |
| DisplayPoint(7,2) | 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. |
| DisplayPoint(13,8) | 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. |
| DisplayPoint(0,0) | Extreme Valid | (-15,-5) | This tests if a point in the top right corner has a relative display point calculated |
| DisplayPoint(19,9) | Extreme Valid | (23,13) | This tests if a point in the bottom left corner has a relative display point calculated |
| DisplayPoint(-1,-1) | Extreme Invalid | Throws out of bounds error | This tests if a point that is out of the bounds of the grid is not accepted |
| DisplayPoint(20,10) | 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](#_File_Pixels_to_1) algorithm should be employed.

##### Flowchart

This is the same steps as the File Pixels to Display Pixels [flowchart](#_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

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

displacementX = displacementX / zoom

displacementY = displacementY / zoom

newX = displacementX + centreFilePoint.X

newY = displacementY + centreFilePoint.Y

return new Point(newX, newY)

}

##### Unit Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Test Type | Expected Output | Description |
| FilePoint(5,5) | 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 |
| FilePoint(1,1) | Valid | (8,3) | This tests if a location near the top of the display box is returned as above the centre location |
| FilePoint(12,7) | Valid | (9,9) | This tests if a location near the bottom of the display box is returned as below the centre location |
| FilePoint(-1,-1) | 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 |
| FilePoint(11,11) | 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 |
| FilePoint(-15,-5) | Extreme Valid | (0,0) | This tests if a point at the top left of the file will be returned as such |
| FilePoint(23,13) | Extreme Valid | (19,9) | This tests if a point at the bottom right of the file will be returned as such |
| FilePoint(6,6) | Valid | (10,5) | This tests if a point that is halfway across a pixel is rounded down to its nearest file location |
| FilePoint(4,4) | Valid | (9,4) | This tests the same as above |
| FilePoint(-17,-7) | 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 |
| FilePoint(25,15) | 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](#_Pseudocode_1) 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 != 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 centreloc.X ± Ceiling(Floor(form.width/zoom) / 2)
  + The ± depends on whether we are looking for the left or right cross
  + In the above example this would be 10 + Ceiling(Floor(10/2)/2) = 10 + Ceiling(Floor(5)/2) = 10 + Ceiling(2.5) = 10 + 3 = 13.

* The X is centreloc.Y ± Ceiling(Floor(form.height/zoom) / 2)
  + The ± depends on whether we are looking for the left or right cross
  + In the above example this would be 5 + Ceiling(Floor(10/2)/2) = 5 + Ceiling(Floor(5)/2) = 5 + 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 {

This algorithm was designed above.

...

GetGreenPixels {

fileTopLeftPoint = orangeRectangle.topLeftCorner

fileBottomRightPoint = orangeRectangle.bottomRightCorner

for x = fileTopLeftPoint.X to fileBottomRightPoint.X

for y = fileTopLeftPoint.Y to fileBottomRightPoint.Y

[DrawGreenPixel(x,y)](#_Pseudocode)

next y

next x

}

}

## 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.

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

### Pseudocode

DrawGreenPixel(x,y) {

displayPoint = [FilePointToDisplayPoint(x,y)](#_File_Pixels_to)

DrawRectangle(displayPoint.X,displayPoint.Y,zoom,zoom,colours[x,y])

}

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 |
| Form is started at normal size | Valid | Image Displays in the middle of the form | This tests that the program can start and display the image |
| Form is maximized | Extreme Valid | Image displays in the middle of the large form | This tests if the image is displayed when the form is very large |
| Form is minimized | Extreme Valid | No image is displayed (as form is currently invisible) | This tests that the program can be minimized safely |
| Form is resized to smallest possible | Extreme Valid | The form cannot be made smaller than the image | This tests that the form can never be made smaller than the image |
| Form is resized to minimum width maximum height | Extreme Valid | A very thin form displays the image | This tests if a form with an extreme height but minimum width is accepted |
| Form is resized to minimum height maximum width | Extreme Valid | A very short form displays the image | This tests if a form with an extreme width but minimum height is accepted |
| The form’s size is rapidly changed. | 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:

Scroll bars

### 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](#_2.1.4.2_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

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Test Type | Expected Output | Description |
| Form is started | 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 |
| Form width is reduced to less than image | Extreme Valid | The horizontal scroll bar becomes visible | This makes sure that the scroll bar becomes visible |
| Form width is then increased | Extreme Valid | The horizontal scroll bar disappears | This makes sure that the scroll bar then becomes invisible |
| Form height is reduced to less than image | Extreme Valid | The vertical scroll bar becomes visible | This makes sure that the scroll bar becomes visible |
| Form height is then increased | Extreme Valid | The vertical scroll bar disappears | This makes sure that the scroll bar then becomes invisible |
| Form height and width is both reduced | Extreme Valid | Both scroll bars becomes visible | This makes sure both bars can be visible at the same time |
| Form height is then increased | Extreme Valid | The vertical scroll bar becomes invisible but horizontal scroll bar remains | This tests than one scroll bar can disappear but the other remains |
| Form width is then increased | Extreme Valid | 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 ½ of its available space. As 10dpx / 20dpx = ½

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](#centreLocation), 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](#_Algorithm_2.8B_Determining), 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 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](#_Pseudocode_2), 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

|  |  |  |  |
| --- | --- | --- | --- |
| Test Data | Test Type | Expected Output | Description |
| Form is resized to half the size of image | Valid | Bar is half size of bounds and in its centre position | This tests if the bar can display normally |
| Form is resized to its smallest position | Extreme Valid | Bar is small but still central. | This tests if the centre location remains in its position when the window size is reduced |
| Form is resized back to half size of image | 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 |
| Horizontal Scroll bar is moved to far left | 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 |
| Horizontal Scroll bar is moved to far right | 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 |
| Vertical Scroll bar is moved to highest | 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 |
| Vertical Scroll bar is moved to lowest | 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 |
| Horizontal scroll bar is moved far right, then form size increased | 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 |
| Form is resized to smallest, then maximized | Extreme Valid | The bars disappear and the view is normal | This tests that the program will cope if form size is rapidly increased |

## 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

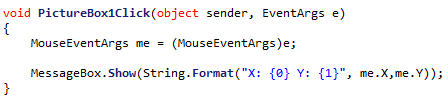
The last task that needs to be implemented now is handling interaction with the mouse. This will be integral for any user interaction with the program.

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.

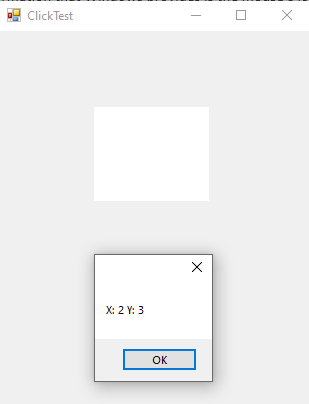
This can be solved by using C#.NET’s in-built Click event. This returns the location of the mouse relative to the picture box.

### Demonstration Code Prototype

This uses a **prototype** to test code, a throwaway experiment to make sure the assumption is correct.



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](#_Pseudocode_1), 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**](#_2.1.7_Class_Relation)**.**

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.

<interface>

IPicturePoint

GetFilePoint

GetDisplayPoint

### 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

<class>

FilePoint

fileX

fileY

\*GetFilePoint

\*GetDisplayPoint

GetFileValue

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 | [FilePoint](#_2.1.6.2_FilePoint) | Will return itself (as this is a file point) |
| GetDisplay-Point | None | [DisplayPoint](#_2.1.6.3_DisplayPoint) | 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

<class>

DisplayPoint

displayX

displayY

\*GetFilePoint

\*GetDisplayPoint

GetDisplayValue

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 | [FilePoint](#_2.1.6.2_FilePoint) | Will convert itself into a file point |
| GetDisplay-Point | None | [DisplayPoint](#_2.1.6.3_DisplayPoint) | 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

<interface>

IPictureRectangle

GetFileRectangle

GetDisplayRectangle

### Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Params** | **Return type** | **Justification** |
| GetFile-Rectangle | None | [FileRectangle](#_2.1.6.5_FileRectangle) | Will return the rectangle represented as a File Rectangle |
| GetDisplay-Rectangle | None | [DisplayRectangle](#_2.1.6.5_DisplayRectangle) | Will return the rectangle represented as a Display Rectangle |

## 2.1.7.5 FileRectangle

<class>

FileRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetFileSizeInAxis

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| topLeft-Corner | (private)  [FilePoint](#_2.1.6.2_FilePoint) | The location of the top left corner in the file  It is private to enforce getting the location through GetTopLeftCorner |
| bottomRight-Corner | (private)  [FilePoint](#_2.1.6.2_FilePoint) | 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 | [FileRectangle](#_2.1.6.5_FileRectangle) | Will return itself as this is already a file rectangle |
| \*GetDisplay-Rectangle | None | [DisplayRectangle](#_2.1.6.5_DisplayRectangle) | Will convert itself into a display rectangle |
| GetTop-LeftCorner | None | [FilePoint](#_2.1.6.2_FilePoint) | Will return the location in the file of the top left corner |
| GetBottom-RightCorner | None | [FilePoint](#_2.1.6.2_FilePoint) | Will return the location in the file of the bottom right corner |
| GetFile-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(fileTopLeftX, fileTopLeftY, fileBottomRightX, fileBottomRightY) {

topLeftCorner = new FilePoint(fileTopLeftX, fileTopLeftY)

bottomRightCorner = new FilePoint(fileBottomRightX, fileBottomRightY)

}

## 2.1.7.5 DisplayRectangle

<class>

DisplayRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetDisplaySizeInAxis

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| topLeft-Corner | (private)  [DisplayPoint](#_2.1.6.3_DisplayPoint) | The location of the top left corner in the Display  It is private to enforce getting the location through GetTopLeftCorner |
| bottomRight-Corner | (private)  [DisplayPoint](#_2.1.6.3_DisplayPoint) | 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 | [FileRectangle](#_2.1.6.5_FileRectangle) | Will convert itself into a File Rectangle |
| \*GetDisplay-Rectangle | None | [DisplayRectangle](#_2.1.6.5_DisplayRectangle) | Will return itself as this is already a Display Rectangle |
| GetTop-LeftCorner | None | [DisplayPoint](#_2.1.6.3_DisplayPoint) | Will return the location in the Display of the top left corner |
| GetBottom-RightCorner | None | [DisplayPoint](#_2.1.6.3_DisplayPoint) | 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

<class>

Image

colours[,]

width

height

zoomSetting

attachedWorkspace

GetPixel()

SetPixel()

ToDisplayRectangle()\*

ToFileRectangle()\*

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 | [ZoomSettings](#_2.1.6.7_ZoomSettings) | Stores the current settings describing the zoom of the image. |
| attached-Workspace | [Workspace](#_2.1.6.8_Workspace) | 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

<class>

ZoomSettings

centreLocation

zoomAmount

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 | [FilePoint](#_2.1.6.2_FilePoint) | 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

The default zoom will be none - 1x.

zoomAmount = 1

}

}

## 2.1.7.8 Workspace

<class>

Workspace

displayPictureBox

attachedForm

displayWidth

displayHeight

LEFT\_PADDING

RIGHT\_PADDING

TOP\_PADDING

BOTTOM\_PADDING

GetPixel

SetPixel

GetDisplaySizeInAxis

GetDisplayRectangle

Display

HandleResize

### 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 [GetPixel](#_Methods), 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 [SetPixel](#_Methods), also included here for convenience |
| GetDisplay-SizeInAxis | None | Integer | Returns the amount of display pixels in a certain axis |
| GetDisplay-Rectangle | None | [DisplayRectangle](#_2.1.6.5_DisplayRectangle) | 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.

<enum>

Axis

X

Y

### 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

<class>

DisplayPoint

displayX

displayY

GetFilePoint

GetDisplayPoint

GetDisplayValue

<class>

FilePoint

fileX

fileY

GetFilePoint

GetDisplayPoint

GetFileValue

<interface>

IPicturePoint

GetFilePoint

GetDisplayPoint

Key:

A B   
A inherits from B

## 2.1.8.2 Rectangle abstraction stack

<class>

DisplayPoint

displayX

displayY

GetFilePoint

GetDisplayPoint

GetDisplayValue

<class>

FilePoint

fileX

fileY

GetFilePoint

GetDisplayPoint

GetFileValue

<class>

DisplayRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetDisplaySizeInAxis

<interface>

IPictureRectangle

GetFileRectangle

GetDisplayRectangle

<class>

FileRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetFileSizeInAxis

Key:

A B   
A inherits from B

A B   
A contains references to B

## 2.1.8.3 Overall stack

<class>

Image

colours[,]

width

height

zoomSetting

attachedWorkspace

GetPixel()

SetPixel()

<class>

Workspace

displayPictureBox

attachedForm

displayWidth

displayHeight

LEFT\_PADDING

RIGHT\_PADDING

TOP\_PADDING

BOTTOM\_PADDING

GetPixel

SetPixel

GetDisplaySizeInAxis

GetDisplayRectangle

Display

HandleResize

<class>

ZoomSettings

centreLocation

zoomAmount

<interface>

IPictureRectangle

GetFileRectangle

GetDisplayRectangle

Key:

A B   
  
A inherits from B

A B   
  
A contains references to B

# 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 |
| new Image(10,10) | Valid | An image of size 10fpx, 10fpx | This tests if a normal image can be created | 1 |
| new Image(10,6) | Valid | A 10fpx, 5fpx image | This tests if a non-rectangular image can be created | 2 |
| new Image(10,5) | Valid | A 10fpx, 2fpx image | This tests when an odd dimension is entered | 3 |
| new Image(10,1) | Extreme Valid | A 10fpx, 1fpx image | This tests if a very short image can be created | 4 |
| new Image(1,10) | Extreme Valid | A 1fpx, 10fpx image | This tests if a very thin image can be created | 5 |
| new Image(1,1) | Extreme Valid | A 1fpx, 1fpx image | This tests when a very small image is created | 6 |
| new Image(10,0) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no height is rejected | 7 |
| new Image(0,10) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no width is rejected | 8 |
| new Image(0,0) | Extreme Invalid | The parameters are rejected and no image is created | This tests if an image with no width or height is rejected | 9 |
| new Image(10000,10) | Extreme Invalid | The parameters are rejected and no image is created | This tests if a width which is much too large is rejected | 10 |
| new Image(10,10000) | Extreme Invalid | The parameters are rejected and no image is created | This tests if a height which is much too large is rejected | 11 |
| new Image(-1,10) | Invalid | The parameters are rejected and no image is created | This tests if a negative width is rejected. | 12 |
| new Image(10,-1) | Invalid | The parameters are rejected and no image is created | This tests if a negative height is rejected. | 13 |
| new Image(10) | Erraneous | The parameters are rejected and no image is created | This tests if an image missing a height is rejected. | 14 |
| new Image(“10”,”10”) | Erraneous | The parameters are rejected and no image is created | This tests if an image with invalid numbers is rejected | 15 |
| SetPixel(5,5,Black) | Valid | A pixel near the middle of the image is set to black | This tests if a normal pixel can be set | 16 |
| SetPixel(5,5,Green) | 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 |
| SetPixel(0,0,Black) | Extreme Valid | A pixel in the top-left corner is set to black | This tests if the top-left corner can be set | 18 |
| SetPixel(9,9,Black) | Extreme Valid | A pixel in the bottom-right corner is set to black | This tests if the bottom-right corner can be set | 19 |
| SetPixel(-1,0,Black) | Extreme Invalid | No pixel is set as it is out of bounds | This tests if a pixel too far left is rejected | 20 |
| SetPixel(0,-1,Black) | Extreme Invalid | No pixel is set as it is out of bounds | This tests if a pixel too far up is rejected | 21 |
| SetPixel(10,0,Black) | Extreme Invalid | No pixel is set as it is out of bounds | This tests if a pixel too far right is rejected | 22 |
| SetPixel(0,10,Black) | Extreme Invalid | No pixel is set as it is out of bounds | This tests if a pixel too far down is rejected | 23 |
| Form is started at normal size | Valid | Image Displays in the middle of the form | This tests that the program can start and display the image | 24 |
| Form is maximized | 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 |
| Form is minimized | Extreme Valid | No image is displayed (as form is currently invisible) | This tests that the program can be minimized safely | 26 |
| Form is resized to smallest possible | 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 |
| Form is resized to minimum width maximum height | Extreme Valid | A very thin form displays the image | This tests if a form with an extreme height but minimum width is accepted | 28 |
| Form is resized to minimum height maximum width | Extreme Valid | A very short form displays the image | This tests if a form with an extreme width but minimum height is accepted | 29 |
| The form’s size is rapidly changed. | 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 |
| Form is resized to half the size of image | Valid | Bar is half size of bounds and in its centre position | This tests if the bar can display normally | 31 |
| Form is resized to its smallest position | 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 |
| Form is resized back to half size of image | 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 |
| Horizontal Scroll bar is moved to far left | 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 |
| Horizontal Scroll bar is moved to far right | 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 |
| Vertical Scroll bar is moved to highest | 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 |
| Vertical Scroll bar is moved to lowest | 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 |
| Horizontal scroll bar is moved far right, then form size increased | 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 |
| Form is resized to smallest, then maximized | 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| The data inputted into the program | 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 |

# 2.1.22 Beta Testing Plan

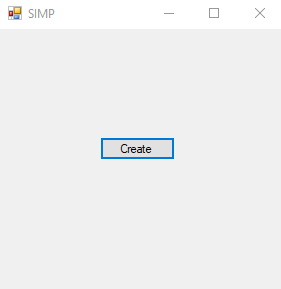
Beta testing will be done after development, after the program is in a stable state. This will involves **Functionality Testing, Durability Testing** and **Usability Testing.**

It will be done in the same format as the above tests, but with a more **black box** approach, where the internals of the algorithm are *not* considered, just the outputs. This is to ensure a more thorough batch testing.

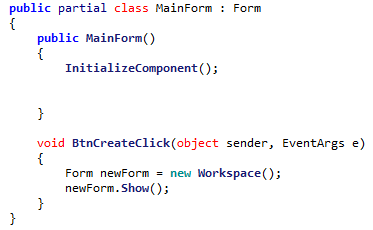
2.2 Developement

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

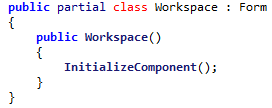
A basic design for the Main Form has been made:



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



The Workspace itself only contains code for creating a form:



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:

<class>

Workspace

image

width

height

form

displayBox

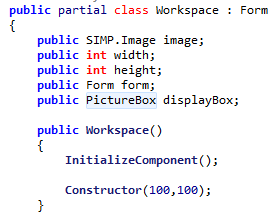
IsScrollBarVisible()

GetScrollBarMax()

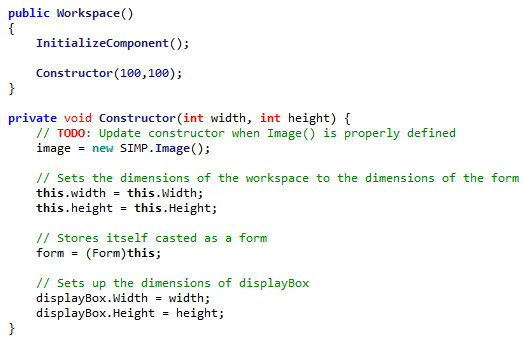
GetScrollBarValue()

SetScrollBarValue()

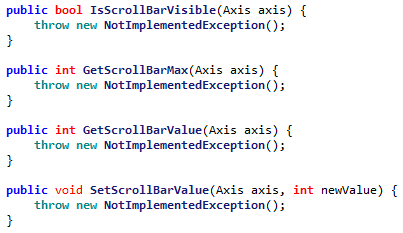
Properties:



The constructor has also been defined:



The necessary functions have also been defined:



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

## Axis Implementation

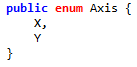
The Axis enum has been implemented in accordance to [2.1.5.2](#_2.1.5.2_Axis):

<enum>

Axis

X

Y



## Image Implementation

The image implementation has been completed in accordance to [2.1.5.1](#_2.1.5.1_Image).

<class>

Image

colours[,]

width

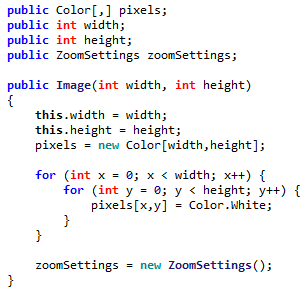
height

zoomSetting

GetPixel()

SetPixel()

GetDisplayImage()



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 {

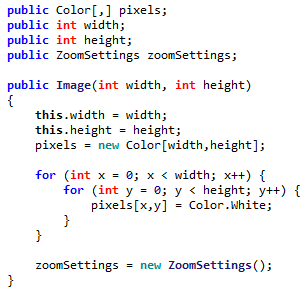
Color = White

}

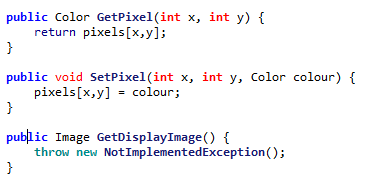
zoomSettings = new ZoomSetting()

}

}



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



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

## ZoomSettings implementation

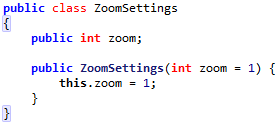
ZoomSettings has been implemented in accordance to [2.1.5.2](#_2.1.5.2_ZoomSettings):

<class>

ZoomSettings

centreLocation

zoomAmount



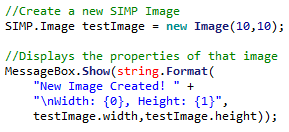
However centreLocation has not been implemented. This is since it will only be relevant when the zooming has been fully implemented.

In doing this, [Algorithm 2.1](#_Algorithm_2.1_Creating) and [Algorithm 2.2](#_Algorithm_2.2_Populating) have already been implemented. This means the [2.2 Unit test](#_Unit_Testing) 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:



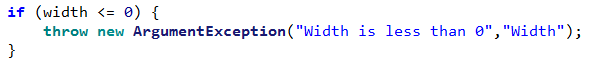
This is the part that will be changed during the testing phase

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| new Image(10,10) | 1 | An image of size 10fpx, 10fpx |  | The valid image is created |
| new Image(10,5) | 2 | A 10fpx, 5fpx image |  | The valid image is created |
| new Image(5,10) | 3 | A 5fpx, 10fpx image |  | The valid image is created |
| new Image(10,1) | 4 | A 10fpx, 1fpx image |  | The valid image is created |
| new Image(1,10) | 5 | A 1fpx, 10fpx image |  | The valid image is created |
| new Image(1,1) | 6 | A 1fpx, 1fpx image |  | The valid image is created |
| new Image(10,0) | 7 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(0,10) | 8 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(0,0) | 9 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(10000,10) | 10 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(10,10000) | 11 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(-1,10) | 12 | The parameters are rejected and no image is created | System.OverflowException: Arithmetic operation resulted in an overflow. | A potentially confusing error is thrown |
| new Image(10,-1) | 13 | The parameters are rejected and no image is created | System.OverflowException: Arithmetic operation resulted in an overflow. | A potentially confusing error is thrown |
| new Image(10) | 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 |
| new Image(“10”,”10”) | 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

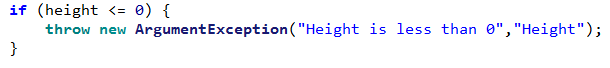
To fix these errors, a small check can be implemented:



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

### Fixing Error #8, #13

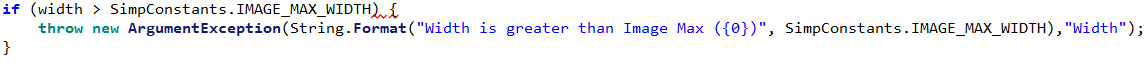
To fix these errors, a small check can be implemented:



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

### Fixing Error #10

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



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

### Fixing Error #11

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



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

## Unit 2.2 Unit Test II

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| new Image(10,10) | 1 | An image of size 10fpx, 10fpx |  | The valid image is created |
| new Image(10,5) | 2 | A 10fpx, 5fpx image |  | The valid image is created |
| new Image(5,10) | 3 | A 5fpx, 10fpx image |  | The valid image is created |
| new Image(10,1) | 4 | A 10fpx, 1fpx image |  | The valid image is created |
| new Image(1,10) | 5 | A 1fpx, 10fpx image |  | The valid image is created |
| new Image(1,1) | 6 | A 1fpx, 1fpx image |  | The valid image is created |
| new Image(10,0) | 7 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| new Image(0,10) | 8 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(0,0) | 9 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(10000,10) | 10 | The parameters are rejected and no image is created | System.ArgumentException: Width is greater than Image Max (9999) | A descriptive error is thrown |
| new Image(10,10000) | 11 | The parameters are rejected and no image is created | System.ArgumentException: Height is greater than Image Max (9999) | A descriptive error is thrown |
| new Image(-1,10) | 12 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(10,-1) | 13 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| new Image(10) | 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 |
| new Image(“10”,”10”) | 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:

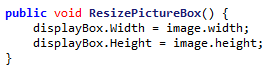


I **discussed with my stakeholders**, and they agreed that 99,999 pixels would be the largest that they’d need for an image.

# 21/09/2019 Picture Box setup

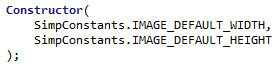
## Algorithm 2.3 – Resizing Picture Box

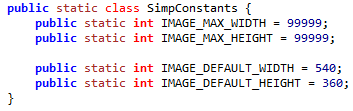
The algorithm for resizing the picture box has been implemented, though contains **changes** from the planned pseudocode:



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.





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 |
| Input Box(10,10) | 1 | An image of size 10fpx, 10fpx |  | The Picture Box is correctly resized. |
| Input Box(10,5) | 2 | A 10fpx, 5fpx image |  | The Picture Box is correctly resized. |
| Input Box(5,10) | 3 | A 5fpx, 10fpx image |  | The Picture Box is correctly resized. |
| Input Box(10,1) | 4 | A 10fpx, 1fpx image |  | The Picture Box is correctly resized.  (Though is very thin) |
| Input Box(1,10) | 5 | A 1fpx, 10fpx image |  | The Picture Box is correctly resized.  (Though is very thin) |
| Input Box(1,1) | 6 | A 1fpx, 1fpx image |  | The Picture Box is correctly resized.  (Though is very small) |
| Input Box(10,0) | 7 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| Input Box(0,10) | 8 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(0,0) | 9 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(10000,10) | 10 | The parameters are rejected and no image is created | System.ArgumentException: Width is greater than Image Max (9999) | A descriptive error is thrown |
| Input Box(10,10000) | 11 | The parameters are rejected and no image is created | System.ArgumentException: Height is greater than Image Max (9999) | A descriptive error is thrown |
| Input Box(-1,10) | 12 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(10,-1) | 13 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| Input Box(10) | 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 |
| Input Box(“10”,”10”) | 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]

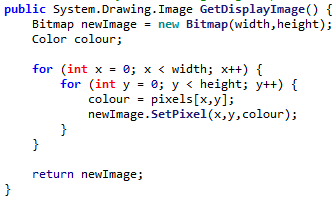
Image.SetPixel(x,y,colour)

}

}

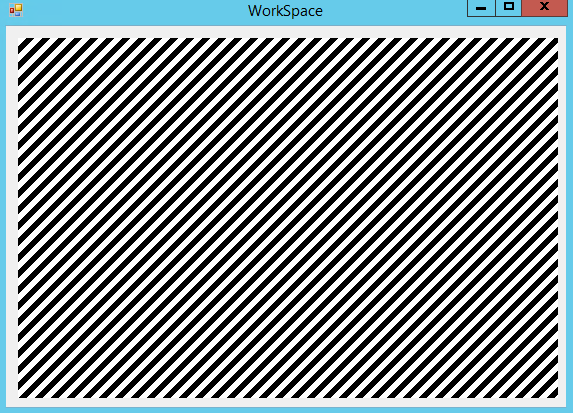
RETURN image

}



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





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 – Settings 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()

}



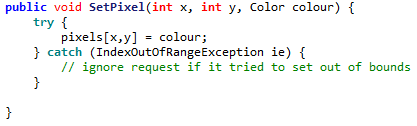
However there has been a change, as the call to Display() has been removed. This makes it so that the image will *not* be redrawn each time a pixel it set, and this helps to optimise large groups of pixel manipulation. This also means that it will be up to the calling function to re-display when necessary

## Unit 2.5 Unit Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| SetPixel(5,5,Black) | 1 | A pixel near the middle of the image is set to black |  | This proves a pixel can be placed on the image. |
| SetPixel(5,5,Green) | 2 | A pixel near the middle of the image is set to yellow |  | This proves multiple colours of pixel can be set |
| SetPixel(0,0,Black) | 3 | A pixel in the top-left corner is set to black |  | This proves the top-left corner is properly accessible |
| SetPixel(9,9,Black) | 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 |
| SetPixel(-1,0,Black) | 5 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(0,-1,Black) | 6 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(10,0,Black) | 7 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(0,10,Black) | 8 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: 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:



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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SetPixel(5,5,Black) | 1 | A pixel near the middle of the image is set to black |  | This proves a pixel can be placed on the image. |
| SetPixel(5,5,Green) | 2 | A pixel near the middle of the image is set to yellow |  | This proves multiple colours of pixel can be set |
| SetPixel(0,0,Black) | 3 | A pixel in the top-left corner is set to black |  | This proves the top-left corner is properly accessible |
| SetPixel(9,9,Black) | 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 |
| SetPixel(-1,0,Black) | 5 | No pixel is set as it is out of bounds |  | This proves a too small X is rejected |
| SetPixel(0,-1,Black) | 6 | No pixel is set as it is out of bounds |  | This proves a too large X is rejected |
| SetPixel(10,0,Black) | 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

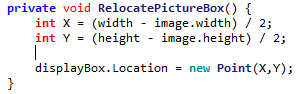
RelocatePictureBox {

Integer X = (width – image.width) / 2

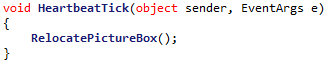
Integer Y = (height – image.height) / 2

displayBox.Location = new Location(X,Y)

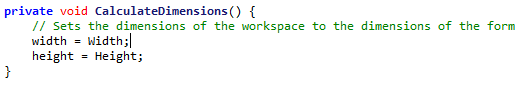
}



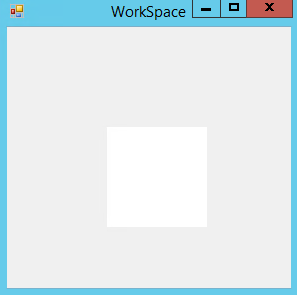
However, some additional code must be added to make the picture box constantly update. To do this a small timer module has been added to the code which calls the following procedure every millisecond:



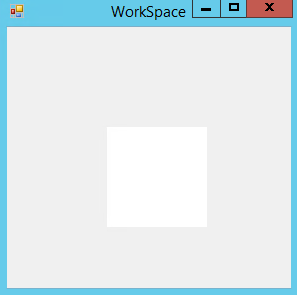
However this is still not enough, as ‘width’ and ‘height’ are currently constant and do not update when the form is resized. This can be solved by adding a new event called when the workspace is resized:



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



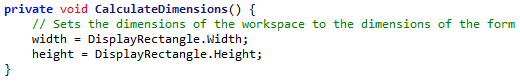
This is caused by the fact that calling the width of the form will return the **entire** width of the form, including its borders. For example:



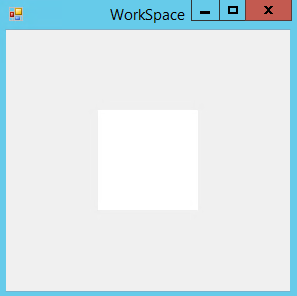
Desired Height

Height returned by ‘Height’

This can be fixed by instead calling the ‘[DisplayRectangle](https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.control.displayrectangle?view=netframework-4.8)’ property, which returns a rectangle with the desired dimensions:



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

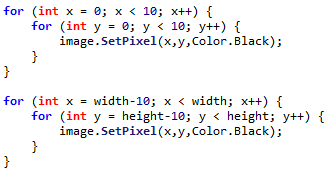
form.minimumHeight = image.height



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

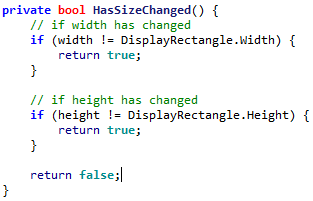
In order to test this, two boxes have been added to the bottom left and top right of the image:



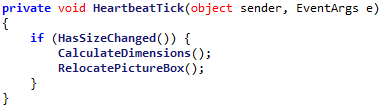
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is started at normal size | 1 | Image Displays in the middle of the form |  | The picture box is not correctly resized upon starting |
| Form is maximized | 2 | Image displays in the middle of the large form |  | The picture box is not resized upon maximizing |
| Form is minimized | 3 | No image is displayed (as form is currently invisible) | (No screenshot) | The program does not crash upon minimizing |
| Form is resized to smallest possible | 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 |
| Form is resized to minimum width maximum height | 5 | A very thin form displays the image |  | The pixels in the corners are still cut off |
| Form is resized to minimum height maximum width | 6 | A very short form displays the image |  | The pixels in the corners are not displayed |
| The form’s size is rapidly changed. | 7 | The image is very quickly moved around but remains centred |  | The picture box is only moved upon the end of resizing so the picture box aligns correctly eventually |

### 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:

****

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



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 |  | The picture box is resized but the picture box does not correctly display the entire image. |
| Form is maximized | 2 | Image displays in the middle of the large form |  | The picture box is correctly placed when the form is resized |

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

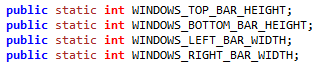
### Fixing Error #1 & #4 & #5 & #6

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

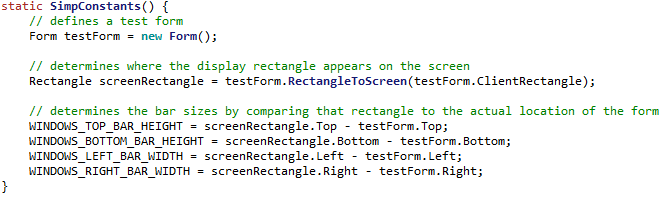


The error comes from the fact that MinimumSize does not take into account the borders around the Form, meaning elements like the top bar can cut the image off.

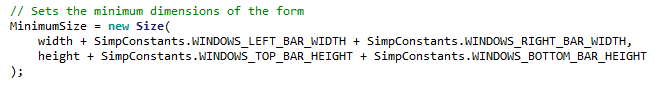
To resolve this, the size of the top bar (and other borders) needs to be determined so that they can be factored into the equation.



They can be calculated using the following code:



So now they can be included in the size calculations



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is started at normal size | 1 | Image Displays in the middle of the form |  | The picture box starts at its smallest size, which is still valid. |
| Form is resized to smallest possible | 4 | The form cannot be made smaller than the image |  | The picture box cannot be made any smaller than this size. |
| Form is resized to minimum width maximum height | 5 | A very thin form displays the image |  | A very tall form still correctly displays the pixels |
| Form is resized to minimum height maximum width | 6 | A very short form displays the image |  | The pixels in the corners are not displayed |

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

<class>

FilePoint

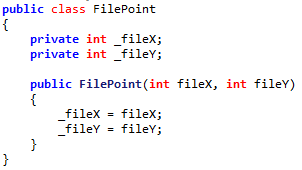
fileX

fileY

GetFilePoint

GetDisplayPoint

GetFileValue



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

<class>

DisplayPoint

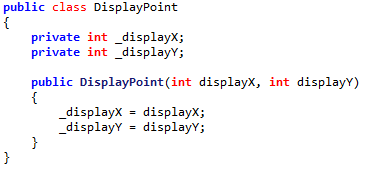
displayX

displayY

GetFilePoint

GetDisplayPoint

GetDisplayValue



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

## IPicturePoint

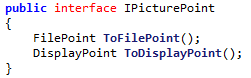
IPicturePoint has been implemented, in accordance to 2.1.6.1

<interface>

IPicturePoint

GetFilePoint

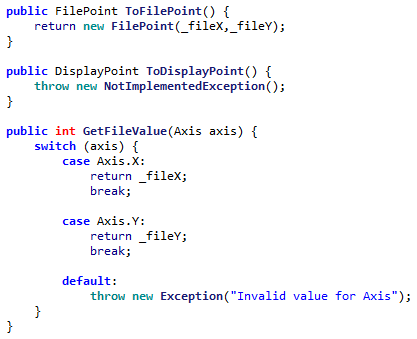
GetDisplayPoint



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

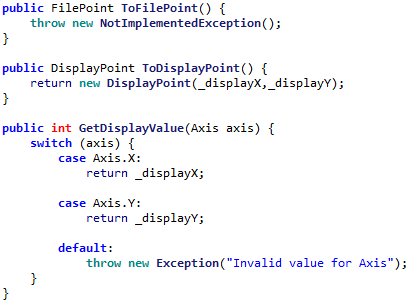
The remaining functions for FilePoint have been implemented:



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:



However the code to convert from a DisplayPoint to FilePoint will be coded in Algorithm 2.8AII

# 26/09/2019 Rectangle Redesign

## IPictureRectangle

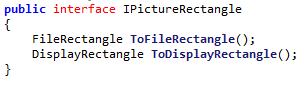
IPictureRectangle has been implemented, in accordance to 2.1.6.4

<interface>

IPictureRectangle

GetFileRectangle

GetDisplayRectangle



## FileRectangle

The FileRectangle class has been implemented in accordance to 2.1.6.5

<class>

FileRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetFileSizeInAxis



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

<class>

DisplayRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetDisplaySizeInAxis



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



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

<class>

Image

colours[,]

width

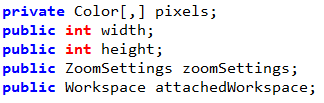
height

zoomSetting

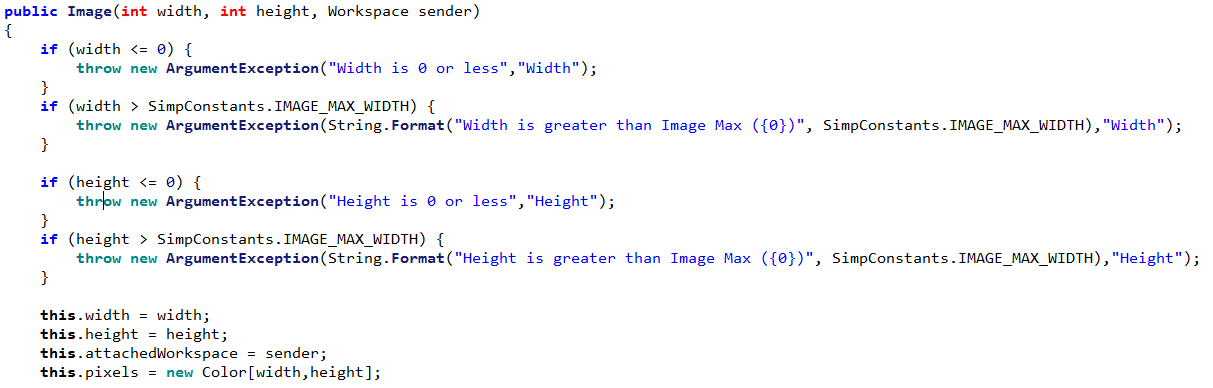
attachedWorkspace

GetPixel()

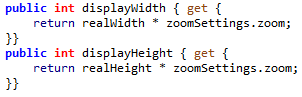
SetPixel()



So thus the constructor has been edited:



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



## Workspace Redesign

Workspace has been upgraded to include the padding parameters and other functions defined by 2.6.1.8

<class>

Workspace

displayPictureBox

attachedForm

displayWidth

displayHeight

LEFT\_PADDING

RIGHT\_PADDING

TOP\_PADDING

BOTTOM\_PADDING

GetPixel

SetPixel

GetDisplaySizeInAxis

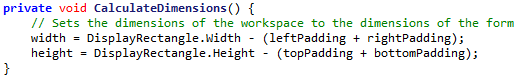
GetDisplayRectangle

Display

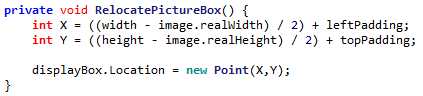
HandleResize

## Implementing Padding

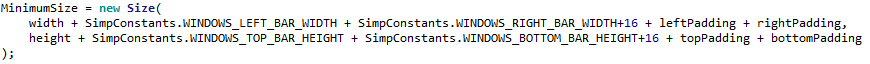
Padding has now been implemented into the relevant calculations, including ones for defining size of displayBox:



Code for determining locations of displayBox:



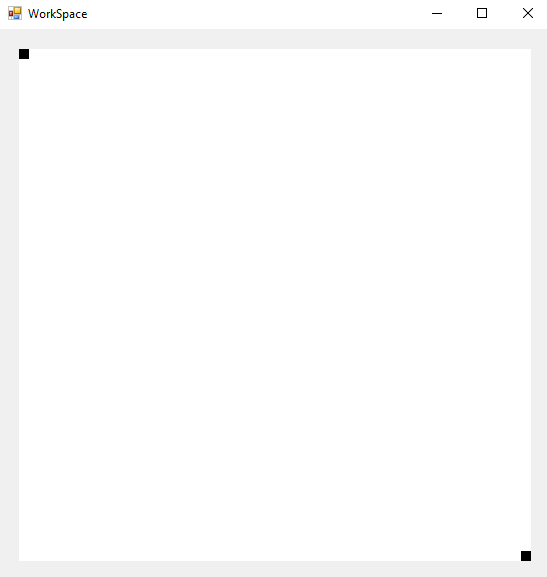
Code for determining minimum size:



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



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 |

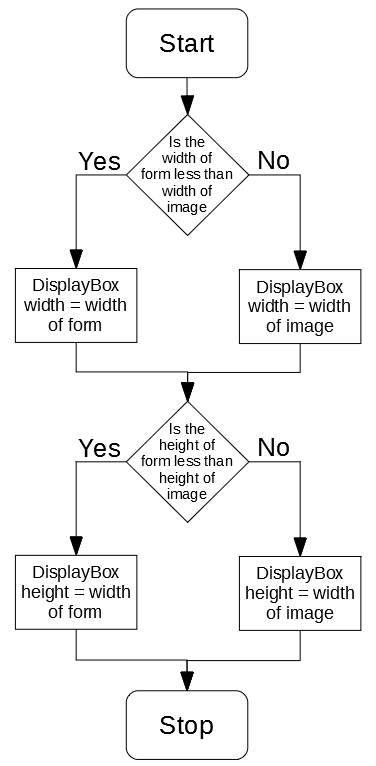
Compared to:



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

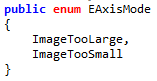
As the zoom is now being implemented, it becomes acceptable for the window to be smaller than the pictureBox. This means that the code for changing the displayBox’s size must be updated to look like:



# 28/09/2019 Implementing CheckImageSize

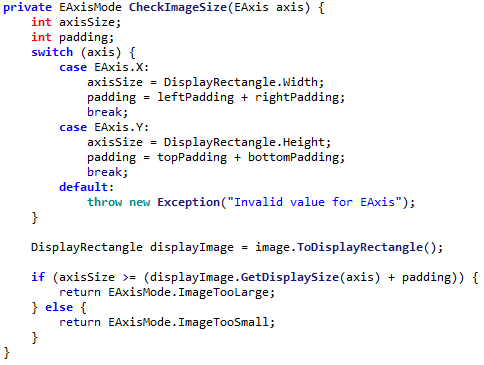
## EAxisMode

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)



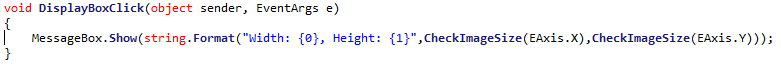
Using this enum, an algorithm for checking whether the image is too large in a specified axis can be implemented:

## Implementing CheckImageSize

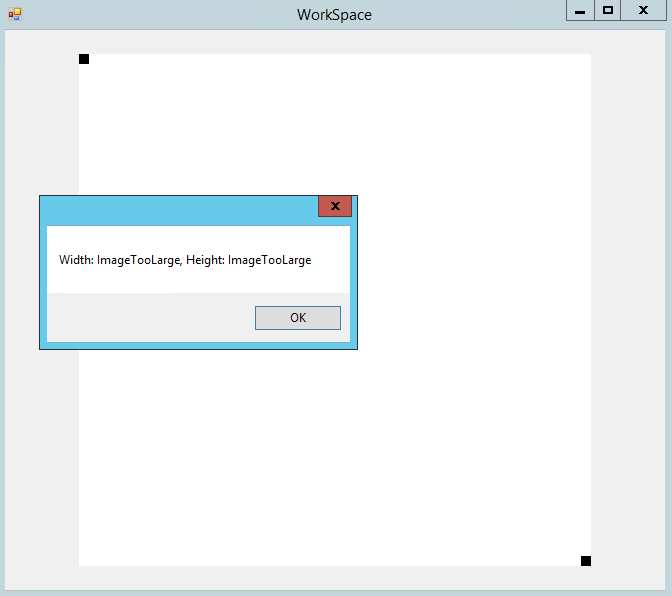


## Alpha Testing CheckImageSize #1

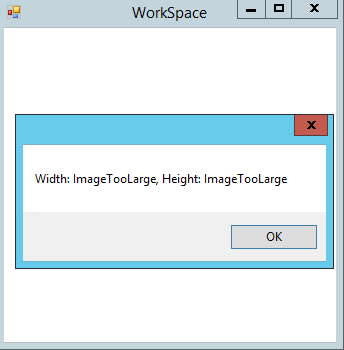
And can be **alpha tested** during development, using the following code:



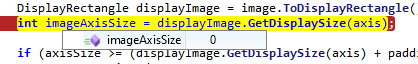
However, this testing reveals a problem, the code returns ImageTooLarge in situations where it should not:



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



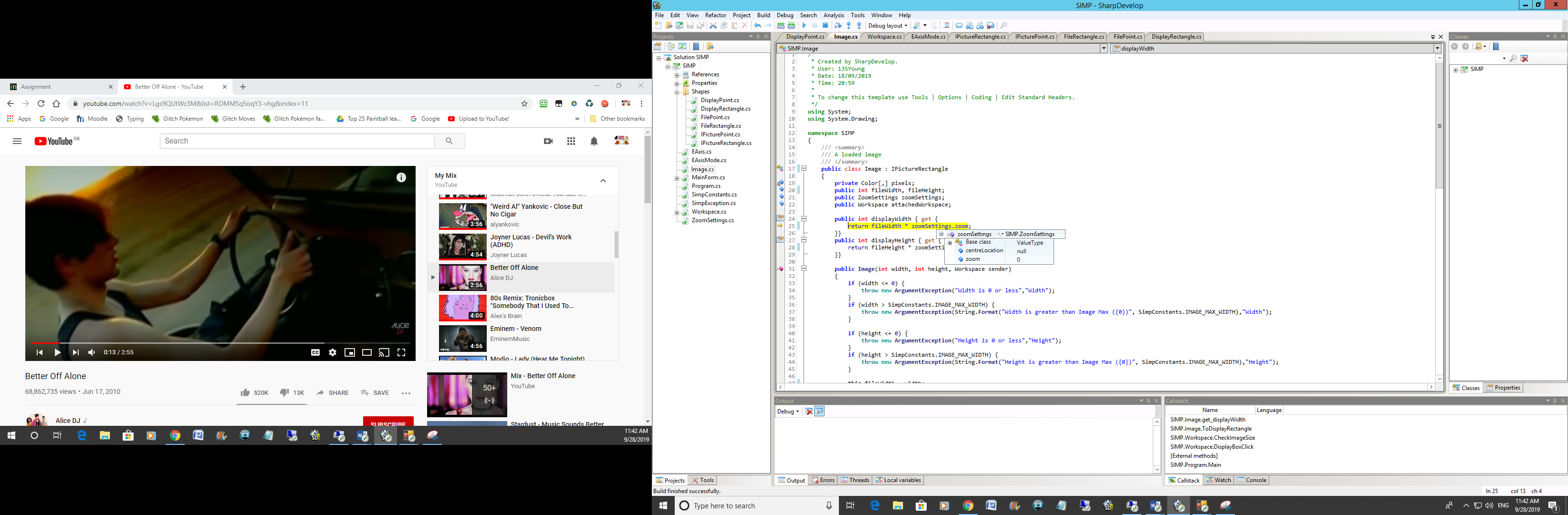
Using a **Break Point** to **debug** the code reveals that:



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: 

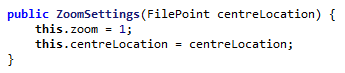
Finally the root of the issue was found, the zoom level was erraneously set to 0:



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

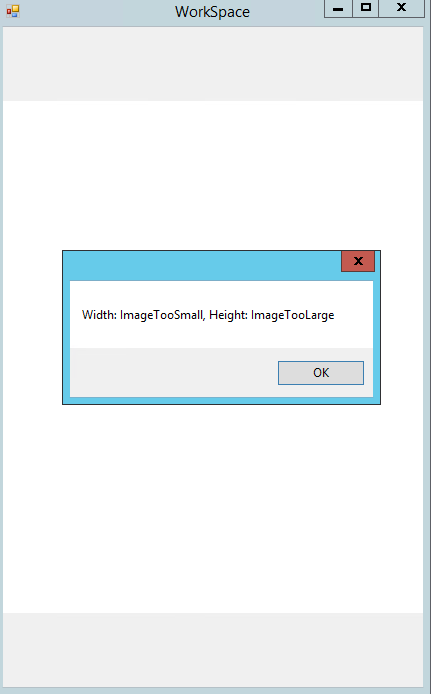
Which causes the class to be populated with null values, which in the case of an integer is 0.

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

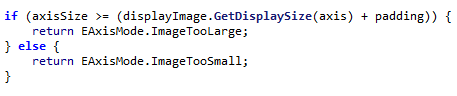


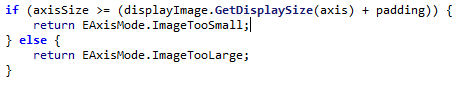
## Alpha Testing CheckImageSize #2

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:



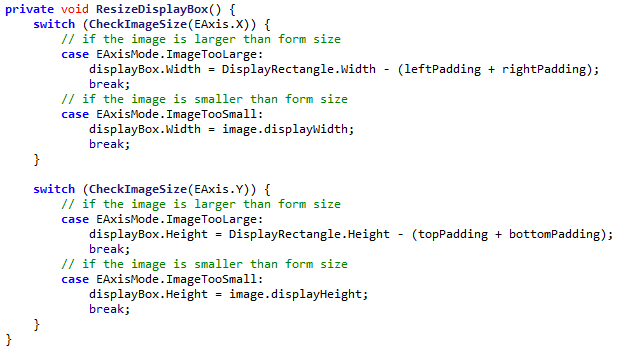


Thus, by **iterative development and bug fixing, the function has been created**.

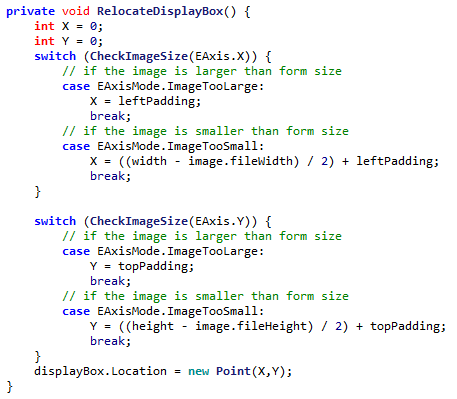
# 29/09/2019 Upgrading displayBox functionality

## Implementing proper resizing and relocating for displayBox

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

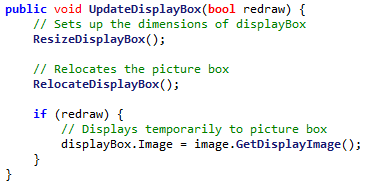


The switch cases take the role of the Diamond shape in the flowchart.

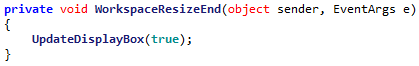
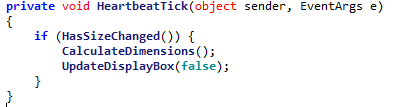


## Implementing new Image Requests

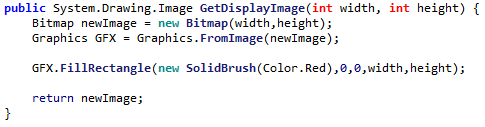
To help save CPU at this time, a new image will only be requested for the displayBox at the end of a Resize. This is enforced by the use of a Boolean when calling UpdateDisplayBox:



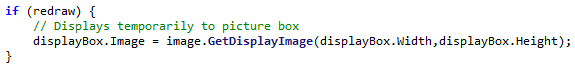
So this means that a new image will only be requested when the function is passed ‘True’. This does **not** happen on a normal resize, but does happen when the resize ends:



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.



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



# 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.

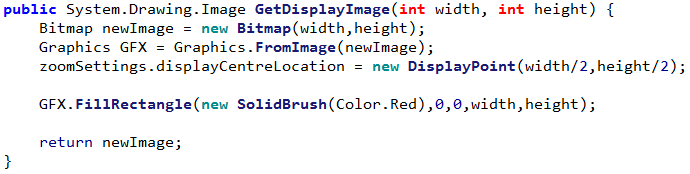
To do this, a new private parameter was added to the image, which contains a reference to the zoom settings of its image:



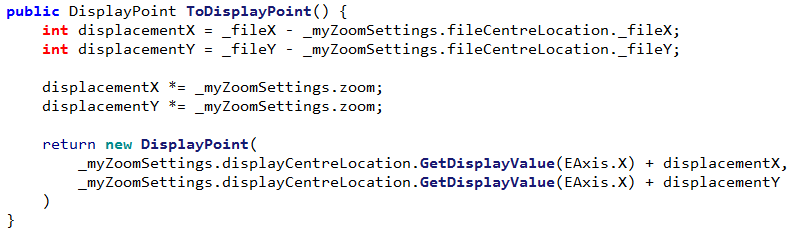
However zoomSettings does not contain any knowledge of where the display centre location is. This means that a parameter for this must be added to ZoomSettings:



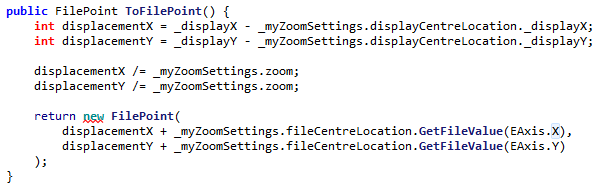
This is updated whenever a new image is requested (whenever the current displayCentreLocation might change).



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



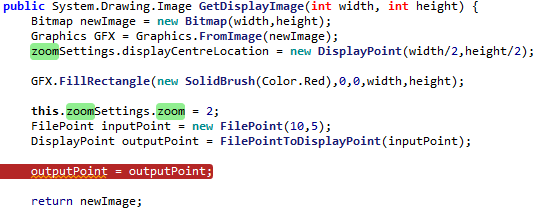
The code for converting from DIsplayPoint to FilePoint can also be implemented, in accordance to 2.8B.



# 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:

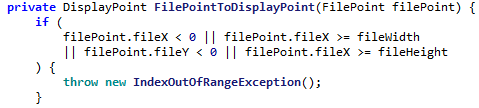


The data from the break point will be inspected to find the properties of the returned point.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |
| DisplayPoint(-1,-1) | 8 | Throws out of bounds error |  | No error is thrown |
| DisplayPoint(20,10) | 9 | Throws out of bounds error |  | No error is thrown |

### Fixing Error #8 & #9

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



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| DisplayPoint(-1,-1) | 8 | Throws out of bounds error | System.IndexOutOfRangeException: Index was outside the bounds of the array. | No error is thrown |
| DisplayPoint(20,10) | 9 | Throws out of bounds error | 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 |
| FilePoint(5,5) | 1 | (10,5) |  | The centre location is returned |
| FilePoint(1,1) | 2 | (8,3) |  | The correct point is returned |
| FilePoint(12,7) | 3 | (9,9) |  | The correct point is returned |
| FilePoint(-1,-1) | 4 | (7,2) |  | The correct point is returned |
| FilePoint(11,11) | 5 | (13,8) |  | The correct point is returned |
| FilePoint(-15,-5) | 6 | (0,0) |  | The correct point is returned |
| FilePoint(23,13) | 7 | (19,9) |  | The correct point is returned |
| FilePoint(6,6) | 8 | (10,5) |  | The point is rounded up |
| FilePoint(4,4) | 9 | (9,4) |  | The point is not correctly rounded down |
| FilePoint(-17,-7) | 10 | This point is rejected | System.IndexOutOfRangeException: Index was outside the bounds of the array. | The point is correctly rejected |
| FilePoint(25,15) | 11 | This point is rejected | System.IndexOutOfRangeException: Index was outside the bounds of the array. | The point is correctly rejected |

### Fixing Issue #9

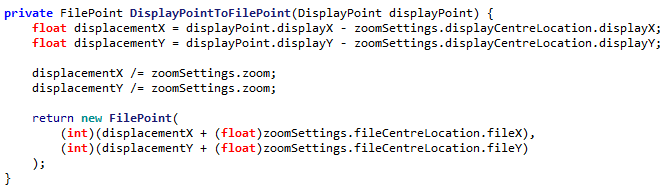
The issue in error #9 occurs because, in this situation, the displacement has a negative value. When this displacement is divided and then rounded, it is rounded **up**, as this is how negative numbers are implemented.





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.



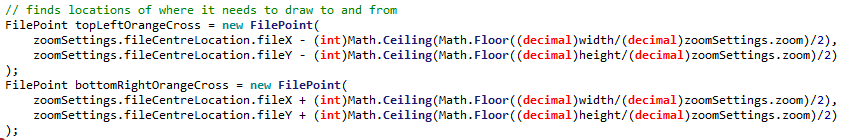
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| FilePoint(4,4) | 9 | (9,4) |  | The point is now correctly rounded down |

# 3/10/2019 Displaying an Image

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

## Algorithm 2.8B – Determining Border Locations

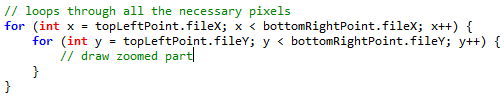
The algorithm for determining the border location has been implemented, in accordance to 2.8B



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

## Algorithm 2.8C – Finding Green Pixels

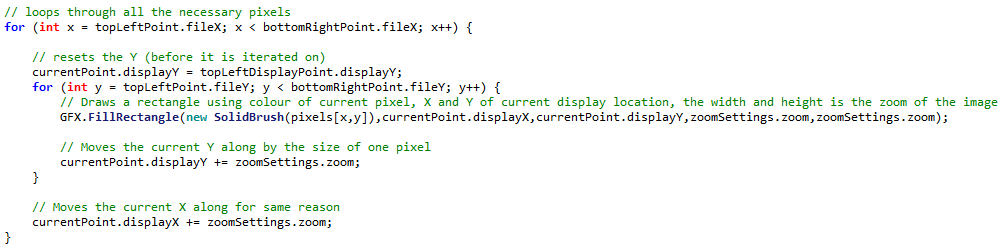
The algorithm for determining pixels to draw has been implemented, in accordance to 2.8C



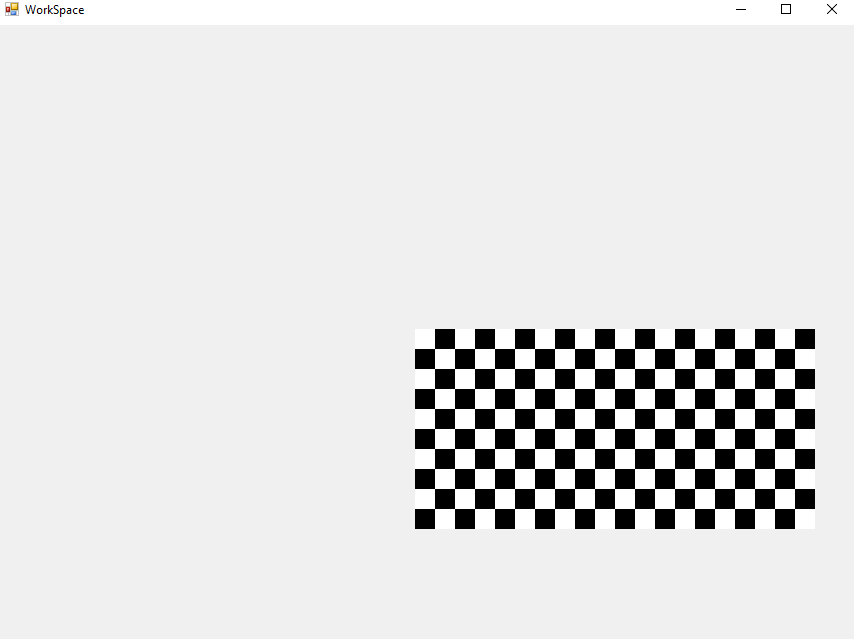
Where the comment will be replaced with the code necessary to draw the zoomed part

## Algorithm 2.9 – Draw Green Pixels

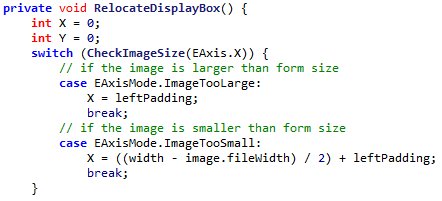
The above code has been upgraded to include drawing functionality:



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



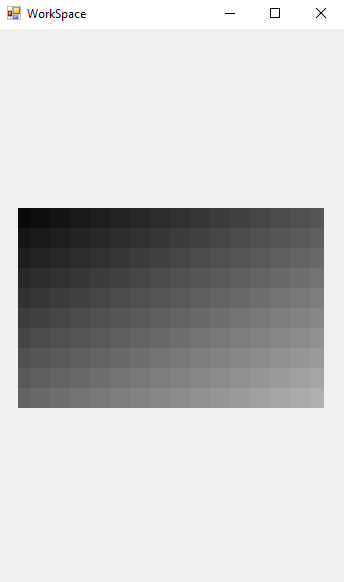
The cause of this was simple, fileWidth was erroneously used instead of displayWidth:



Changing this to displayWidth resolves the error.

**SIMP can now display images:**

****

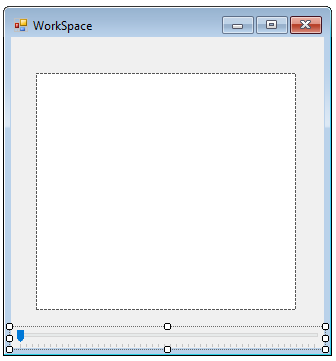
****

****

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

## Adding Zoom Bar to GUI

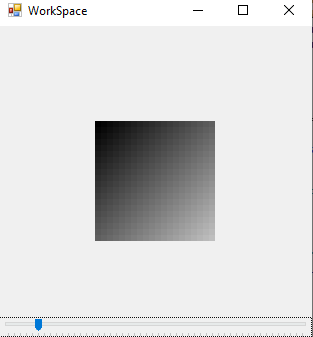
The zoom bar can be added quite simply, filling the border area at the bottom.

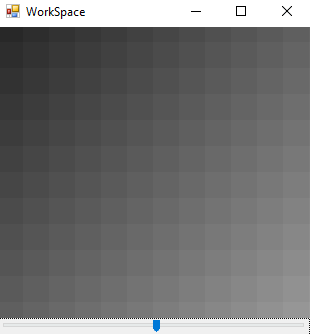


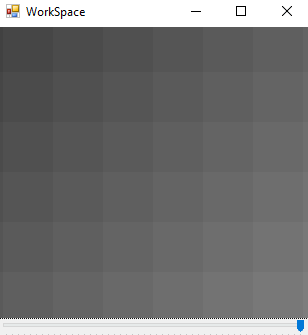
Its value can also be interpreted quite simply. Its minimum and maximum has been set to 1 and 50 respectively. When its value is changed, the image’s zoom is updated and the box is redrawn.



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

 Small amount of zoom

 Large amount of zoom

 Maximum zoom

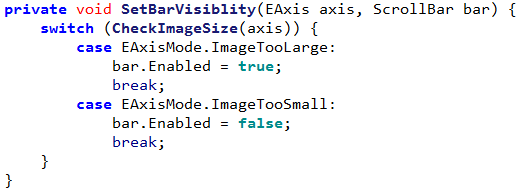
# 04/10/2019 Implementing Bars

## Determining whether Bars are visible

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

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:



## Algorithm 2.11A Determining Crosses in Axis

This code has been implemented, in accordance to Algorithm 2.11A

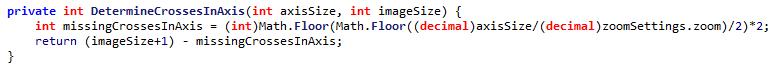
DetermineCrossesInAxis(Axis axis) {

MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)\*2

MaxCrossesInAxis = image.getSizeInAxis(axis)

return MaxCrossesInAxis – MissingCrossesInAxis

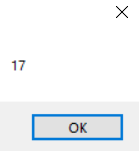
}



When testing the code with the following inputs:



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:

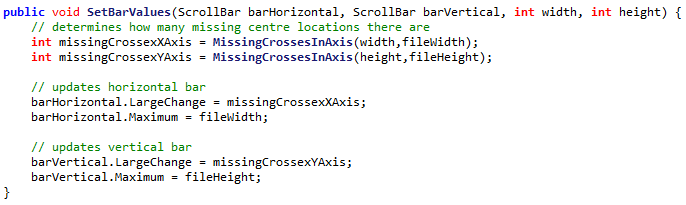
UpgradedSetupBarSize(progressBar, Axis) {

MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)\*2

progressBar.barSize = MissingCrossesInAxis

progressBar.max = image.getFileWidth(Axis)

}



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)

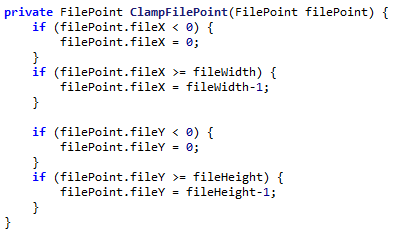
RETURN bar.value + firstCrossPosition

}

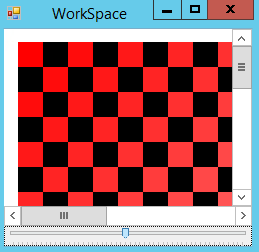
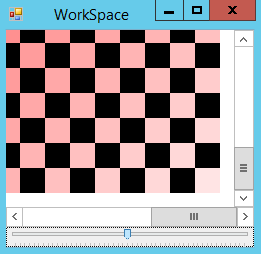
**Insert screenshot here since my laptop is too small to view it properly**

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



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

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

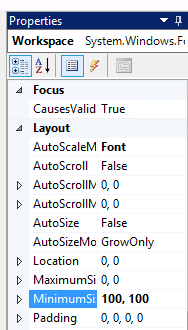
## Unit Test 2.11 #1

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. | System.ArgumentException: Parameter is not valid. | 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 |
| 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 |
| Horizontal scroll bar is moved far left, then form size increased | 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 |
| Form is resized to smallest, then maximized | 9 | The bars disappear and the view is normal | System.ArgumentException: Parameter is not valid. | An error is thrown as the image becomes too small to display. |

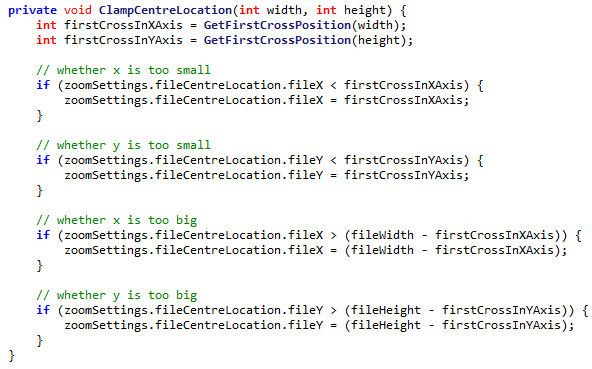
### Fixing Error #2 & #9

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:

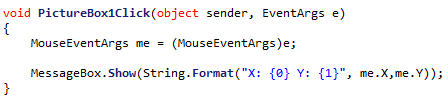


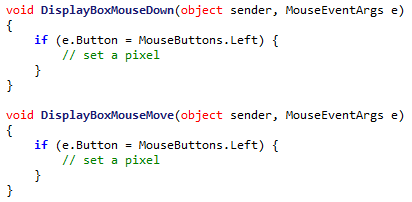
## Unit Test 2.11 #2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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. |  | The minimum form size prevents form from getting too small |
| Form is resized back to half size of image | 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 |
| Horizontal scroll bar is moved far left, then form size increased | 8 | Centre locations is moved to the left when needed |  | The centre location of the image now correctly updates to the centre of the image, resolving the invalid bar value |
| Form is resized to smallest, then maximized | 9 | The bars disappear and the view is normal |  | The system copes with this sudden change of size. |

## Algorithm 2.13 Detecting Mouse Clicks

This algorithm has been implemented, in accordance to Algorithm 2.13 demonstration code:

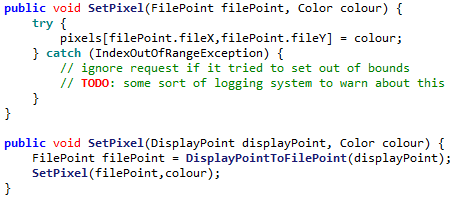




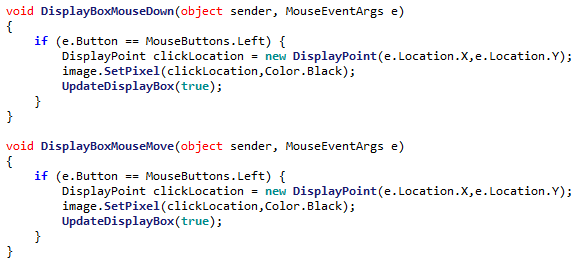
However this implementation uses two events rather than one, for when the mouse is held down and moved.

## 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:



And then adding a call from WorkSpace:



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

2.3 Testing

# 2.3.1 Full Testing

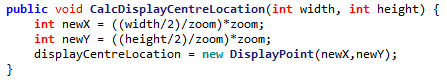
## Initial Testing Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | ID | | Expected Result | Actual Result | Comment |
| new Image(10,10) | | 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 |
| new Image(1,1) | | 6 | A 1fpx, 1fpx image |  | A very small picture is created, but it is not centrally aligned in the image. |
| new Image(10,0) | | 7 | The parameters are rejected and no image is created |  | A height of 0 is not accepted, and autocorrects to 1 |
| new Image(0,10) | | 8 | The parameters are rejected and no image is created |  | A width of 0 is not accepted, and autocorrects to 1 |
| new Image(0,0) | | 9 | The parameters are rejected and no image is created |  | Both boxes are checked independent of each other so this poses no new issue |
| new Image(100000,10) | | 10 | The parameters are rejected and no image is created |  | The width is autocorrected back down to the allowed maximum |
| new Image(10,100000) | | 11 | The parameters are rejected and no image is created |  | The height is autocorrected back down to the allowed maximum |
| new Image(-1,10) | | 12 | The parameters are rejected and no image is created |  | The invalid width is corrected |
| new Image(10,-1) | | 13 | The parameters are rejected and no image is created |  | The invalid height is corrected |
| new Image(10) | | 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. |  |
| SetPixel(0,0,Black) | | 18 | A pixel in the top-left corner is set to black |  | A pixel at the top left is set |
| SetPixel(9,9,Black) | | 19 | A pixel in the bottom-right corner is set to black |  |  |
| SetPixel(-1,0,Black) | | 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 |
| SetPixel(0,-1,Black) | | 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 |
| SetPixel(10,0,Black) | | 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 |
| SetPixel(0,10,Black) | | 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 |
| Form is started at normal size | | 24 | Image Displays in the middle of the form |  |  |
| Form is maximized | | 25 | Image displays in the middle of the large form |  | The image (while small) is correctly displayed in the middle of the large form |
| Form is minimized | | 26 | No image is displayed (as form is currently invisible) | No image | There is no problem when the program is minimized |
| Form is resized to smallest possible | | 27 | 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 |
| Form is resized to minimum width maximum height | | 28 | A very thin form displays the image |  | A very wide form correctly displays the image in its centre |
| Form is resized to minimum height maximum width | | 29 | A very short form displays the image |  | A very tall form displays the image, but the top control bar is still displayed incorrectly |
| The form’s size is rapidly changed. | | 30 | The image is very quickly moved around but remains centred | No image, but program remains stable | The program can be very quickly manipulated but still appear smooth |
| Form is resized to half the size of image | | 31 | Bar is half size of bounds and in its centre position |  | The bar is in its correct position |
| Form is resized to its smallest position | | 32 | Bar is small but still central. |  | The bar still remains close to the centre of the image |
| Form is resized back to half size of image | | 33 | Bar is half size of bounds and in its centre position |  |  |
| Horizontal Scroll bar is moved to far left | | 34 | The far left of the image is displayed, but no more |  | The far left is displayed correctly |
| Horizontal Scroll bar is moved to far right | | 35 | The far right of the image is displayed, but no more |  | The far right is displayed correctly |
| Vertical Scroll bar is moved to highest | | 36 | The highest of the image is displayed, but no more |  | The top right is now correctly displayed |
| Vertical Scroll bar is moved to lowest | | 37 | The lowest of the image is displayed, but no more |  | The bottom right is now correctly displayed |
| 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:  


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:  


So, performing the tests again:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| new Image(5,10) | 3 | A 5fpx, 10fpx image |  | A tall image is created, and now displays its needed pixels correctly. |
| new Image(10,1) | 4 | A 10fpx, 1fpx image |  | A very thin image is created |
| new Image(1,10) | 5 | A 1fpx, 10fpx image |  | A very tall image is created |
| new Image(1,1) | 6 | A 1fpx, 1fpx image |  | A very small image is created |

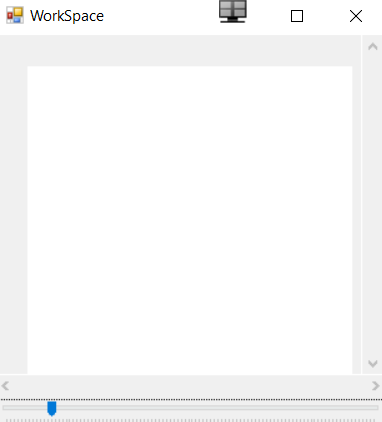
This now means that SIMP has passed **Alpha Testing**. It will now be handed to the Stakeholders for their initial thoughts and **Beta Testing**.

# 2.3.2 Client Testing & Feedback

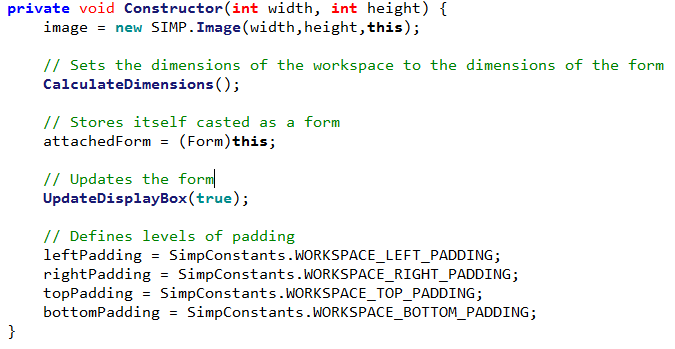
## Alex G Client Feedback

### Alignment at startup issue

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

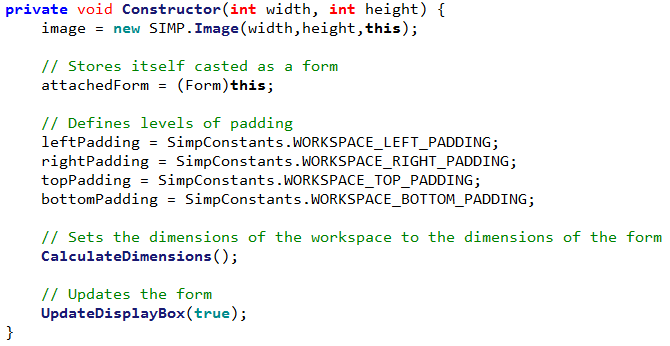


The error with the code happens because of the way the constructor is ordered:

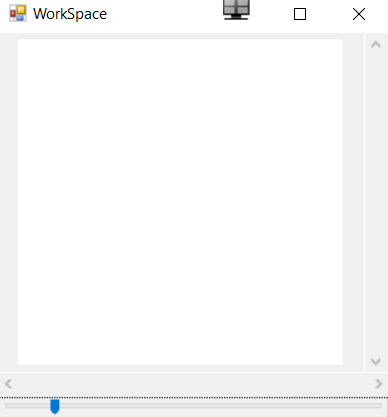


As the dimensions and updates are done *before* the padding is defined, so the program initially displays as if the padding is 0

A simple code re-order resolves this:



So the program now starts correctly.



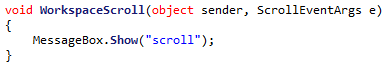
However, he was able draw an image 

### 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.

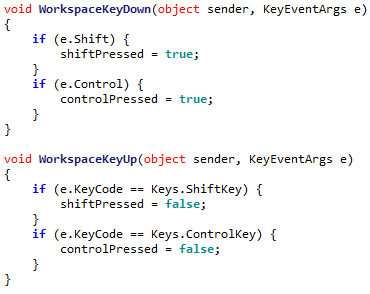
He suggested that scrolling normally should move the vertical bar upwards, pressing shift+scroll should move the horizontal bar, and pressing control+scroll should zoom in or out.

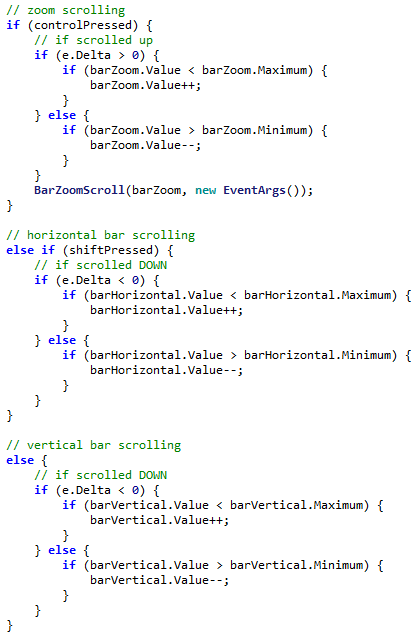
However, a first attempt to code this proved unsuccessful:



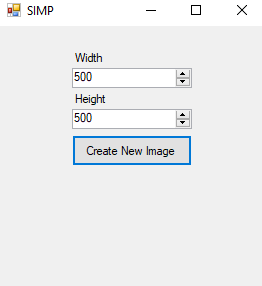
The MessageBox did not show. This was because the ‘scroll’ event is designed for controls that have an attached scroll bar. The WorkSpace did not have a scroll bar directly attached to it so did not trigger this event.

This can be solved by using the MouseWheel event:  


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

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


### 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:



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.



This then means the drawing code can be implemented as:



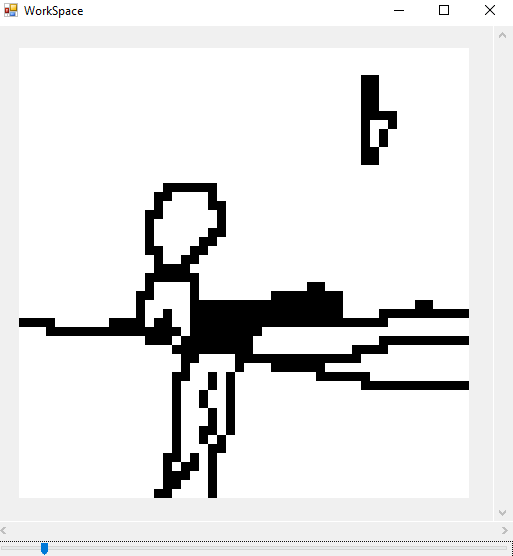
Which drastically reduces the code complexity, and increases performance.

## 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



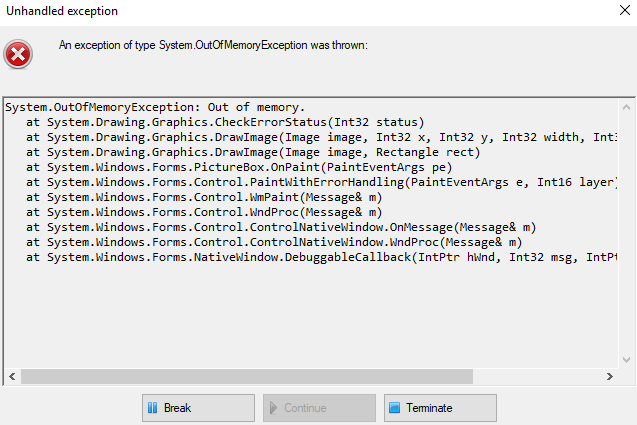
### Inconsistent Brush

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



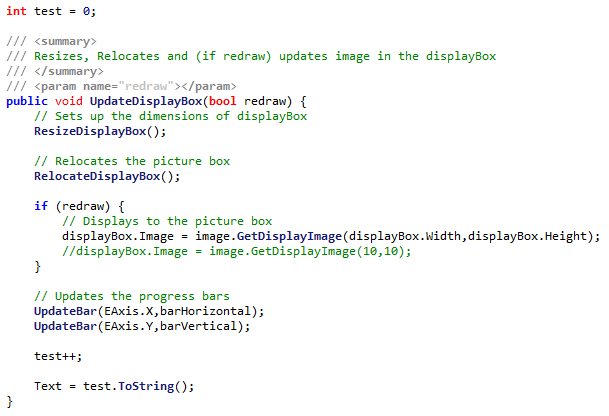
### Memory Leak Crash

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

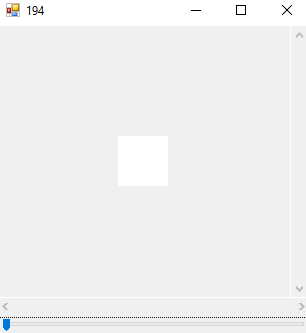
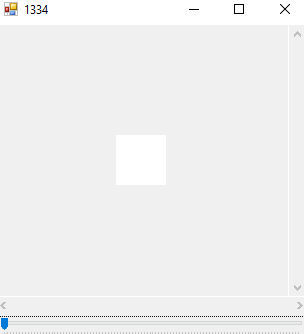


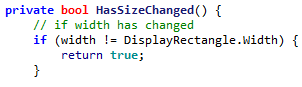
The error occurred because each time a new image is created, the old image is not removed from memory. This means if too many images are created at the same time then they are made faster than the garbage collector can delete them, and the program crashes.

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



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

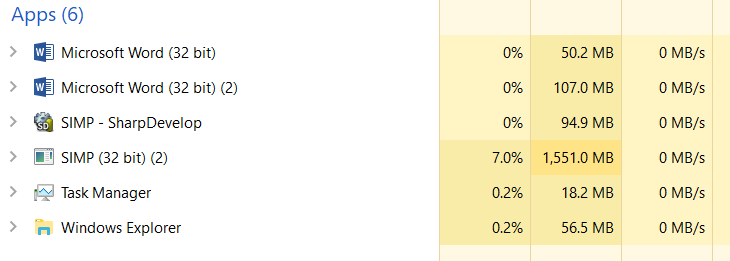
 🡪 

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

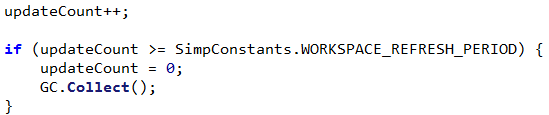
However was being set to the size of the form factored with padding:  

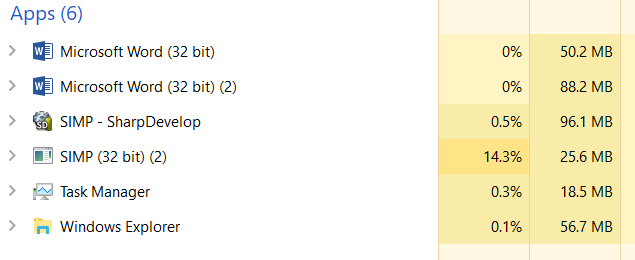

This meant that HasSizeChanged always returned true, and the image was updated many extra times per second.

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



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:  


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


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 | Screenshot of a currently being viewed image | B1 |
| Bitmap image editor | Screenshot of a zoom in on the image showing the pixels | B2 |
| 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 |

So this makes the full diagram:

Not completed

To be done this section

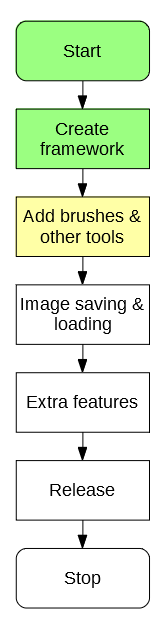
Completed

|  |  |  |
| --- | --- | --- |
| **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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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 3 – Brushes & Tools

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

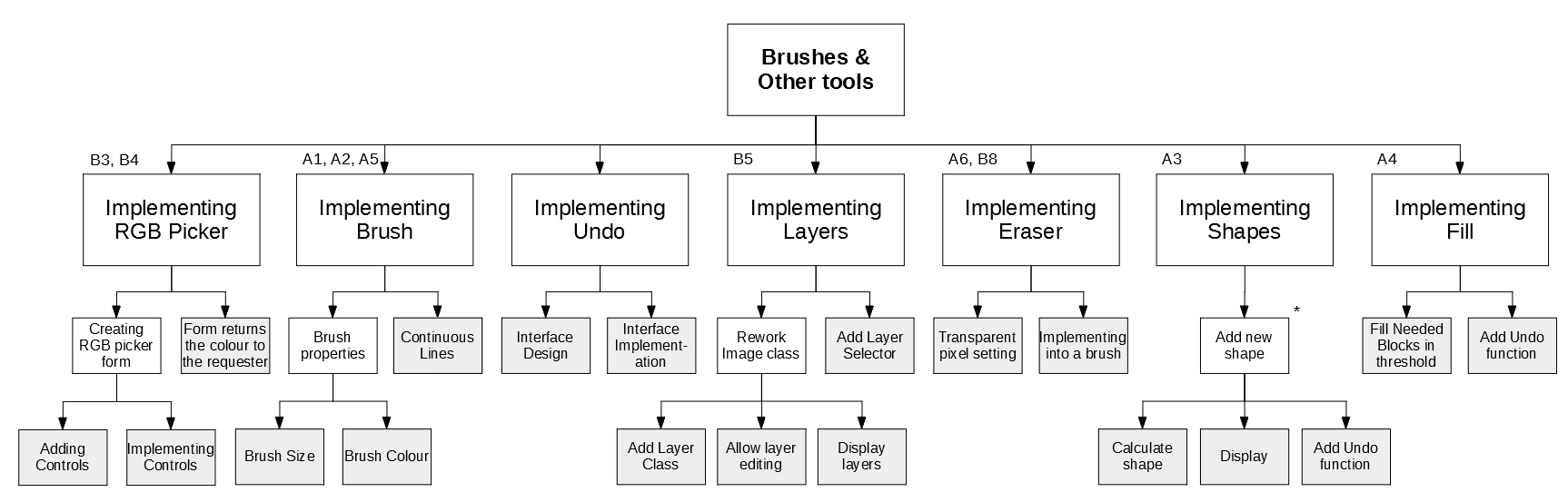
To be done this section

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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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 |

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

# 3.1.2 Section Decomposition



19

18

17

16

15

14

13

12

11

10

9

8

7

6

3

5

1

2

4

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

<interface>

IAction

Do()

Undo()

### 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

<class>

PixelAction

newPixels

oldPixels

Do\*

Undo\*

AddPixel()

### 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

<class>

ITool

name

description

properties

HandleMouseDown

HandleMouseUp

HandleMouseClick

HandleMouseMove

GetProperty

### 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

<interface>

IToolProperty

name

### 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

<class>

NumericalProperty

name\*

value

min

max

### 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

<class>

ColorProperty

name\*

value

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| name\* | String | Inherited from IToolProperty |
| value | Color | Stores the value of the property |

## 3.1.3.7 LineTool

### Class Diagram

<class>

LineTool

name\*

description\*

properties\*

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

GetProperty\*

DrawLine

### 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

<class>

LineTool

pixels

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| pixels | Array of Colours | Stores the colours needed in this class |

# 3.1.4 Class Relation Diagram

Other actions in future

<class>

PixelAction

newPixels

oldPixels

Do\*

Undo\*

AddPixel()

<interface>

IAction

Do()

Undo()

<class>

ITool

name

description

properties

HandleMouseDown

HandleMouseUp

HandleMouseClick

HandleMouseMove

GetProperty

<class>

LineTool

name\*

description\*

properties\*

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

DrawLine

Other tools in future

<class>

ITool

name

description

properties

HandleMouseDown

HandleMouseUp

HandleMouseClick

HandleMouseMove

GetProperty

<class>

ColorProperty

name

value

Other properties in future

<class>

NumericalProperty

name

value

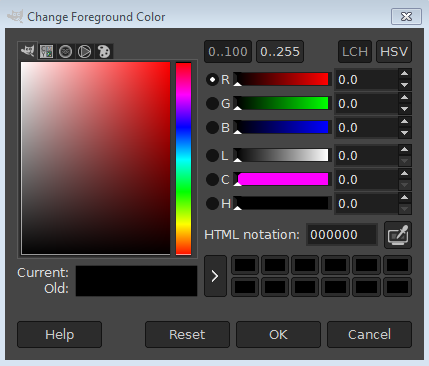
min

max

# 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

#FF0000

### 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

Recent

255

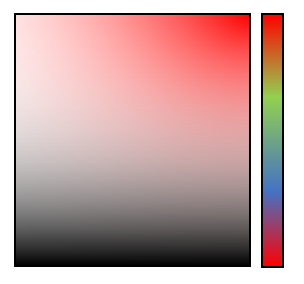
0

0

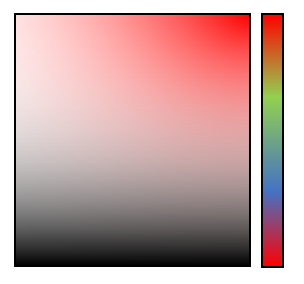
#FF0000

## 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:



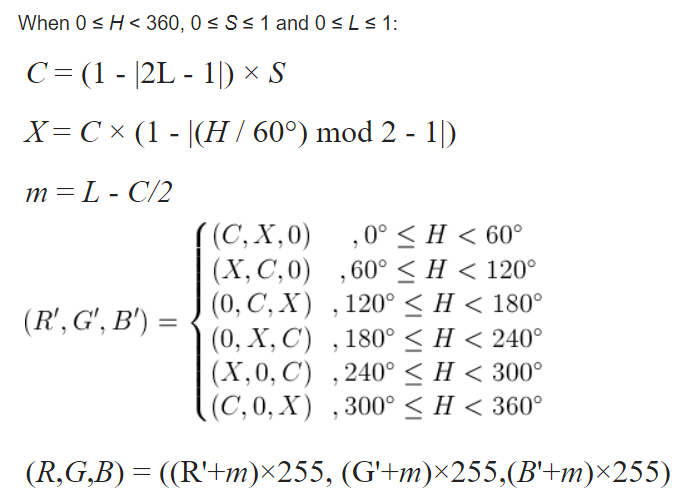
Light

Saturation

Hue

### 3.2B Converting HSL to RGB

The algorithm for converting between HSL and RGB is



#### Pseudocode

Implementing the above diagram as pseudocode gives:

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

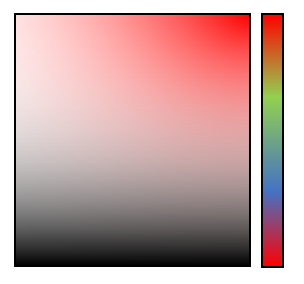
B = (b + m) \* 255

RETURN new Colour(R,G,B)

}

### 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 |
| Set Hue to Red, click top-left corner | 1 | White | This tests that the top left corner is functional |
| Click top-right corner | 2 | Red | This tests that the top left corner is functional |
| Click bottom-left corner | 3 | Black | This tests that the bottom left corner is functional |
| Click bottom-right corner | 4 | Black | This tests that the bottom right corner is functional |
| Click three random positions on the colour square | 5 | Correct corresponding colour outputs | This tests that other areas on the colour square are valid |
| Click three random positions on the hue slider | 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 |
| Set RGB values to 255,0,0 | 7 | Red, hue adjusts appropriately | This tests that a standard colour can be implemented |
| Set RGB values to 255,255,255 | 8 | White | This tests that a maximum brightness colour can be implemented |
| Set RGB values to 0,0,0 | 9 | Black | This tests that the darkest colour can be entered |
| Set RGB values to -1,0,0 | 10 | Not accepted, round -1 to 0 | This tests that an incorrect value is adjusted up appropriately |
| Set RGB values to 256,255,255 | 11 | Not accepted, round 256 to 255 | This tests that an incorrect value is adjusted down appropriately |
| Set RGB values to 255,255 | 12 | Not accepted, missing value substituted for 0 | This tests that if a value is left out, it will be subsituted |

## 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 RBGPicker(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 coordinated 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 original can be calculated (x2 +y2), and if it is less than the brushSize2, then the point is inside the circle.

So again coming back to the size 2 brush:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 1.5² + 1.5² = 4.5  4.5 > 4 so False |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 1.5² + 0.5² = 2.5  2.5 < 4 so True |  |
|  |  |  |  |  |  |  |  |
|  |  | 0.5² + 0.5² = 0.5  0.5 < 4 so True |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

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 the second pixel is further left than the first, they are swapped. This decreases the amount of potential point configurations

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

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](#_Methods)(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 |
| Draw a 1 width line across middle of image | 1 | A line appears and is displayed on the image | This tests that a normal line can be drawn on the image |
| Draw a 1 width line at the edge of the image | 2 | A line is drawn at the edge | This tests that a normal line can be drawn at the edge of the image |
| Draw a 4 width line across middle of image | 3 | A thicker line is displayed on the image | This tests that a thick line can be drawn on the image |
| Draw a 4 width line at the edge of the image | 4 | A thicker line is displayed at the edge | This tests that a thick line can be drawn at the edge of the image |
| Move the mouse rapidly drawing a 1 width line | 5 | The drawn line has no visible breaks | This tests whether the filled line algorithm works correctly |
| Move the mouse rapidly drawing a 4 width line | 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) {

The purpose of these two data structures will be explained in 3.8B

pastActions.add(action)

futureActions.flush()

}

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 implent 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:

LineAction2

LineAction3

futureActions

pastActions

LineAction1

Then if the user undoes two of those lines then two actions should be moved into the future:

LineAction1

futureActions

pastActions

LineAction3

LineAction2

Then if the user chooses to redo a line the first action from the future is popped

LineAction2

LineAction1

futureActions

pastActions

LineAction3

Finally, if the user makes a new line the futureActions stack is cleared, it’s no longer relevant.

LineAction4

LineAction2

LineAction1

futureActions

pastActions

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 |
| Undo an action (on default image) | 1 | Cannot be done | Tests that an action cannot be undone if no action has been perfoemd |
| Redo an action (on default image) | 2 | Cannot be done | Tests that an action cannot be redone if there is no future |
| Perform an action | 3 | Action is performed onto image | Tests that actions can still be performed |
| Undo the action | 4 | Action is undone | Tests that an action can be undone |
| Redo the action | 5 | Action is redone | Tests that an action can be redone |
| Undo the action | 6 | Action is undone | Tests that a previously redone action can be undone |
| Redo the action | 7 | Action is redone | Tests that a previously undone action can be redone |
| Perform another action | 8 | Action is done | Tests that actions can still be done from here |
| Undo both actions | 9 | Both actions redone | Tests that multiple actions can be redone |
| Undo an action | 10 | Cannot be done | Tests that the program stops undoing when there are no previous actions |
| Redo both actions | 11 | Both actions redone | Tests that both actions can be redone |
| Redo an action | 12 | Cannot be done | Tests that the program stops redoing when there is no future |
| Undo an action, and preform a new action | 13 | New action is performed on top of old action | Tests that actions can be performed on a previously undone action |
| Redo an action | 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](#_3.1.3.5_Layer)

<class>

LineTool

pixels

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:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Layer 2

Layer 3

Layer 1

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



Layer 1

Layer 2





Layer 3

### 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



Layer 1

Layer 2





Layer 3



### 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 |
| Select the top layer and draw onto it | 1 | There is drawing on layer | This tests whether a layer can be drawn onto |
| Attempt to delete the layer | 2 | The layer cannot be deleted as it is the only one | This tests whether the program is prevented from having 0 layers |
| 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 | This tests whether the user can tell what layer they are drawing onto |
| Move lower layer up | 4 | The lower layers moves on top of previous top layer | This tests whether the drawing adjusts when the layers are moved |
| Move the highest layer up | 5 | Should be impossible as it cannot go up | This tests whether the highest layer cannot be moved too high |
| Move the lowest layer down | 6 | Should be impossible as it cannot go down | This tests whether the lowest layer cannot be moved too low |
| Add a new layer | 7 | A new (empty) layer is created | This tests whether a new layer can be made |
| Add 10 new layers | 8 | Many new layers are added | This tests whether the system can handle many layers |
| Layers are labelled then randomly rearranged | 9 | The layers are moved | This tests whether the program can handle moving many layers |
| The 11 layers are deleted | 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 |
| The final layer is deleted | 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:

<class>

IPixelTool

name\*

description\*

properties\*

shapePoints

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

GetProperty\*

DrawShape

SetShapePixel

AddShape

<class>

IShapeTool

name\*

description\*

properties\*

point1

point2

shapePoints\*

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

GetProperty\*

DrawShape\*

SetShapePixel\*

AddShape\*

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

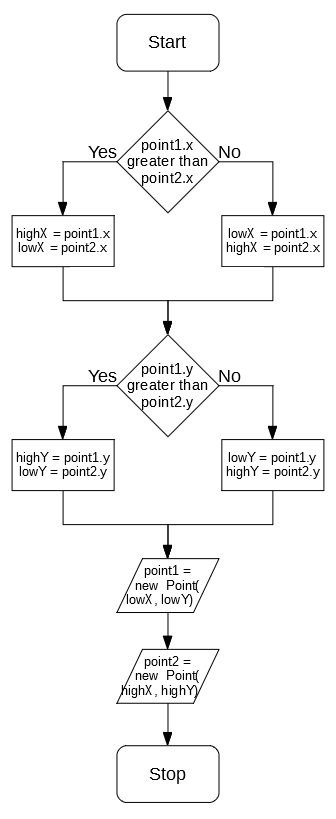
#### Resolving points

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:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | Point1  Point2 |  |  |  |  | Point1  Point2 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | Point2  Point1 |  |  |  |  | Point2  Point1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

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](#_3.6A_Bresenham’s_Line), 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

}

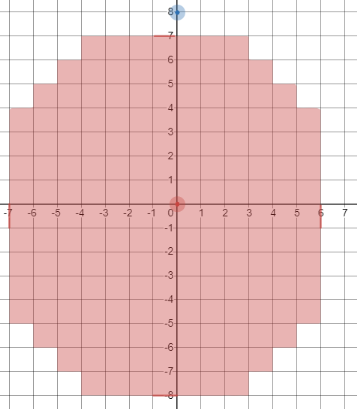
### 3.15D Circle Drawing

Circle drawing is a more in-depth process, but can be implemented reasonably simply using a similar process to [Algorithm 3.4](#_3.4B_Constructing_current). All potential candidate points are checked as to whether they appear in the circle (or ellipse) generated, if so, draw it.

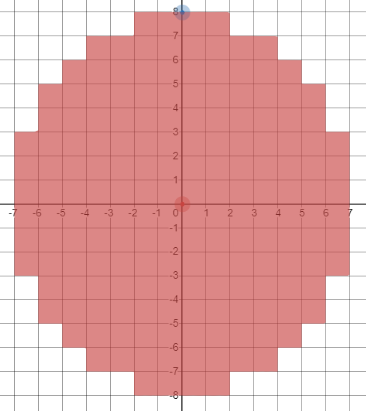
This means a condition for an ellipse will need to be defined. In order to do this a standard representation can be used:

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 |
| Create circle with points (5,5) & (10,10) | 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 |
| Create circle with points (5,10) & (10,5) | 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 |
| Create circle with points (10,5) & (5,10) | 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 |
| Create circle with points (10,10) & (5,5) | 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 |
| Create circle with points (5,5) & (10,8) | 21 | A shorter circle is drawn | This tests that a shorter, non-square circle can be drawn |
| Create circle with points (5,5) & (8,10) | 22 | A thinner circle is drawn | This tests that a thinner, non-square circle can be drawn |
| Create circle with points (5,5) & (10,5) | 23 | A single row is drawn | This tests that a shape can be drawn that has same Y |
| Create circle with points (5,5) & (5,10) | 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](#_3.1.3.2_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:

<class>

LineTool

<class>

RectangleTool

<class>

CircleTool

<class>

IShapeTool

<class>

FillTool

<class>

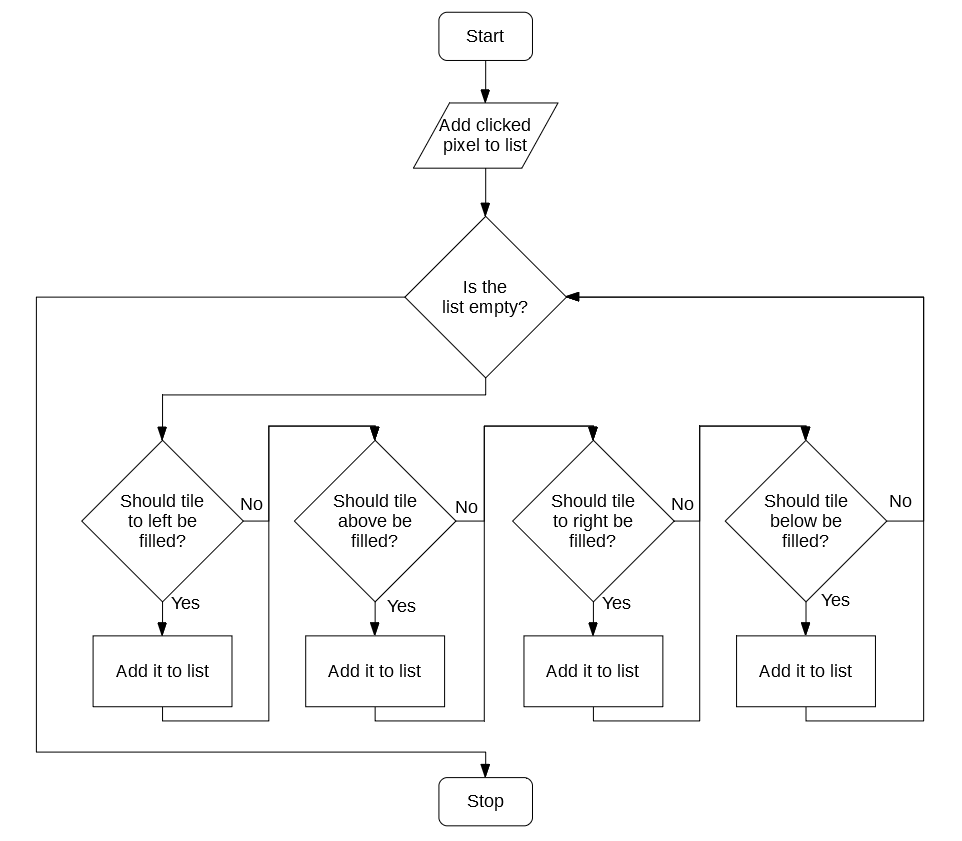
IPixelTool

<class>

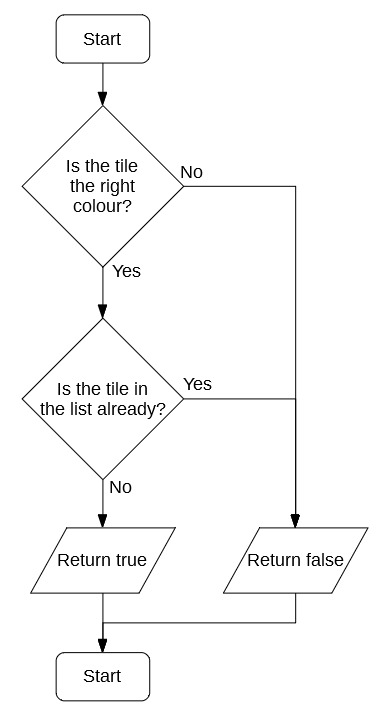
ITool

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:

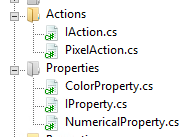


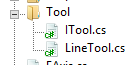
Stop

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

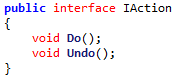
IAction has been implemented with its two functions in accordance to 3.1.3.1:

<interface>

IAction

Do()

Undo()



## PixelAction

PixelAction has been partly implemented, its Do() and Undo() will be implemented later in development, in accordance to 3.1.3.2:

<class>

PixelAction

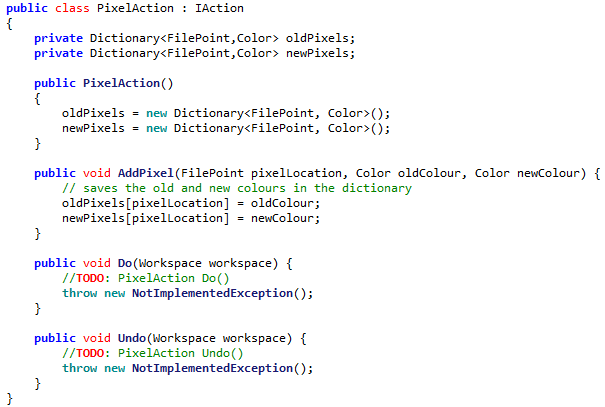
newPixels

oldPixels

Do\*

Undo\*

AddPixel()



## 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:

<class>

ITool

name

description

properties

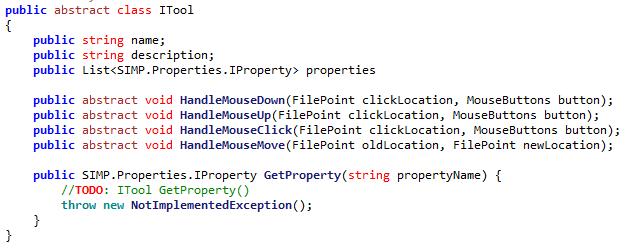
HandleMouseDown

HandleMouseUp

HandleMouseClick

HandleMouseMove

GetProperty



## 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:

<interface>

IToolProperty

name



## NumericalProperty

NumericalProperty has been implemented in accordance to 3.1.3.5:

<class>

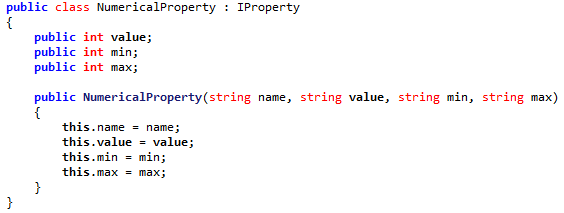
NumericalProperty

name

value

min

max



(Name is inherited from IProperty)

## ColorProperty

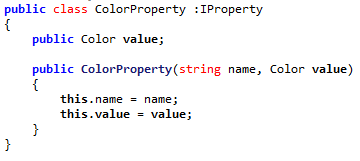
NumericalProperty has been implemented in accordance to 3.1.3.6:

<class>

ColorProperty

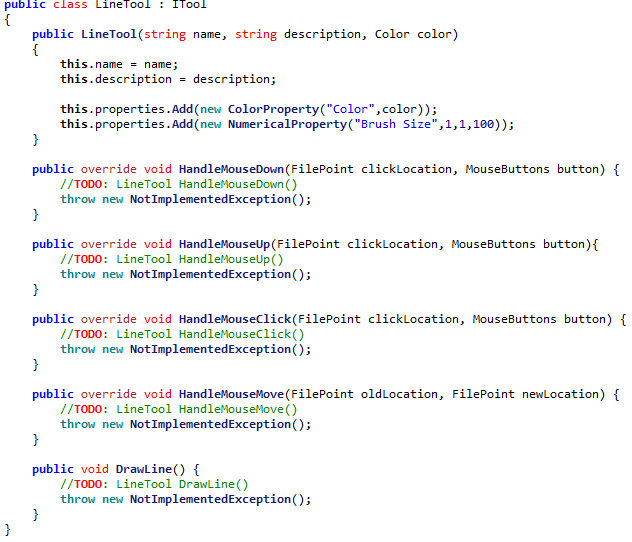
name

value



## LineTool

The skeleton of LineTool has been implemented in accordance to 3.1.3.7:



## 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:

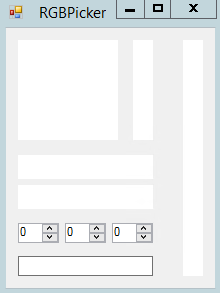
Recent

255

0

0

#FF0000



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

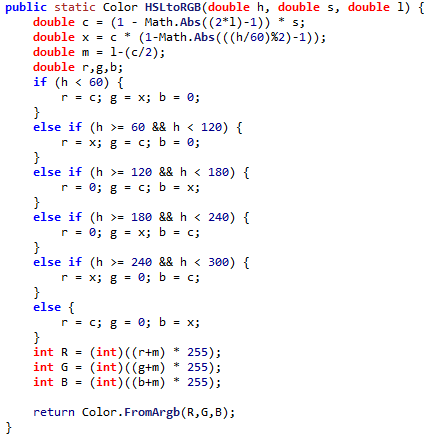
END IF

R = (r + m) \* 255

G = (g + m) \* 255

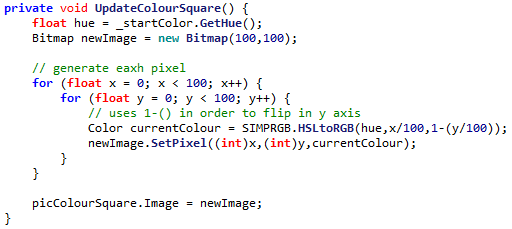
B = (b + m) \* 255

RETURN new Colour(R,G,B)  
}

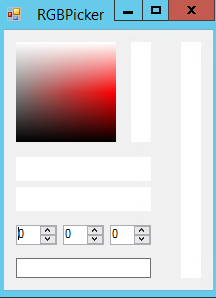
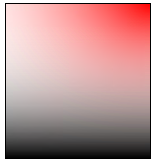


## Generating Colour Square

The code to generate the Colour Square has been implemented, in accordance to Algorithm 3.2C:



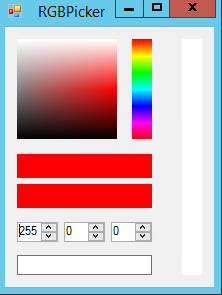
However the output square is different to that expected in the design:

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:

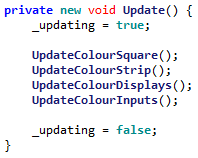


# 10/11/2019 RGB Picker functionality

## 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:

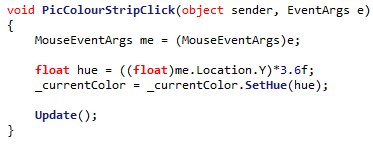




This prevents the current colour from being changed during update.

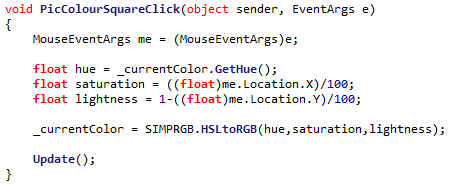
## Colour Strip Input

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:



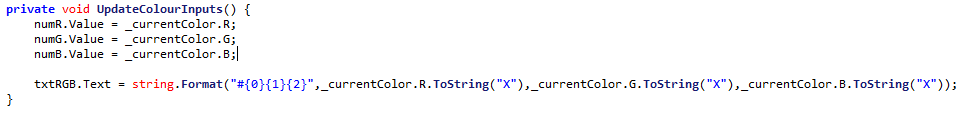
## Colour Square Input

The input from the Color Square can also be easily extracted, with some arithmetic on the X and Y of the clicklocation:



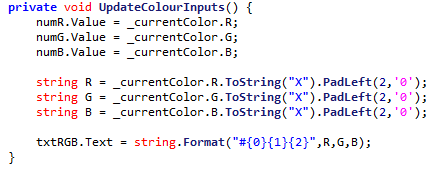
## RGB Display

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:





This can be fixed by adding some extra padding to each part of the hex code:



The parts are now padded

## Problem with returning

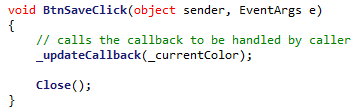
When originally designing the RGBPicker feature, it was assumed that since a colour was an object, it would be passed by reference. Thus any changes to the colour would reflect in its original function. However this is not true for C#’s colour class, which is passed by value. This means that changing the reference won’t work.

## Solution - Delegates

Instead, a **delegate** can be used, which defined by the caller and will handle changing the colour when necessary:



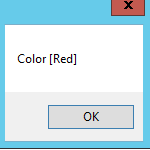
Thus, when a new Save button is clicked, this delegate is called, sending the new colour back to the caller and allowing it to do whatever is needed with the new colour:



Meaning a call can be made like this:



The colour is indeed correctly returned after pressing ‘Save’:



# 13/11/2019 Defining Tools

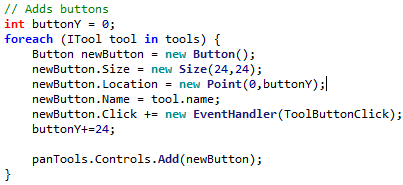
## Storing Tools

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:

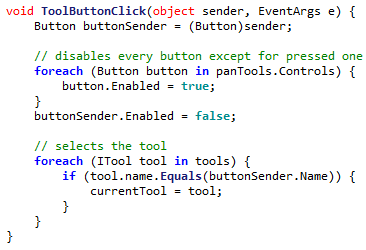


## Displaying Tools

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:



Then when the button is pressed:

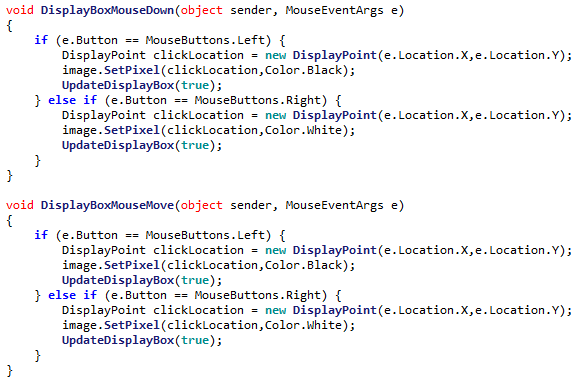


This means that currentTool will store the current tool being used.

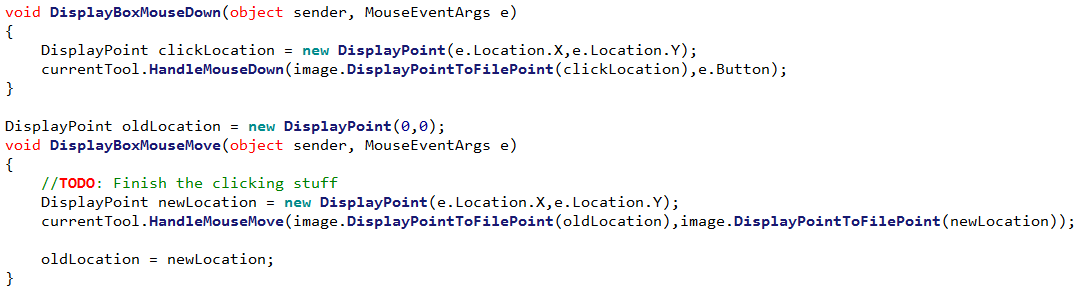
## Hooking Tools

This means that code can be taken out of the current click events, and moved to be encapsulated by the tool:

Before:



After:

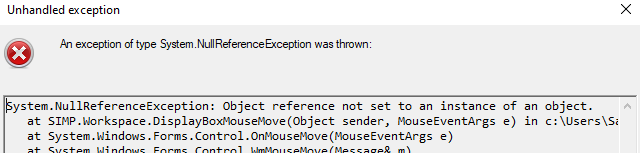


Which uses a new constructor for FilePoint

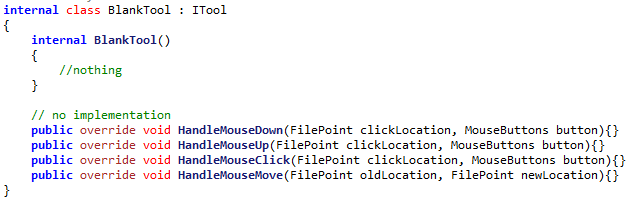


## Fixing a NullReferenceError

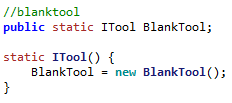
However, a problem emerges with this implementation. At the start of the program currentTool is unset and thus null, so when any reference to this before a tool is selected causes a NullRefernceError to occur.



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:



In ITool:



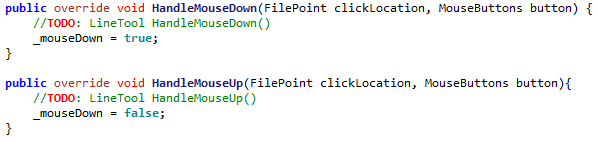
And so the Blank Tool can be set like so:

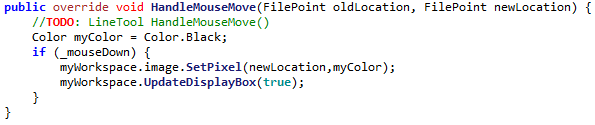


Meaning that the crash is avoided.

# 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:

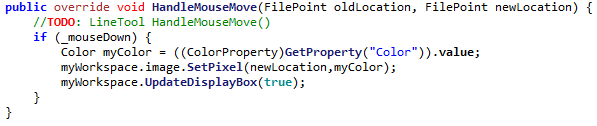




Which is able to draw. The tool will be iteratively improved on until the tool is complete.

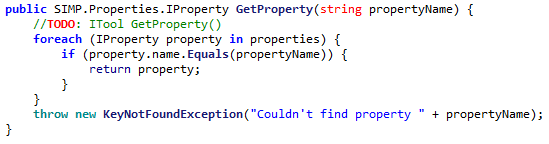
## Implementing Colour Parameter

The first change that can be made is implementing the Colour property into the tool, which uses the GetProperty() method:



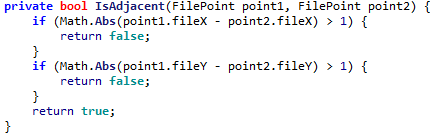
Casts an IProperty to a ColorProperty to get its value

However GetProperty is not implemented, so must be implemented like so:



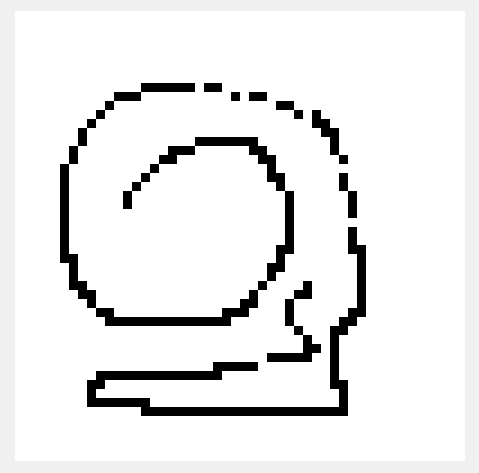
## Connecting Pixels – Bresenhem’s

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:



Then, Bresenhem’s algorithm can be implemented in accordance to Algorithm 3.6A

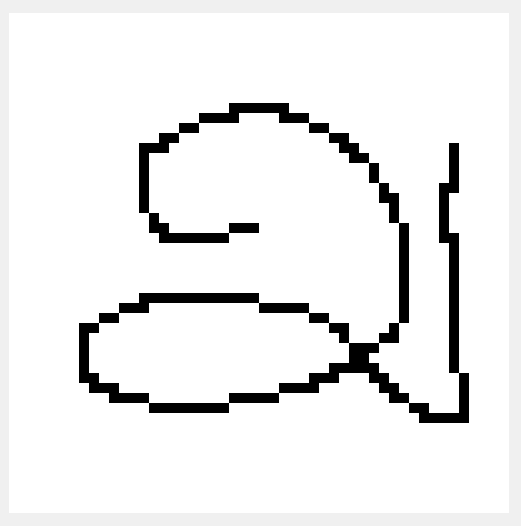
However, it was discovered that there was a problem with the implemented algorithm, it sometimes left a gap:



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:



This makes sure all lines are now connected:

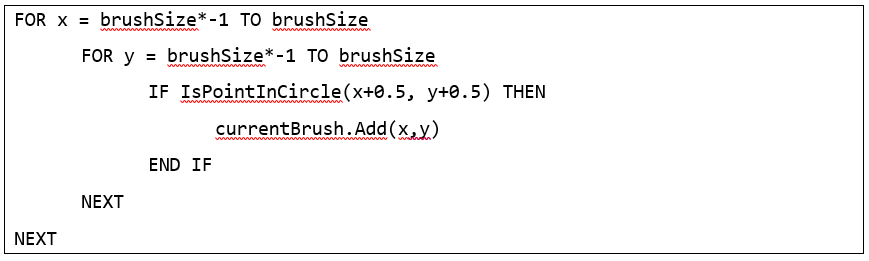


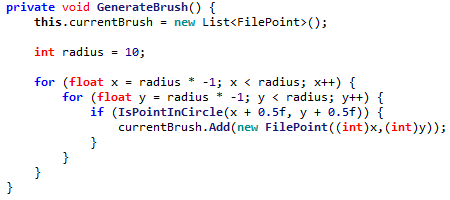
# 17/11/2019 Generating Brushes

In order to store the necessary points for the brush, Algorithm 3.4A has been implemented, making a list for this purpose:

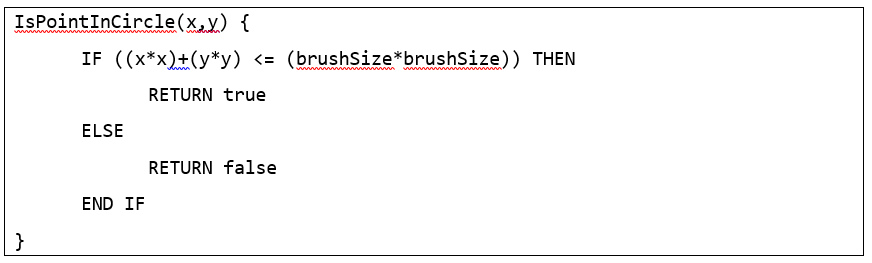


Then, to generate the brush’s points, Algorithm 3.4B has been implemented, which checks all candidate points:





Then the relevant test has been implemented, again according to Algorithm 3.4B:





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:



Brush

Layer 1

Layer 2

Layer 3

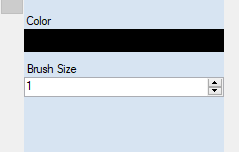
Size

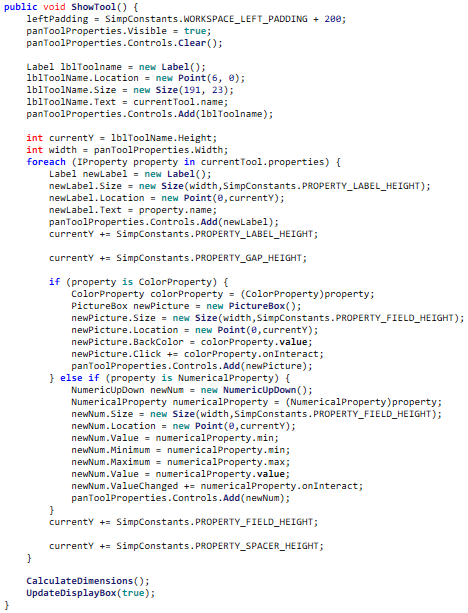
5

Colour



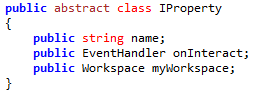
Then some code was implemented to generate the needed controls from a list of properties:



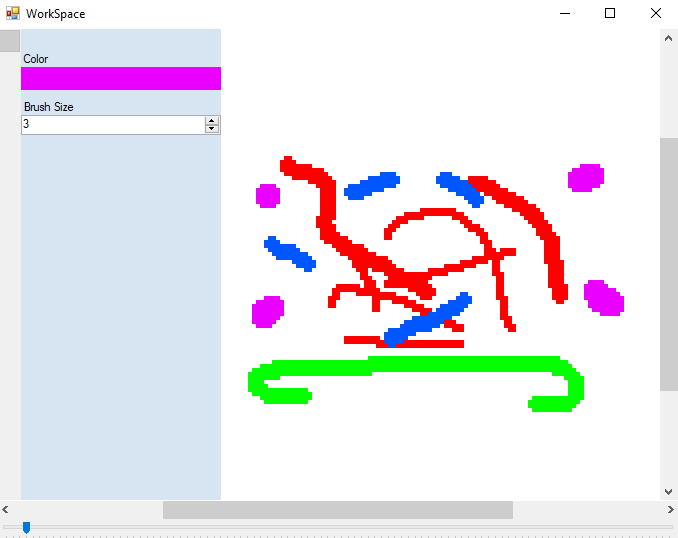


onInteract is implemented by each property.

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:



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

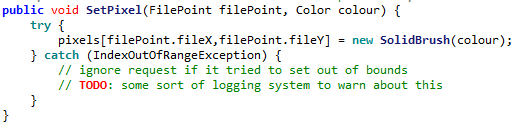
Some elements brought up by the stakeholder feedback are already planned or designed, such as copy & paste, recent colours, rubber, undo / redo & layers. The following can be fixed reasonably easily and so will be acted upon at this time.

|  |  |  |
| --- | --- | --- |
| Issue | ID | Requested by |
| Using a brush of size 100 causes large slowdown | 1 | Alex H |
| Clicking the top of the hue selector box resets colour | 2 | Alex H |
| Starting program with brush selected | 3 | Alex H |
| Cannot start stroke off canvas | 4 | Alex H, Alex G |
| No recent colours | 5 | Alex H |
| No single pixel brush | 6 | Alex H, Dheshpreet |
| Holding button on colour picker should update in real-time | 7 | Alex G |
| Colour picker should be arranged with lightest hue on top | 8 | Alex G |
| Moving cursor off colour picker does not update to max | 9 | Alex G |
| Crash when making window with open properties too small | 10 | Alex G |
| Performance issues | 11 | Dheshpreet |

These 14 issues will be addressed before further progress is made.

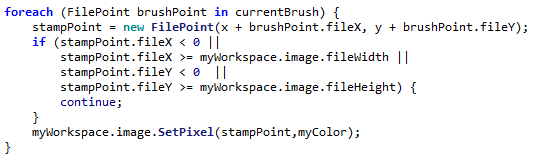
## Performance issues with large brushes

The performance issue can be tracked to the following excerpt of code:



Under normal condition, only the valid section is used. However when big brushes are used the majority of the points are invalid, meaning many error 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:



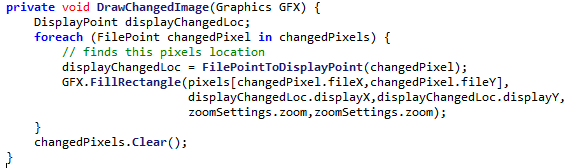
## Performance issues displaying image

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:



Then, when a brush makes a change to the image, the following code is called which only updates the changed pixels:

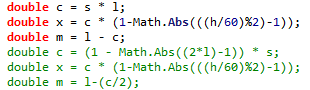


This significantly improves performance on larger images.

## Using HSV rather than HSL

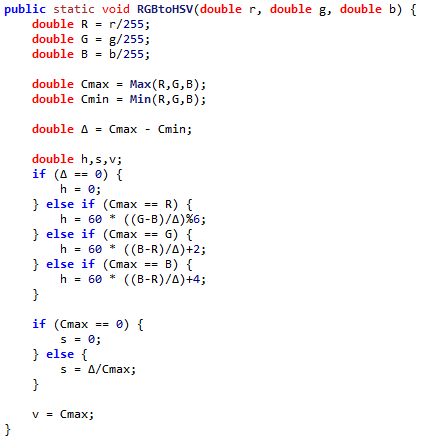
The reason why the colour square didn’t look right to the stakeholders was because it used the HSL colour notation, where they are used to seeing HSV.

Changing the conversion from HSLtoRGB to HSVtoRGB is simple, and only requires a few line changes,

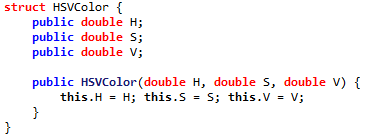


However, it means that an algorithm for converting from RGB to HSL will be needed, as C# does not provide this.

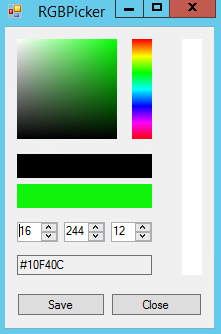
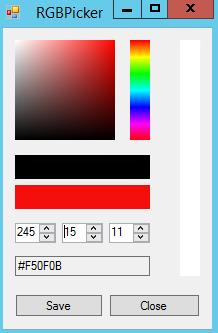
This means an algorithm for the conversion will need to be implemented:



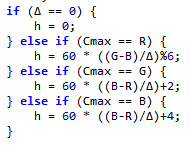
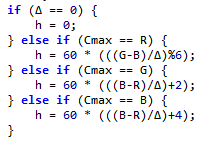
Currently however, there is nothing to return, as there is no class for a HSV colour. This means one must be created:



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:

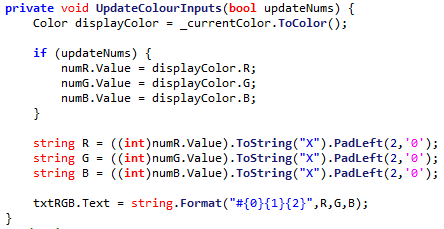
🡪 

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:



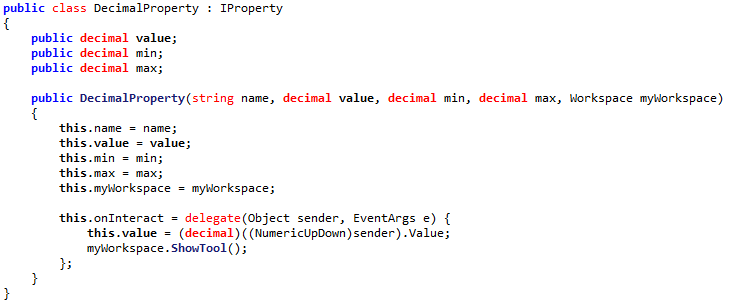
# 20/11/2019 Further improvements

## 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).

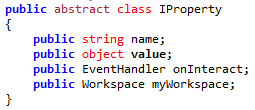
This then introduces a problem – there is no property that accepts decimal input. Thus a new class must be made to have decimal input: 

Then this can be added to the Single Pixel Brush:



However, when adding the brush, an error is introduced as the brush now contains a DecimalProperty rather than a NumericalProperty, so this case fails:



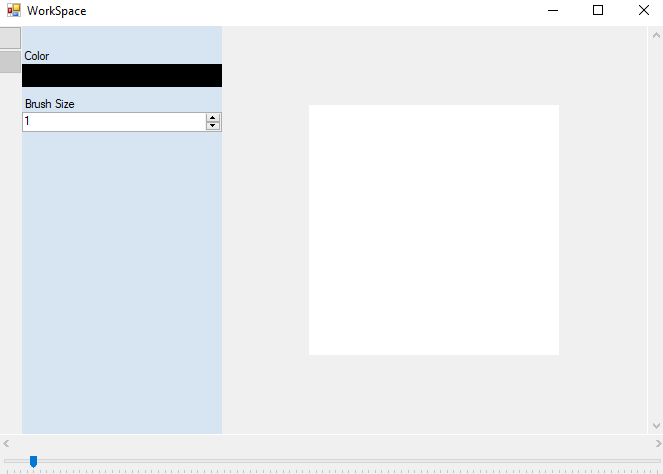
The solution to this is to enforce that all properties stores the value as a public object: 

Thus this object can be converted to a float when needed by the radius:



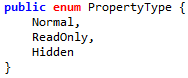
## Property properties

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:

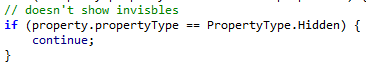


Should not be visible as can’t change size of single pixel brush

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)



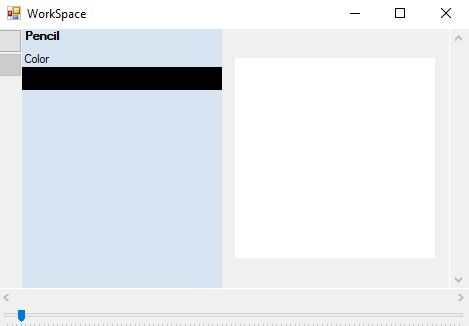
Then, with a check on the display code, hidden properties will not be shown:





## 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:



So the program now starts with the brush selected.

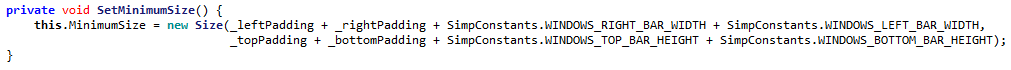
## Fixing crash when resizing window

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:



To fix this, a minimum size can be enforced constantly whenever the values are changed:

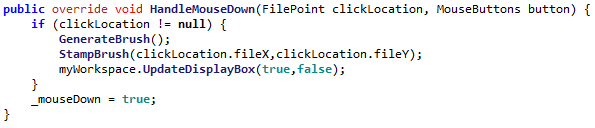




# 21/11/2019 Further improvements

## 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:



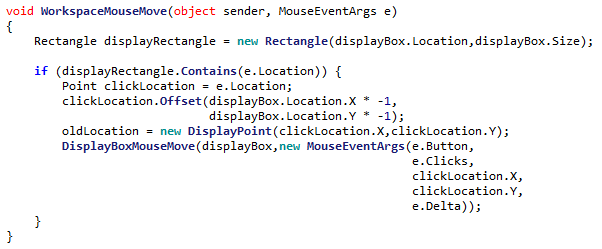
Then code can be called to override the normal movement when a movement is made on the workspace form:



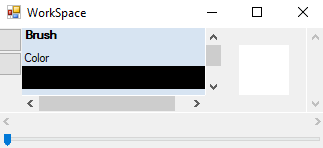
However, after implementing this, a problem arose where when leaving the canvas and coming back on, a line would be drawn from the last location on the canvas to location or re-entry. For example:



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:



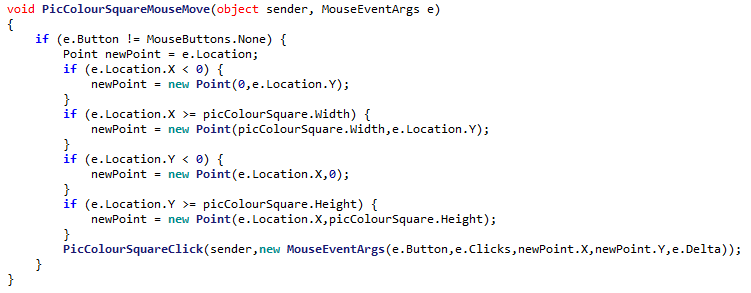
However this still has the same error, as this is now exactly the smallest image size. Adding a further constant now means that the window cannot be made too small. The smallest potential size is now:



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:



However here, the potentially invalid X coord is used

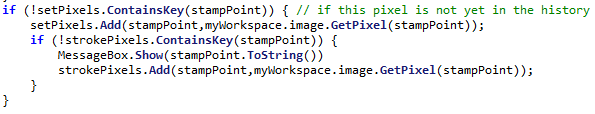
The X coord is fixed here…

To fix this, the updated X is used in the fixing of the Y coord.

# 22/11/2019 Undo

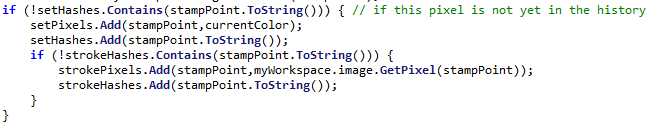
## Implementing image functions

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:



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.

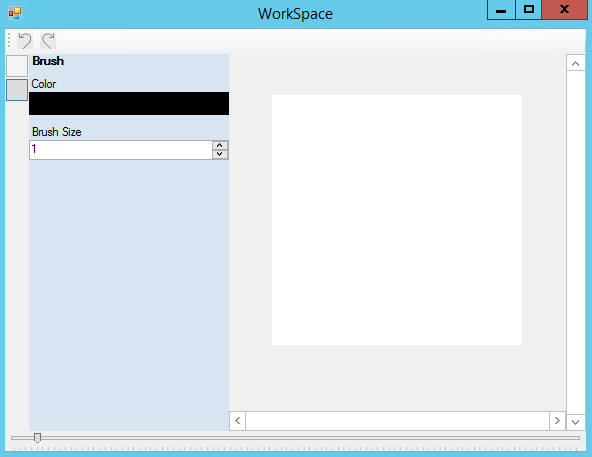
To fix this, a list of primitive strings can be used instead to check whether an item is in the list:



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:



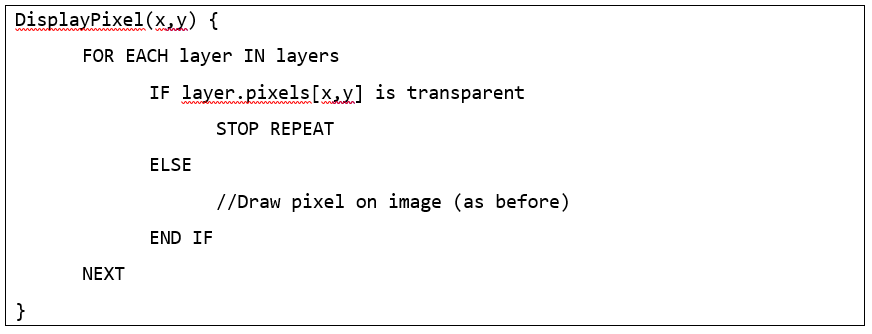
# 23/11/2019 Undo Unit Test

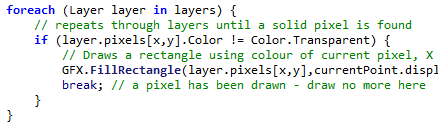
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Undo an action (on default image) | 1 | Cannot be done |  | The undo button is correctly disabled on startup |
| Redo an action (on default image) | 2 | Cannot be done |  | The redo button is correctly disabled on startup |
| Perform an action | 3 | Action is performed onto image |  | The action can still be performed and the undo button becomes enabled |
| Undo the action | 4 | Action is undone |  | The action is reverted and the redo button is now enabled. |
| Redo the action | 5 | Action is redone |  | The action is redone and the undo button is now enabled |
| Undo the action | 6 | Action is undone |  | The action is undone again |
| Redo the action | 7 | Action is redone |  | The action is redone again |
| Perform another action | 8 | Action is done |  | A new action is performed on top of the old one |
| Undo both actions | 9 | Both actions redone |  | Both actions can be undone by pressing undo twice |
| Undo an action | 10 | Cannot be done |  | Undo is currently disabled after undoing the first two actions so cannot be pressed |
| Redo both actions | 11 | Both actions redone |  | Both actions are put back onto the image |
| Redo an action | 12 | Cannot be done |  | The redo button is disabled after redoing twice |
| Undo an action, and preform a new action | 13 | New action is performed on top of old action |  | The action is preformed successfully |
| Redo an action | 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:

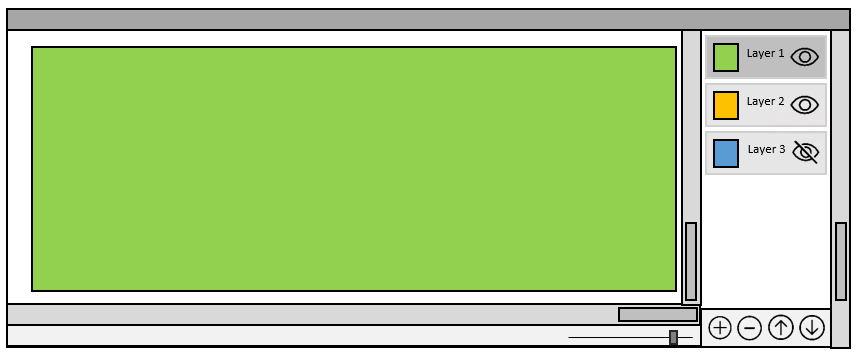


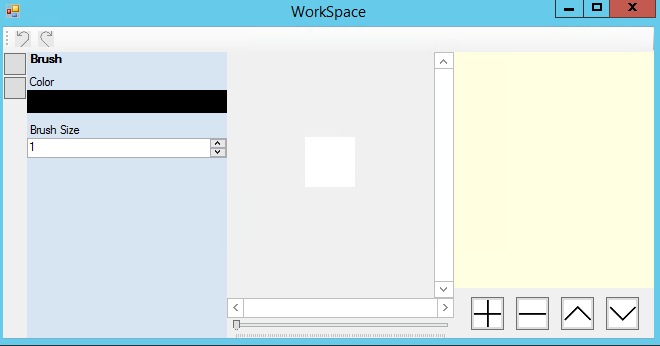


It has been changed to stop repeating when a pixel is filled

## Layer Selector Design

A basic design for the layer selector has been added, according to Algorithm 3.12’s updated design:



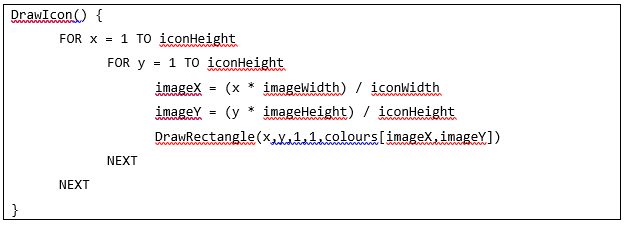


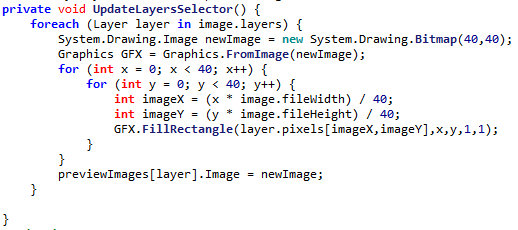
Though the actual layer displaying will be implemented shortly.

# 27/11/2019 Implementing Layer Buttons

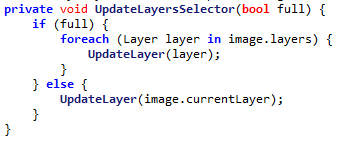
## Layer Preview Rendering Optimisation

An implementation for drawing a layer preview has been implemented, in accordance to Algorithm 3.11A:





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.

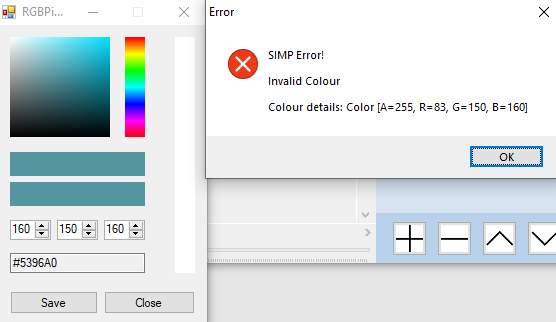


Then, during runtime the false Boolean is passed and only the currentlayer is updated. This is much better for performance.

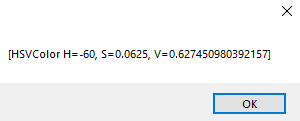
## Colour Interpretation Error

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: 

The Hue is -60, which is clearly invalid.

The introduction of the negative was isolated to these lines of code:



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°



This means the problem colour can now successfully be manipulated:



# 28/11/2019 Layer Unit Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | 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 | System.ArgumentOutOfRangeException | 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:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 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:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | 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:



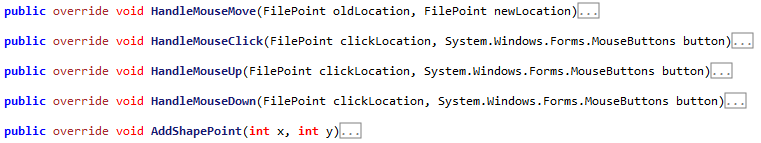
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Move the lowest layer down | 6 | Should be impossible as it cannot go down |  | The program now disables the button when needed |

# 29/11/2019 Shape framework

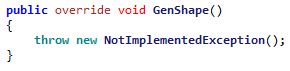
The necessary classes have now been added to implement the various extra tools:

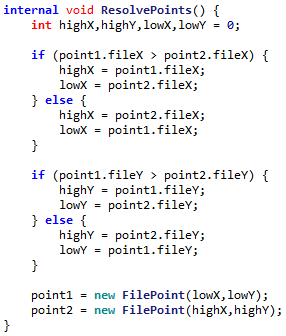


Thus, following the plan laid out in Algorithm 3.15, each class is given its duties. This means that the IShapeTool class handles everything generic to any shape:

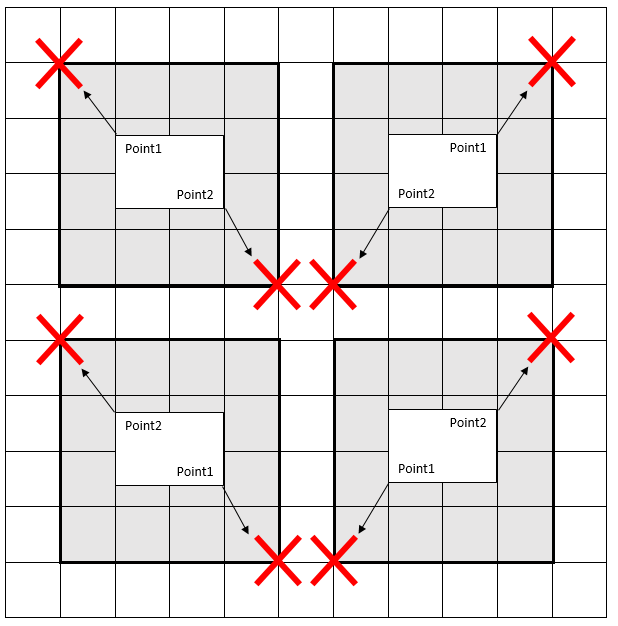


So that the specific shape (in this case LineTool) only needs to provide code for drawing itself, everything else has already been implemented:



Then, the algorithm for resolving points was implemented in accordance to 3.15A: 

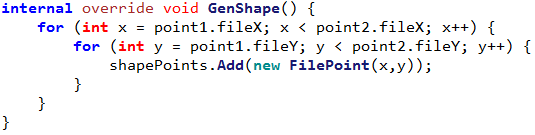
However, this leads to a problem. The ResolvePoint algorithm is meant to collapse these four scenarios into a single scenario:



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:



Then, after this, adding code for each specific tool is simply. It need only override one method. For example the rectangle tool:



Then, when adding the diamond tool, a formula for generating diamonds was needed. Thankfully this is very similar to the circle formula:

To make this into the diamond formula, the square is switched for an Absolute operation:

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 |
| Create line with points (10,5) & (5,10) | 3 | A line is drawn from (10,5) to (5,10) |  | The line does not include the two starting points |
| Create line with points (10,10) & (5,5) | 4 | A line is drawn from (10,10) to (5,5) |  | The line does not include the two starting points |
| Create line with points (5,5) & (10,8) | 5 | A shorter line is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (8,10) | 6 | A thinner line is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (10,5) | 7 | A single row is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (5,10) | 8 | A single column is drawn |  | The line does not include the two starting points |
| Create rectangle with points (5,5) & (10,10) | 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 |
| Create rectangle with points (5,10) & (10,5) | 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 |
| Create rectangle with points (10,5) & (5,10) | 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 |
| Create rectangle with points (10,10) & (5,5) | 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 |
| Create rectangle with points (5,5) & (10,8) | 13 | A shorter rectangle is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (8,10) | 14 | A thinner rectangle is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (10,5) | 15 | A single row is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (5,10) | 16 | A single column is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create circle with points (5,5) & (10,10) | 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 |
| Create circle with points (5,10) & (10,5) | 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 |
| Create circle with points (10,5) & (5,10) | 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 |
| Create circle with points (10,10) & (5,5) | 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 |
| Create circle with points (5,5) & (10,8) | 21 | A shorter circle is drawn |  | The circle has not included the coords of the lower point in its calculations |
| Create circle with points (5,5) & (8,10) | 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:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |
| Create line with points (10,5) & (5,10) | 3 | A line is drawn from (10,5) to (5,10) |  | The line does not include the two starting points |
| Create line with points (10,10) & (5,5) | 4 | A line is drawn from (10,10) to (5,5) |  | The line does not include the two starting points |
| Create line with points (5,5) & (10,8) | 5 | A shorter line is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (8,10) | 6 | A thinner line is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (10,5) | 7 | A single row is drawn |  | The line does not include the two starting points |
| Create line with points (5,5) & (5,10) | 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 >

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Create rectangle with points (5,5) & (10,10) | 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 |
| Create rectangle with points (5,10) & (10,5) | 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 |
| Create rectangle with points (10,5) & (5,10) | 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 |
| Create rectangle with points (10,10) & (5,5) | 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 |
| Create rectangle with points (5,5) & (10,8) | 13 | A shorter rectangle is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (8,10) | 14 | A thinner rectangle is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (10,5) | 15 | A single row is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create rectangle with points (5,5) & (5,10) | 16 | A single column is drawn |  | The rectangle has not included the coords of the lower point in its calculations |
| Create circle with points (5,5) & (10,10) | 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 |
| Create circle with points (5,10) & (10,5) | 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 |
| Create circle with points (10,5) & (5,10) | 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 |
| Create circle with points (10,10) & (5,5) | 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 |
| Create circle with points (5,5) & (10,8) | 21 | A shorter circle is drawn |  | The circle has not included the coords of the lower point in its calculations |
| Create circle with points (5,5) & (8,10) | 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 |

# 3.2.1 Stakeholder Feedback Drawings

My stakeholders each drew images using SIMP, with the latest tools.

## Dheshpreet



# 3.2.2 Success Criteria Evalution

|  |  |  |
| --- | --- | --- |
| **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 | | |
| 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 |
| 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

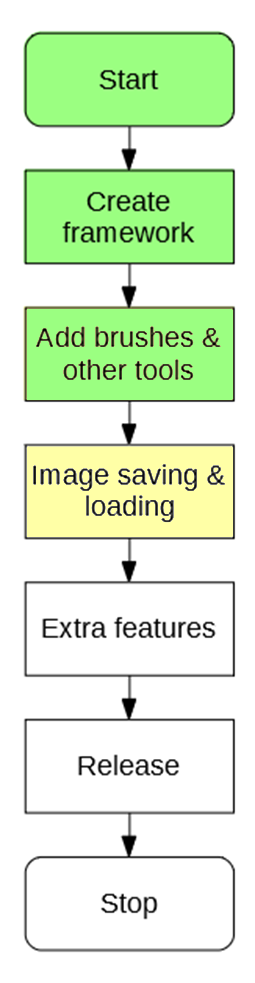
Completed

|  |  |  |
| --- | --- | --- |
| **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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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  
}

Writes R, G, and B as three separate values

This makes saving to the file a much simpler task

# 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](#_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;  
 }  
 try {  
 layer.SetPixel(new Color(R,G,B))  
 } catch InvalidParameterException {  
 layer.SetPixel(new Color(0,0,0))  
 }  
 NEXT y  
 NEXT x  
 NEXT i  
}

As soon as the end of the file is reached, stop reading

# 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

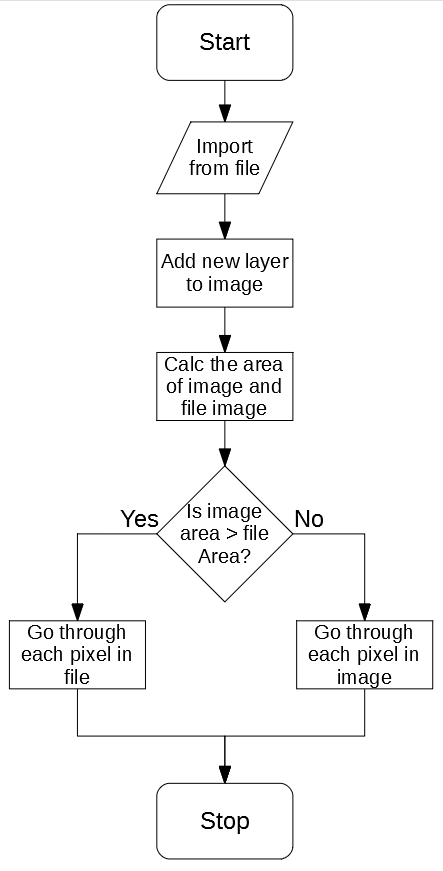
Importing as much the same as exporting, as C# also contains inbuilt classes for this purpose. However once the image has been encapsulated into a bitmap, it must be placed onto the image. This can be done by creating a new layer, and transposing each pixel of the file onto the image, or check each pixel 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:

New Image

Width

Height

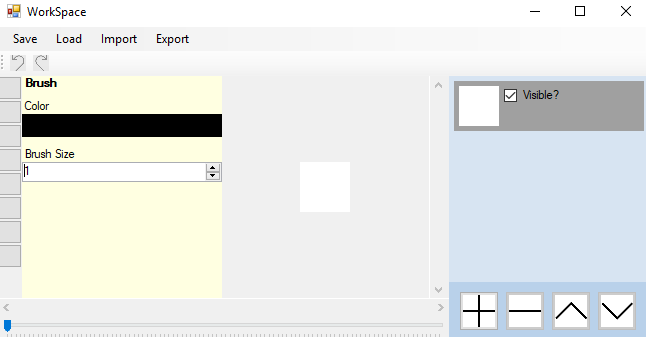
Create

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

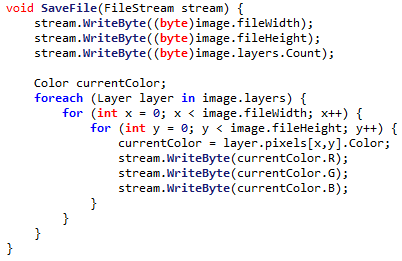
## Writing algorithm

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  
}



## Header Error

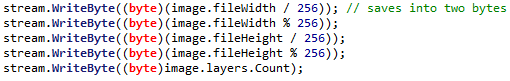
After programming the header, an error was detected. It occurred when one of the dimensions of the image was greater than 255. As this could not be saved into one byte, an error was thrown.



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 255² (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.

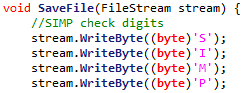
This makes the new code for saving:

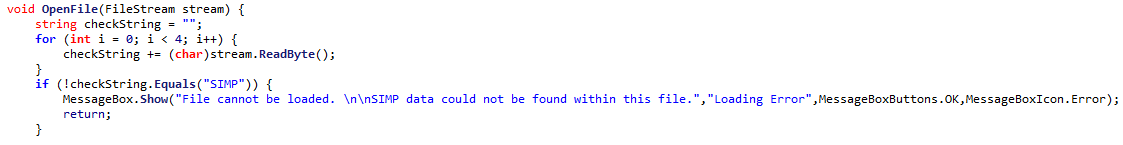


# 9/12/2019 More robust importing

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’.





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:

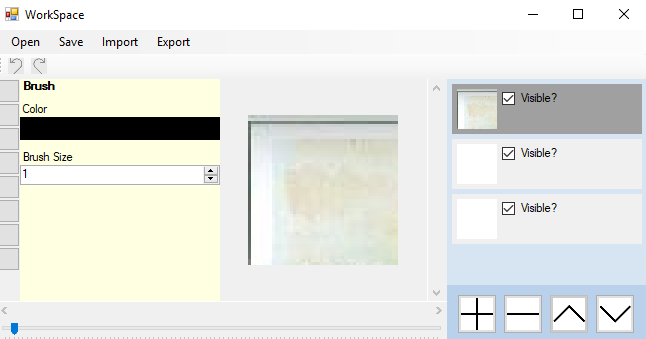


To fix this, the zoom of the image is temporarily set to 1, then exported. This means the proper image is always exported:

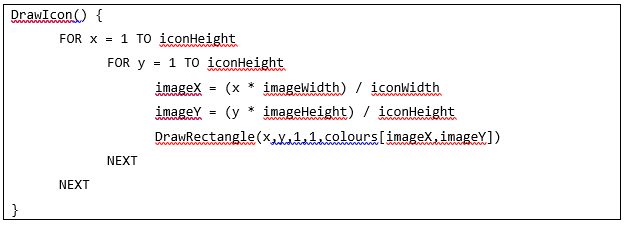


## Fixing Importing

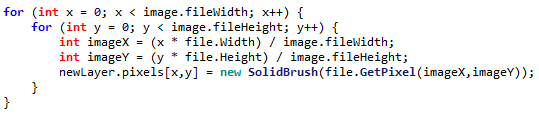
The importing however turned out to be quite flawed, for most images the result would be:



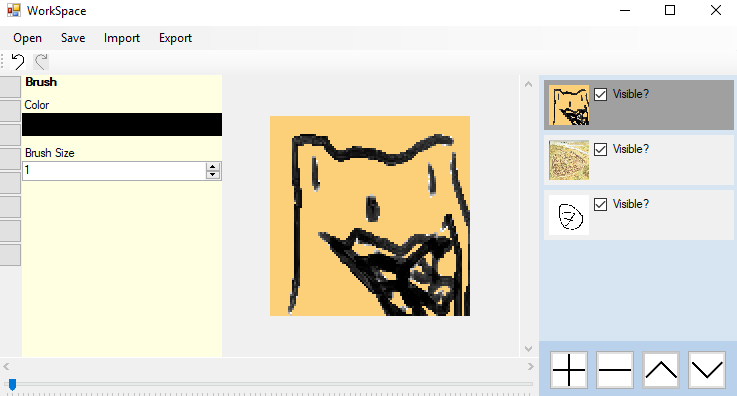
So, a different importing system needs to be used. A similar system to the one used for drawing the layer preview, where a foreach is used on each pixel of the image.



Changing the code for importing to:

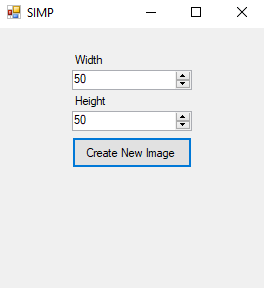


This means that images can now be properly imported:

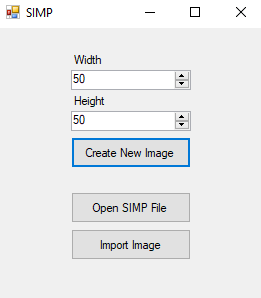


## Improving Start page

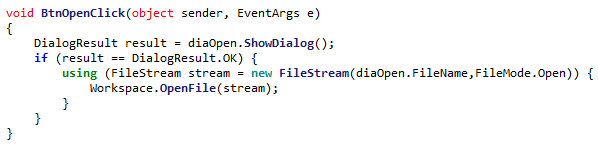
Currently the start page for SIMP is very basic, only allowing to create an image, and without any options to import or open an existing SIMP file:



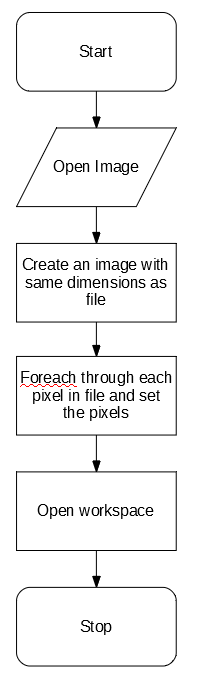
So, a few more buttons have been requested by the stakeholders:



Opening an existing SIMP file is reasonably simple, it can use the existing open file function designed before.



However importing an image will need to be designed slightly differently

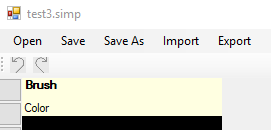


After doing this, the start page is fully functional.

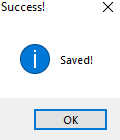
# 14/12/2019 Improvements to saving

## 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:



# 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 | 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 |

Not completed

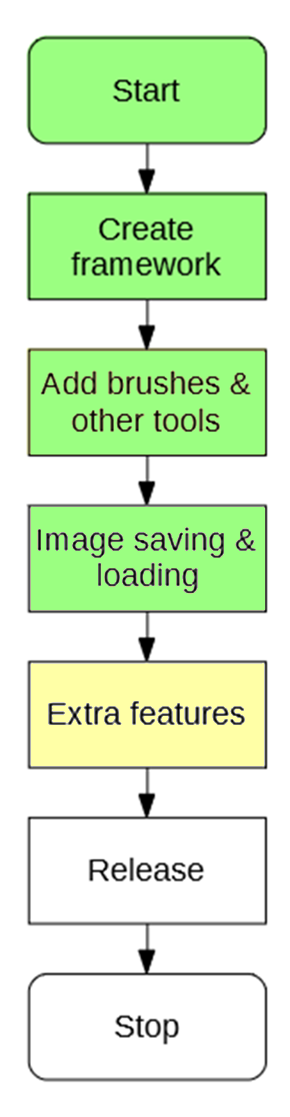
Completed

|  |  |  |
| --- | --- | --- |
| **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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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 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:

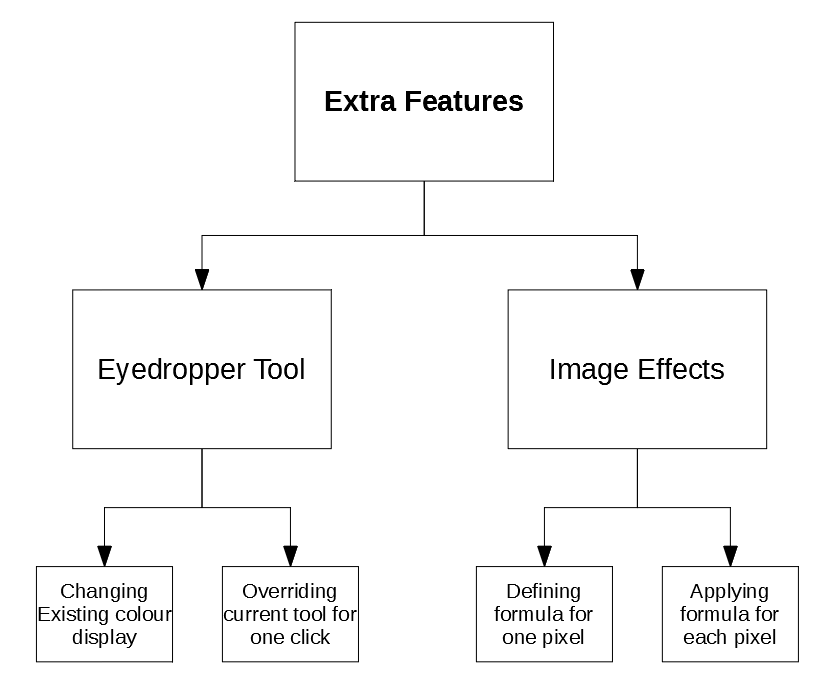
Not completed

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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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

<class>

EyedropperTool

name\*

description\*

properties\*

editingProperty

returnToTool

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

GetProperty\*

SetColour

### 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

<class>

FXTool

name\*

description\*

properties\*

pixelTransformation

HandleMouseDown\*

HandleMouseUp\*

HandleMouseClick\*

HandleMouseMove\*

GetProperty\*

ChangeImage

### Properties

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| name | String | Inherited from ITool |
| description | String | Inherited from ITool |
| properties | List of ToolProperties | Inherited from ITool |
| pixelTransformation | [PixelTransformation](#_5.1.3.3_PixelTransformation) | 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.

<class>

PixelTransformation

transform

### 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 alogirthm 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

<class>

NumericalProperty

name\*

value

options

### 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:

Colour

This should be changed to look like:



Colour

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

## Algorithm 5.1 & Algorithm 5.2 Unit Test

|  |  |  |  |
| --- | --- | --- | --- |
| Test | ID | Expected Result | Comment |
| Opening a tool with a color property | 1 | Color has option to use the eyedropper tool on it. | This makes sure that the eyedropper tool button is correctly visible. |
| Pressing the eyedropper button | 2 | Button indicates that it is activated and eyedropper is active | This makes sure that the user is aware the tool is active |
| Selecting a point on the image | 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 |
| Selecting the eyedropper tool again | 4 | The same process can be done, same as first time eyedropper is pressed | This makes sure the eyedropper tool is reset correctly |
| Selecting a point not on the image | 5 | The tool is still active and no colour is changed | Tests that the program handles clicking at a point not on the image |
| Pressing the eyedropper button again whilst it is selected | 6 | The tool deactivates and normal execution resumes | Tests that the eyedropper can be deselected if user wants to |
| Selecting another tool with eyedropper active | 7 | Eyedropper deactivates and new tool is selected | Tests that the program handles deselecting eyedropper when a different tool is selected |
| Selecting a transparent pixel | 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

|  |  |  |  |
| --- | --- | --- | --- |
| Test | ID | Expected Result | Comment |
| Apply grayscale transformation | 1 | The image becomes shades of grey | Tests that the grayscale transformation functions correctly |
| Apply black & white transformation | 2 | The image becomes black and white | Tests that black and white transformation functions correctly |
| Apply invert transformation | 3 | The colours of the image invert (e.g. green -> purple) | Tests that the invert transformation functions correctly |
| Apply invert transformation twice | 4 | The colours return to normal | Tests that the invert is reversible |

5.2 Development

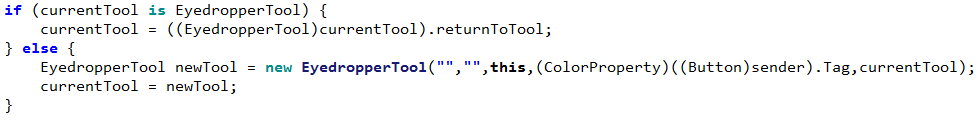
# 19/12/2019 Adding Eyedropper

## Algorithm 5.1 & Algorithm 5.2 Unit Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Opening a tool with a color property | 1 | Color has option to use the eyedropper tool on it. |  | This makes sure that the eyedropper tool button is correctly visible. |
| Pressing the eyedropper button | 2 | Button indicates that it is activated and eyedropper is active |  | This makes sure that the user is aware the tool is active |
| Selecting a point on the image | 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 |
| Selecting the eyedropper tool again | 4 | The same process can be done, same as first time eyedropper is pressed |  | This makes sure the eyedropper tool is reset correctly |
| Selecting a point not on the image | 5 | The tool is still active and no colour is changed |  | Tests that the program handles clicking at a point not on the image |
| Pressing the eyedropper button again whilst it is selected | 6 | The tool deactivates and normal execution resumes |  | The eyedropper tool is selected again, causing a crash when trying to return to the previous tool |
| Selecting another tool with eyedropper active | 7 | Eyedropper deactivates and new tool is selected |  | Tests that the program handles deselecting eyedropper when a different tool is selected |
| Selecting a transparent pixel | 8 | 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:



## Fixing Error #8

This can be fixed with a single line, replacing the colour with white if it is transparent

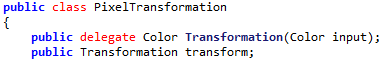


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Pressing the eyedropper button again whilst it is selected | 6 | The tool deactivates and normal execution resumes |  | The eyedropper tool now deselects when pressed again |
| Selecting a transparent pixel | 8 | White is returned instead |  | The new line returns white |

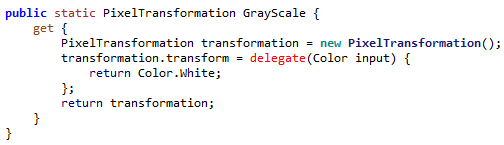
# 20/12/2019 Adding FX

## Implementing Delegate System

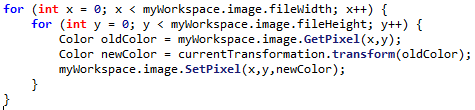
The delegate system is implemented as a part of the PixelTransformation class. It is defined, then PixelTransformation is given it as a property:



Then, when creating the static members, the transform member can be set to the necessary code:



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:



This closely follows the previously laid out design:

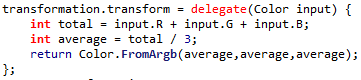
ChangeImage() {  
 FOR x = 0 TO width  
 FOR y = 0 TO height  
 oldColor = image.GetPixel(x,y)  
 newColor = currentTransformation.transform(oldColor)  
 image.SetPixel(x,y,newColor)  
 NEXT  
 NEXT  
}

## 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)  
}

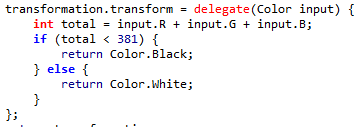
R, G and B all use the same average value



## 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:



## Implementing Invert

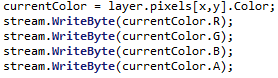
Invert(colour) {  
 newR = 255 – colour.R  
 newG = 255 – colour.G  
 newB = 255 – colour.B  
 return new Colour(newR,newG,newB)  
}

# 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 was saved, meaning that transparency was **not** saved. This should be resolved so that transparency is saved.

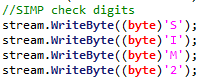
So the saving code has been changed to save the alpha (transparency) channel.



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)



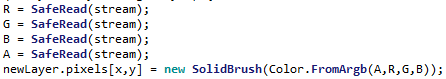
Then, when loading, the image, there is a switch case to decide what sort of file it is:







Then, in the SIM2 loading, the alpha channel is loaded.



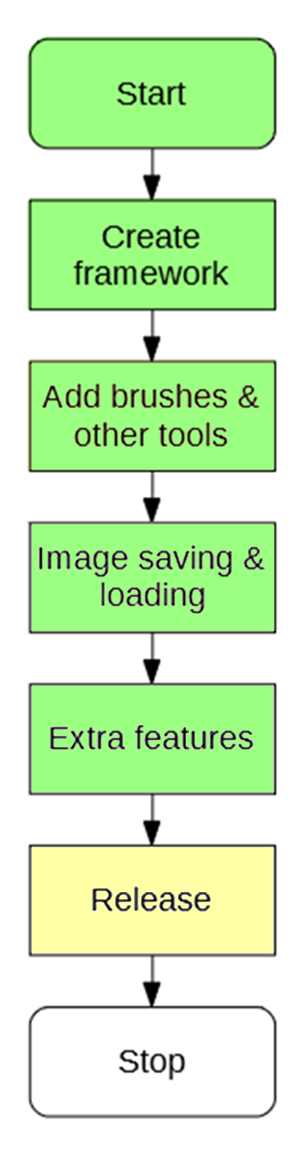
# Success Criteria Evaluation

|  |  |  |
| --- | --- | --- |
| **Feature** | **Proof** | **Code** |
| Section B – Other Editing Tools | | |
| Eyedropper tool | Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it | B11 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | 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    Every feature is either clickable as a button here, accessible after clicking the tool’s button, or clickable after opening the RGB Editor. | D3 |

|  |  |  |
| --- | --- | --- |
| **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 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | 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 Beta Testing

## Functionality Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Can you create an image using SIMP | 1 | A SIMP workspace opens |  | The user can start the program |
| Use the pen tool | 2 | Pen appears on canvas |  | A simple shape can be drawn and cross over itself |
| Change the width of the brush | 3 | The user can draw with a different width of brush |  | The width has been increased to a radius of 3 and so is a thicker stroke than the first one |
| Change the colour of the brush | 4 | The user can draw with a different colour |  | After using the RGB editor, the colour is now red |
| Use the pencil tool | 5 |  |  | The pencil tool can be used for finer details. |
| Undo stroke | 6 |  |  | The last stroke, which was the red one, is undone and is no longer visible |
| Redo stroke | 7 |  |  | The last action to be undone, the red stroke, is now put back on the canvas |
| Use the eraser tool | 8 |  |  | The eraser can remove any pixels it touches in its stroke, clearly making a line |
| Use the straight line tool | 9 |  |  | The straight line tool creates as straight a line as it can between two points. |
| Use the rectangle tool | 10 |  |  | The rectangle tool makes a solid rectangle, which can also have its colour changed |
| Use the circle tool | 11 |  |  | The circle tool can create an almost perfect circle, on top of the existing rectangle |
| Use the diamond tool | 12 |  |  | The diamond tool creates a sideways rectangle |
| Use the fill tool | 13 |  |  | Parts of the image have been filled in purple, these are complex shapes that could not be filled easily with a brush. |
| Use the image FX tool | 14 |  |  | The image as a whole has been changed such that it is now entirely black and white |
| Zoom in | 15 |  |  | The image has been zoomed in so that the centre can be edited more finely. Scroll bars have appeared to move the view |
| Move bar to left | 16 |  |  | The middle left of the image is now being viewed |
| Move bar to right | 17 |  |  | The middle right of the image is being viewed |
| Move bar up | 18 |  |  | The top right of the image can now be viewed |
| Move bar down | 19 |  |  | The bottom right of the image can now be viewed |
| Zoom out | 20 |  |  | The image is now slightly less zoomed in, the scroll bars indicate this as they have less possible values |
| Add new layer | 21 |  |  | The image now has two layers, with the new layer being completely transparent |
| Change selected layer | 22 |  |  | The new layer is now selected, any changes will instead be made to this layer |
| Draw on layer | 23 |  |  | When making a red stroke, the layer indicator shows the red is separate from the lower layer |
| Make layer invisible | 24 |  |  | Then making the red stroke layer invisible, only it vanishes and not the lower layer |
| Make layer visible | 25 |  |  | The layer can then be made visible again |
| Shift layer down | 26 |  |  | The red stroke is now below the previous layer and peeks from behind |
| Shift layer up | 27 |  |  | The red stroke now becomes back on top of the grey layer |
| Remove layer | 28 |  |  | The red stroke is now completely removed from the image |
| Save image as new file | 29 |  |  | A file browser opens and allows the user to enter the name and path of the new file. A popup is then shown on the main window to alert the user that the file saved successfully |
| Open the file | 30 |  |  | A dialog opens for selecting the file to open, and this then opens in a new window. |
| Make changes | 31 |  |  | The opened file is a new copy of the existing file, and so changes can be made to it. |
| Save file | 32 |  |  | This is done by pressing ‘Save’ instead of ‘Save as’. This instead overwrites the file you are currently editing |
| Open edited file | 33 |  |  | This file is then opened, but the program doesn’t recognise that in this case you are editing a known SIMP file. ‘Save’ should be accessible on the new window to allow you to save edits but the user must open a new file |
| Import image as layer | 34 |  |  | An external image (in this case bomb clipart) can be imported as a layer into the SIMP workspace, where it resizes to the size of the image |
| Export image | 35 |  |  | This edited image can be exported to either a PNG, JPG or BMP format, which can be opened in any image viewer. |
| Import image as starter | 36 |  |  | Any BMP, PNG or JPG image can be imported as the starting base for a SIMP image, allowing edits to existing images using the SIMP toolset |

## Durability Testing

This will be testing focused on making sure the program is durable, can withstand any invalid output. Durability will also be tested during [Usability Testing](#_Usability_Testing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Create image with largest width and height possible | 1 | A large image successfully opens |  | Whilst hard to demonstrate, as shown by the small scroll bars a large image is there. However many actions perform slowly, such as the line tool. |
| Create an image with smallest width and height possible | 2 | A 1 pixel by 1 pixel image is opened. It is not very useful but does work |  | There is a tiny image created successfully and the program handles it like it would any other image. |
| Create an image with maximum width and smallest height | 3 | A very thin image is created |  | As shown by the lower scroll bar, there is a lot of image to scroll |
| Create an image with maximum height and smallest width | 4 | A very tall image is created |  | As shown by the side scroll bar, there is a lot of image to scroll |
| Create ten maximum size images | 5 | Ten separate workspaces are created, and can each be edited |  | After attempting to open the fifth workspace, SIMP closes without warning. This could potentially cause data loss if unhandled. |
| Attempt to enter a string as width | 6 | This is blocked, a string cannot be used as a width |  | The numeric box handles this, and prevents the user from typing any text at all |
| Attempt to enter a string as width | 7 | This is blocked, a string cannot be used as a width |  | The numeric box is an already tested unit of the .NET library so does not need thorough testing. **Somebody else has done that.** |
| Attempt to start a stroke off the edge of the canvas | 8 | The stroke starts as soon as the cursor makes it onto the canvas |  | Attempting the stroke illustrated causes no stroke to appear. This was a feature of SIMP at some point but has broken during development |
| Have a stroke that starts on the canvas, leaves the canvas then re-enters the canvas | 9 | The stroke starts on the canvas, then when the cursor re-enters the stroke continues but is not connected |  | Attempting the stroke illustrated does work, if the stroke starts on the canvas. |
| Make a long stroke moving mouse around for 10 seconds |  | A very long stroke is created, but there is no sort of limit for how long the stroke can be |  | There is a very long stroke, but the program doesn’t appear to have any sort of limit on how long a stroke can be |
| Make a long stroke by moving mouse very quickly |  | The program still creates a smooth curve as if the mouse were moved slowly |  | The program experiences a lot of slowdown, visibly pausing while making the stroke, shown by the jagged lines. |
| Increase brush size to max and make a stroke |  | A small, thick stroke is created |  | The program makes a very long pause before creating the stroke, potentially due to how the large brush is inefficient |
| At this size, make a long stroke moving mouse at random |  |  |  | The program pauses for an extended period of time before resulting in a stroke not very close to the path drawn |
| At this size, make a stroke by moving mouse very quickly |  | The program draws the same image as above |  | The program pauses for an even longer period of time before drawing a line that’s not very close either. It does however not crash. |
| Create a filled image and then move the easer tool very fast |  | The path followed by the cursor is erased |  | This is to show that the problems that the pen tools have are also shared by the eraser tool, as it uses the same code. |
| Create a shape with a point starting off the canvas |  | A shape that is in theory bigger than the canvas is drawn, with the point clicked being the in theory location of the point |  | Clicking at any point off the canvas does nothing, so it is not possible to draw a circle that has edges off the canvas. |
| Draw the largest rectangle possible |  | A very large rectangle is drawn |  | The program draws this large shape in good time, taking less than a second |
| Draw the largest circle possible |  | A very large circle is drawn |  | The program also draws this large shape in good time, taking less than a second |
| Draw the largest diamond possible |  | A very large diamond is drawn |  | As the algorithm for these shapes simply involves iterating points, it has roughly **O(n)** complexity and so scales well for large shapes |
| Fill the previous diamond |  | The diamond is now a different colour |  | The program however takes around 5 seconds to complete this action, far too slow |
| Fill in the entire canvas |  | The entire canvas is now the same colour |  | Filling the whole canvas at once took nearly two minutes, and the program was not responding while this was happening. Much too slow |

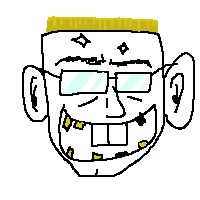
## Usability Testing

To test for usability, I tasked each stakeholder to create an image using SIMP, to see how easy it was for them to use the program. This also allowed me to compare to an image made earlier on in development, to see how SIMP as a program had progressed.

### Alex G

Image created previously in development

Image created using latest SIMP:



Generally SIMP is much easier to use now, however the colour should transfer across the tool, as right now each tool has its own colour. Also it’s not possible to tell right now what a tool will do before you use it, some sort of indicator would be handy.

### Alex H

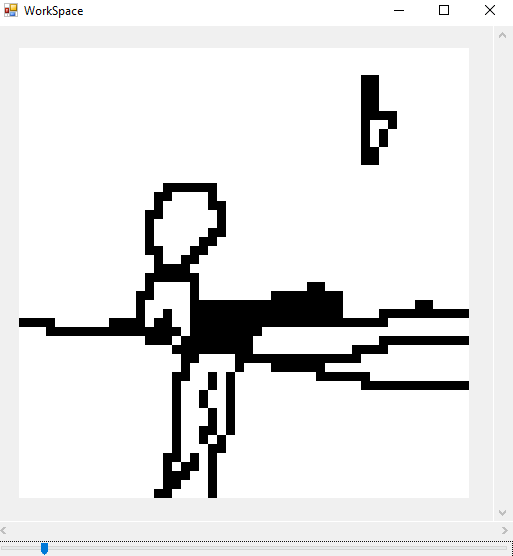
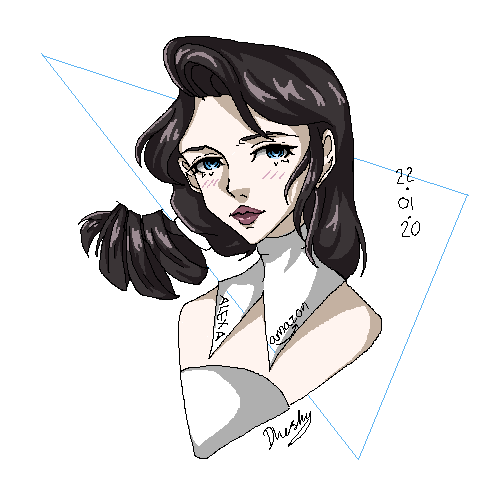
Image created previously in development: 

Image created using latest SIMP: 

SIMP has many more features from when I last made an image with it, especially the option for colour and multiple brush sizes, however I did find an error where you could eyedropper tool an invisible layer, and you can only eyedropper the currently selected layer.

### Dheshpreet

Image created previously in development: 

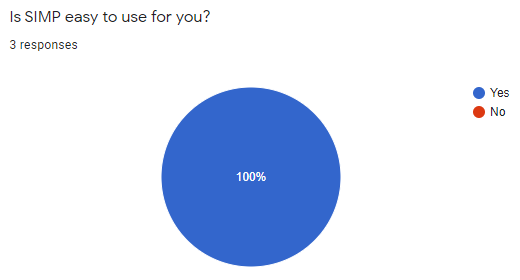
Image created using latest SIMP: 

I did find SIMP quite limited compared to what I usually use, and the fill tool was particularly slow for filling in large areas of the image (meaning I had to colour in by hand a lot) but overall it’s a useful little program.

Or, to conclude for the usability testing there are these problems:

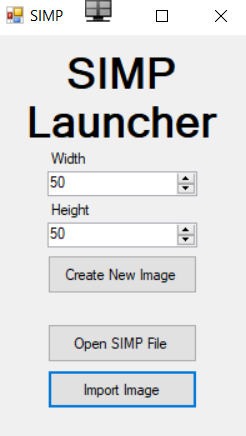
* Colours should transfer across brush to any brush with a colour
* Fill tool is much too slow to be effective
* Eyedropper tool yields the incorrect colour in some scenarios

Otherwise the stakeholders found the program quite usable. In a poll as to whether SIMP was use or not, all 3 stakeholders replied positive:



# 6.2 Criteria Fulfilment

## Starting Window



Allows you import an image of varying types

**Fulfils: C2**

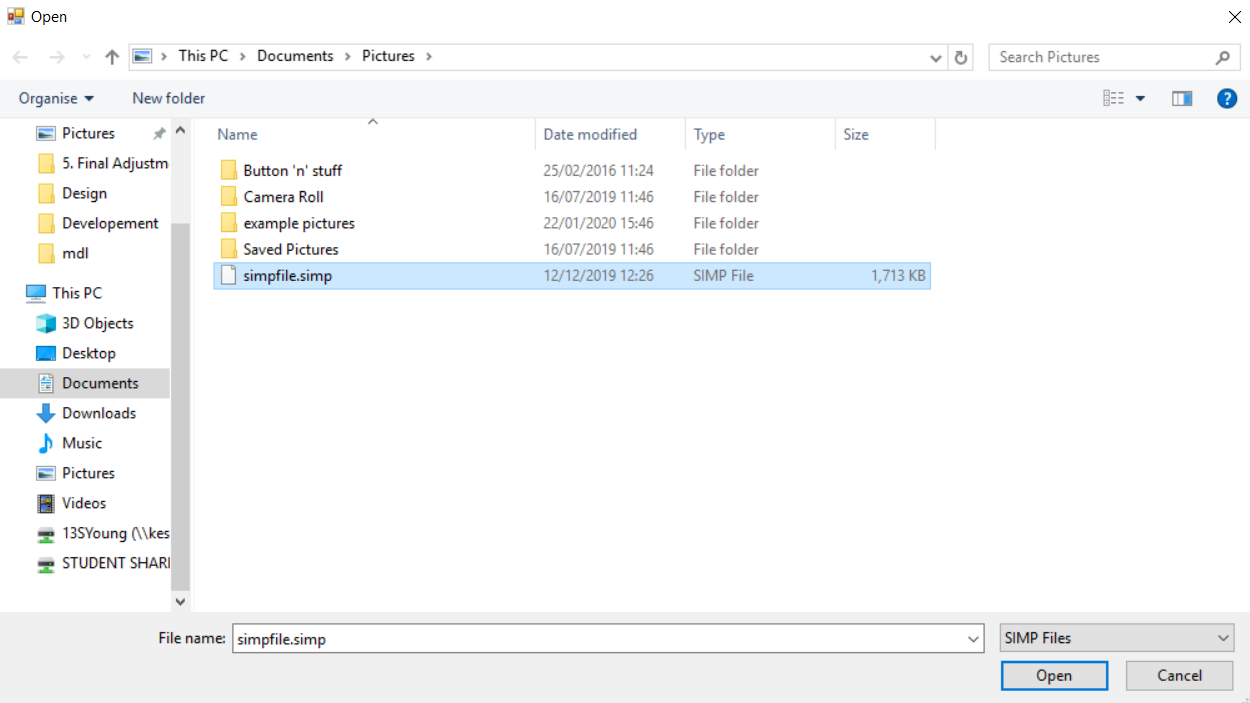
Allows you to open an existing SIMP File

**Fulfils: C5**

Allows you to define what size you want

**Fulfils: C1**

## SIMP Open File Browser



Validation such that only .simp files can be opened

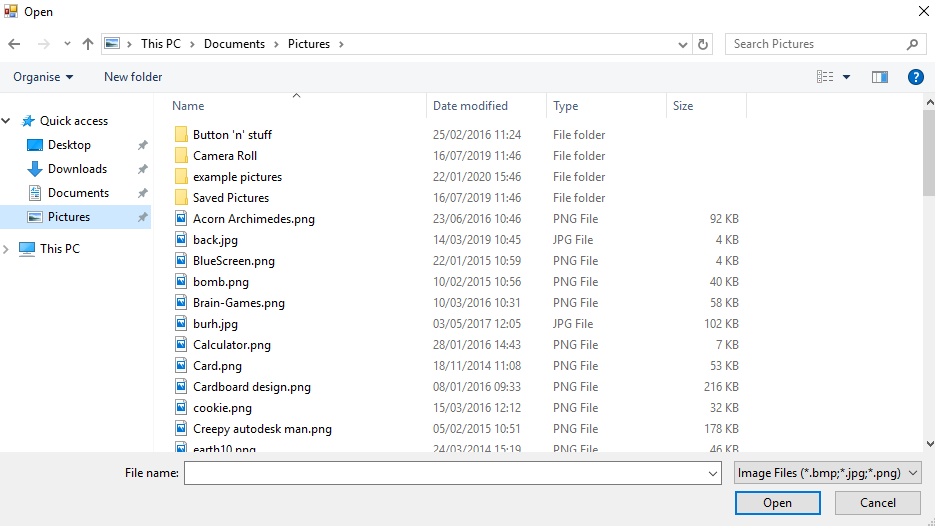
An existing file in the file browser

**Fulfils: C5**

## SIMP Import Image Browser

An image file that can be imported

**Fulfils: C2**



Has options to import both PNG and JPEG

**Fulfils: C5**

Validation such that only certain image files can be opened

## Main Workspace

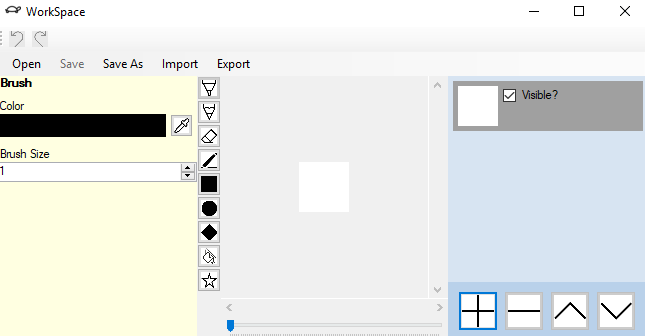
Eyedropper tool

**Fulfils: B11**

Image viewer

**Fulfils: B1**

Colour of the pen can be changed



Layer navigator

**Fulfils: C2**

Brush size can be varied

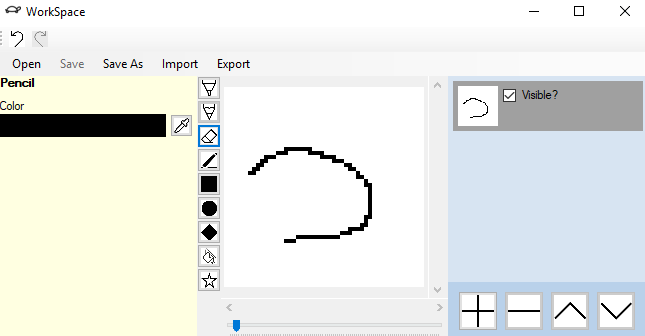
**Fulfils: A1**

Open existing proprietary file format

**Fulfils: C1**

Image is more zoomed in than previously

**Fulfils: B9**



Edge of the line has a hard edge, no transparent

**Fulfils: A2**

Zoom bar to allow zoom to be changed

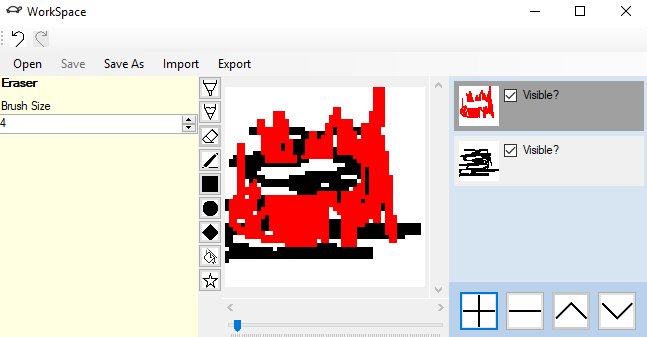
Add, remove, shift up and shift down layer buttons.

Single pixel pencil tool for fine work

**Fulfils: A5**

Layer tool showing two layers, with the top one currently selected

Overwrite currently saved file



Eraser has made pixels transparent, so you can see layer below

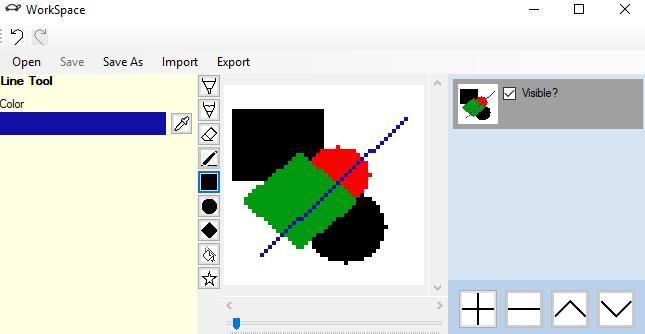
**Fulfils: B8**

Eraser tool for erasing the upper layer, revealing lower layer

**Fulfils: A6**

Save current workspace to a proprietary file

**Fulfils: C5**

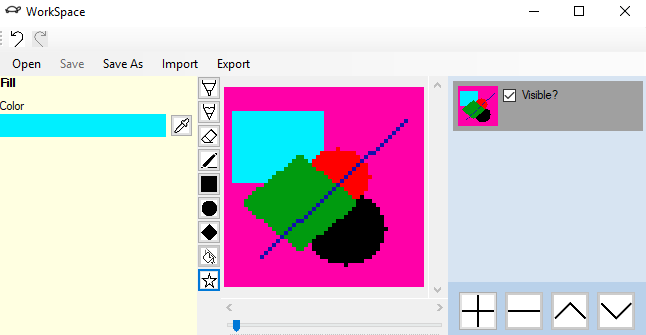


Shapes a drawn using pixels and not using vector structure

**Fulfils: B2**

Shape creation tools for making shapes

**Fulfils: A3**



Import from a PNG or JPEG file into workspace

**Fulfils: C2**

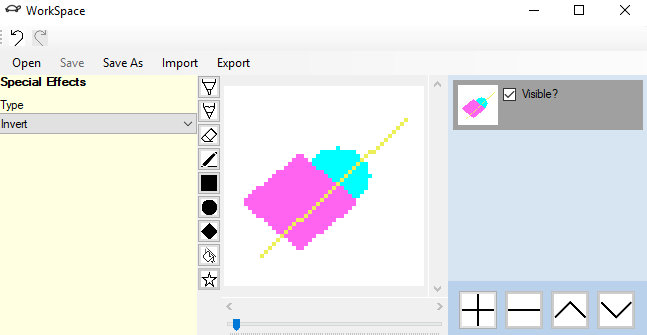
Fill tool that fills an area

**Fulfils: A4**

Image has been filled in a few places

Export image to a PNG or JPEG

**Fulfils: C3**

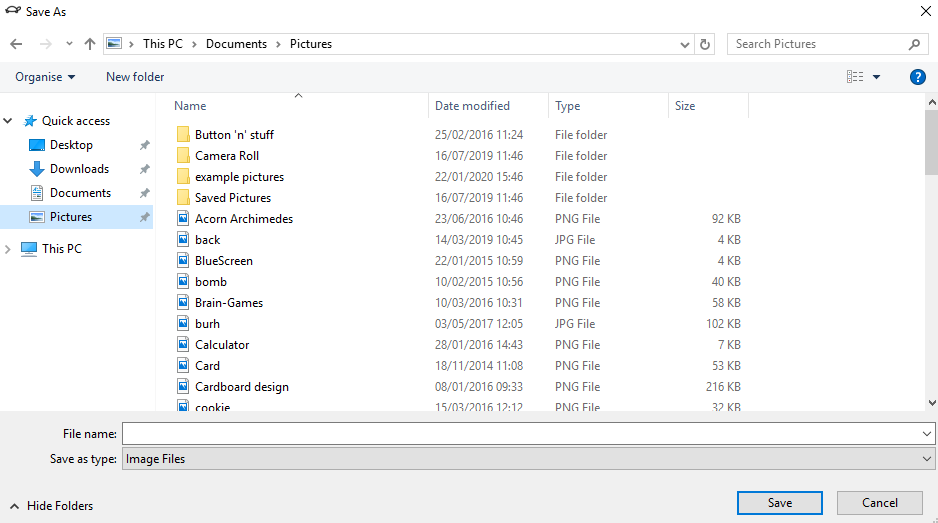


Shape creation tools for making shapes

**Fulfils: B12**

Image has been inverted from previous

SIMP Export Image Browser



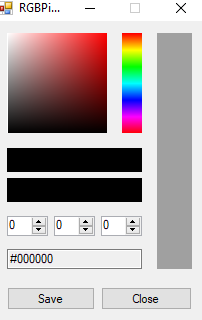
Allows user to input name of file to export, and type (PNG or JPEG)

## SIMP RGB Picker

Colour strip so user can define hue of colour

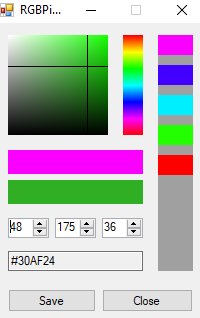
**Fulfils: B3**

Colour square to allow user to define shade of colour



R, G and B input to enter RGB code

**Fulfils: B4**



Crosshair gives user an indication of what colour they are using

RGB Code

Previous colour selector allows user to select a colour they used before

Current RGB colour

Previous RGB colour

## Full Criteria Fulfilment Diagram

|  |  |  |
| --- | --- | --- |
| **Feature** | **Fulfilled?** | **Code** |
| Section A - Brushes | | |
| 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 |
| Section B – Other editing tools | | |
| 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 |
| *Image effects* | Yes | B12 |
| *Rotating Images* | No | B13 |
| *Clipping masks* | No | B14 |
| Section C – File System | | |
| 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 |
| Section D – Usability | | |
| Program should be stable and not crash. | Yes | D1 |
| Program should be easy to use | Yes | D2 |
| Features should be easily accessible | Yes | D3 |

## Uncompleted Criteria

### Selection Tools (B6, B7)

Selection tools were not completed due to their potential complexity (all sorts of selection shapes may need to be accounted for) and because of this there would have been a large impact on SIMP’s performance. Performance is already an issue for SIMP due to the action logging system but a check for each pixel as to whether it is in the selection (which may, at worse, require iterating a list for each pixel) would be too much.

### Text (B10)

Text tools were not included due to the complexity of creating a font (needing to do a lot of work to implement 56 letters and some extra characters) and due to the small average size of a SIMP canvas, the text would be unlikely to be very legible.

Whilst there are some in-built classes in .NET for generating text, these are designed for larger sizes of text and would also be illegible or not fit in the canvas. There would also be an overhead in converting from .NET’s image system to SIMP’s image system that may make the conversion too slow to be usable.

### Rotating Images (B13)

Rotating images was not implemented due to it being a niche feature, not directly brought up by a stakeholder during development as a necessary feature for the next cycle. It did not fit into any previously made framework (would not work as an Image Effect (B12) as it is not a colour transformation) and so would require a new tool and some new framework to be possible. It also may have performance issues for larger images (as it changes the entire image at once) and would also require extra work from the layers to rotate each layer, leading to even worse performance in multi-layer images.

### Clipping Masks (B14)

Clipping masks was not implemented due to it being a very niche feature, only requested by one stakeholder in Analysis, and being rather complex to add, as it requires layers to be linked together such that one is the clipping mask of another. Coupled with fears of worse performance it was not suitable to add at this time.

### Completed Features

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | ID | Comment | Improvement |
| Variable brush width | A1 | Both the brush and the eraser can have their width edited using a field. | The brush width can currently only be whole integers, meaning there is some fine-tuning lost.  When using large brush sizes, the program slows down significantly. This is because the brush re-stamps its entire area each time it is moved, leading to many duplicate operations. |
| Hard brushes | A2 | All brushes are hard brushes, there is no transparency. | This feature was more to simplify design by not having to worry about colour mixing. Colour mixing however is quite an important feature in many art programs and could be implemented in the future. |
| Shape creation tools | A3 | There are tools for rectangle, circle and diamond shapes. | The shapes are fine and generate reasonably fast, however the shape selection is quite limited and there is no support for custom polygons. |
| Fill (bucket) tool | A4 | A fill tool will fill a certain shape of the same colour with a new colour. | The fill tool could support for thresholds, which is where it will fill every colour *close* to the target colour rather than just identical colours, to allow more fine-tuned filling.  The fill tool also has very poor performance for large areas, making it (for some stakeholders) unusable. The algorithm may be inefficient and could be improved. |
| Single pixel pencil | A5 | The pencil can be accessed for finer detail. | The pencil is currently an entirely separate tool from the normal brush, meaning you have to switch tools tool have a 1 pixel brush, and potentially have to copy across colour too. It would be better implemented as the minimum size of the normal brush tool. |
| Rubber | A6 | There is a rubber tool which can completely remove a pixel | Like above, a soft rubber may also have been useful (to make parts of a layer semi-transparent).  An indicator for what is about to be rubbed out could also be useful, so that it is easy to know what parts of the image are about to be removed |
| Image viewer | B1 | There is a viewer and scroll bars which can be used to manipulate the image | Currently, the image viewer can be obscured by some of the tabs in the workspace. It may be more useful that it is possible to hide these tabs so it is easier to edit the image:    Dock-able tabs could be an alternate implementation of this, which was brought up as a possibility by **Alex G** during [Analysis](#_Question_1:_What) |
| Bitmap image editor | B2 | SIMP uses the bitmap system of defined blocks of colour in an array of defined locations. | Whilst supporting vector images would not be practical in this program as it requires a completely different toolset to manipulate vector images, support to import / export to vector file formats such as SVG may have been a useful compromise. |
| RGB colour picker | B3 | There is an RGB colour picker in SIMP, with support for editing hues, shade and storing recent colours. | The form contains many controls, but no labels. This can make it hard to navigate for those not familiar with image editing programs. |
| RGB direct input | B4 | RGB colours can be entered using three numeric boxes | This implementation requires the user to convert from hexadecimal notation to decimal notation, and split it into three boxes. A text box for entering from an RGB code would be more useful, though this was not implemented as performing proper validation and interpretation on an RGB input can be a difficult task. |
| Layer system | B5 | SIMP has a layer panel where layers can be added, removed, and shifted up and down. | Currently the layer panel does not allow naming of layers, which can be quite a useful feature for images with many layers.  Another useful feature missing is layer merging, where two layers are combined into one layer depending on which one was on top, allowing both to be edited at once. |
| Transparent pixels | B8 | SIMP can support transparent pixels in a layer | On the image however, it is not possible to tell which pixels are transparent and which ones are white, as the default backdrop colour is white. A more distinct checkerboard pattern may make this difference easier to spot. |
| Zoom in (no zoom out) | B9 | SIMP can zoom in up to 50x, but cannot zoom out. | This 50x is however an arbitrary limit. The option to zoom in further (especially on large images) may be needed.  Zooming out may also be useful for large images, however this was not implemented due to making the calculations for drawing pixels more complicated, and determining click location inconsistent when multiple pixels occupy the same spot. |
| Eyedropper tool | B11 | The user can press the eyedropper tool attached to any colour property, then click a position on the image to set the colour. | The eyedropper tool could become a bit more intuitive. There is currently no indication of whether the eyedropper tool is selected or not and this can make the tool confusing for users.  The button becoming highlighted or the cursor changing to an eyedropper icon when hovering over the image would make the tool much more easy to use |
| Image effects | B12 | The user can select an image effect and an option from a drop down box, click the image and to apply it. | Currently there are only three image effects, more could be added in the future.  The current user cycle involves them selecting the wanted effect, and then needing to click the image like a normal brush. This could be improved to be an ‘Apply Effect’ button attached to the tool as it may not be obvious the image must be clicked. |
| Creating a new image | C1 | On the starting form, the user can define the width and height of the image and then press ‘Create Image’ to make a blank image | A preview section could be implemented so that the user can get a feel of how big the image is before its created.  The responsibility of creating an image also should be moved off of the starting form. This isn’t a problem right now as there are few image creating options but in the future as more is added a more organised system will be needed, such as a separate image creating form. |
| Importing images | C2 | The user can utilise a file browser to import an image into a new workspace or as a layer in an existing workspace. | Right now, it is not possible to specify the size of the image when it’s imported. When importing as a new image it will imported at its source size, when importing as a layer it will be resized to the current workspace size. Options to specify how large the image is when it is imported (especially for images that are too large) |
| Exporting images | C3 | The current workspace can be exported as a PNG, JPG or BMP image using a file browser. | The options to export a single layer at a time could be implemented.  As above, exporting at a specific size is also a feature that could be needed, especially as most SIMP images are small, and do not display well in many image viewers. |
| Supporting PNG and JPEG | C4 | SIMP supports PNG and JPEG by use of a selector in the file browser |  |
| Saving and loading from a proprietary format | C5 | SIMP can save and load using a .SIMP format. | The SIMP format is currently very simple, containing a few pieces of header information followed by all of the colours. This makes it a very inefficient format, a simple form of **Run-Length Encoding** could help reduce file sizes. |
| Program should be stable and not crash. | D1 | SIMP after many bugfixes has become very stable. | Unfortunately, to achieve this stability some harsh limits needed to be enforced, images can only be 1000px by 1000px maximum, many operations are slow. Most of this stems to issues with running out of memory, so optimisations could reduce this bottleneck. |
| Program should be easy to use | D2 | Stakeholders found SIMP’s features easy to find with minimal help. | A readme file or tutorial would still help, as being a stakeholder they may have already known some things from experience. |
| Features should be easily accessible | D3 | Almost all features are accessible with one click. | There are however no keyboard shortcuts, aside from ctrl+z, and ctrl+y (redo and undo), which can limit accessibility and workflow. |

# 6.3 Maintenance Plan

This also includes suggestions made in the prior section.

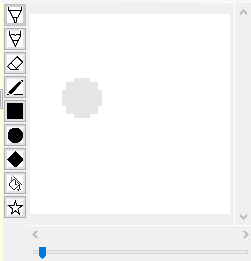
## Usability Improvements – How can the program be made easier to use

### Global colour property

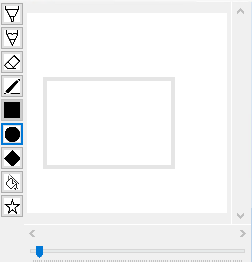
Many of my stakeholders noted that not having a global property (instead a local colour property for each tool) was very inconvenient, and is something that could definitely be improved upon.

### Tool preview

Currently, it is very difficult to tell what a tool is about to do before you perform the action, such as make a stroke. A tool indicator would help this somewhat, for example when about to make a stroke the following ghost circle, which tells the user how big the current brush is, could be displayed:



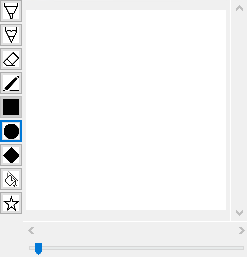
Or, when creating a shape, the shape outline could be shown so that the user knows what the shape will look like before comitting:



These are demonstration mock-ups, not programmed features.

### Starting strokes off edge of canvas

It is currently not possible to start any strokes off the edge of the canvas. Thus if the user wants to make a circle like this:

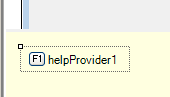


Or in other cases strokes made near the edge of the canvas will fail if the user misses slightly. In future support for clicking off the image can be provided, to solve both of these problems.

### Tutorial

Whilst SIMP, from stakeholder feedback, appears to be a self-explanatory program, a tutorial or readme file may be useful for those who do not use art programs very often.

This could be done through in-built help providers, such as .NET’s unit which shows help upon pressing F1:



### Accessibility Features

SIMP could also benefit from becoming more accessible.

For example, many of the buttons are very small, and may be difficult to press for those with poor motor ability, or poor eyesight. Screenreader support may also be useful here, such as it reading the name of each tool as it is hovered over. SIMP also isn’t optimised for tabbing (pressing the tab key to move from one option to the next) and this could be unintuitive for those who rely on tabbing.

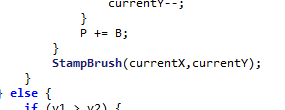
## Programmatic Improvements – How can the program be made to run better

### Fill tool

The fill tool is far too slow to be useful, especially on larger (and even medium) areas. The algorithm will need to be investigated to find out why its performance is much too slow, and rectified so that the fill tool can be properly used to fill larger areas.

### Large brushes

Large brushes are also much too slow, freezing the application if too large of a stroke is made. Unlike the fill tool however, the cause is known:



In the code for drawing a line, the code will ‘stamp’ a new brush circle at each point along the line. However as each point is adjacent, this leads to a lot of redundant stamping, for example, if each stamp were a circle:

The darker the area is, the more times that its colour is set. This unnecessary setting of pixels leads to a lot more calculations and storage (especially as each pixel set is recorded so that it can be undone).

To fix this, a more efficient stamping algorithm is needed such that it only sets each pixel once.

### Error handler

SIMP will occasionally still crash, especially if it runs out of memory. In these cases, it is of great importance that SIMP has a robust error handler for these scenarios. If there is a problem the program cannot recover from it needs to:

* Alert the user of this
* Allow them to save any unsaved worked
* Provide details of the error such that it can be reported and fixed

Right now, if SIMP encounters an error the program will simply close, losing any open work in the process.

### Multithreading

Currently SIMP is entirely single threaded, however there are multiple places where multi-threading can be used:

* Speeding up image display as multiple threads work together to generate the needed image. Each thread could be given a ‘block’ of the image to render, thus by adding all these blocks together you get the completed image
* Running the .NET window handling on a separate thread, such that when the program is processing a hard instruction, the window remains responsive as another thread is handling that.

Many computers will have access to multiple cores too, so it is imperative that SIMP be able to take advantage of this processing power such that it remains responsive and scalable.

### Decrease RAM usage

Aside from increasing algorithm speed, algorithm memory requirements also play a vital part in the overall program usability. This is especially true as SIMP suffers from using too much memory. For example, SIMP stores the image in a large array of colour objects, however this can be quite inefficient as a lot of the image contains the same colour. Thus the same colour is stored many thousands of times, causing the program to run out of memory if too many images are created. This could be improved such that each colour only exists once in memory, and the array instead contains references to these colours.

# Code Listing

## EAxis

**namespace** SIMP {  
    /// <**summary**>  
    /// Axis to edit in  
    /// <**/summary**>  
    **public** **enum** EAxis {  
        X,  
        Y  
    }  
}

## EAxisMode

**namespace** SIMP  
{  
    /// <**summary**>  
    /// The state of the image in relation to an axis  
    /// e.g. ImageTooLarge means the image size in that axis is larger than the amount of space to display it in,  
    /// thus must be trimmed  
    /// <**/summary**>  
    **public** **enum** EAxisMode  
    {  
        ImageTooLarge,  
        ImageTooSmall  
    }  
}

## Actions

### IAction

**namespace** SIMP.Actions  
{  
    /// <**summary**>  
    /// An action to be done (and undone) by the program  
    /// <**/summary**>  
    **public** interface IAction  
    {  
        void **Do**(Workspace workspace); // Perform the action on this workspace  
        void **Undo**(Workspace workspace);  
    }  
}

### PixelAction

**namespace** SIMP.Actions  
{  
    /// <**summary**>  
    /// An action that changes the pixels of the image  
    /// <**/summary**>  
    **public** class PixelAction : IAction  
    {  
        **private** Dictionary<FilePoint,Color> oldPixels;  
        **private** Dictionary<FilePoint,Color> newPixels;  
        **public** Layer layerPerformedOn = **null**;  
          
        **public** **PixelAction**()  
        {  
            oldPixels = **new** Dictionary<FilePoint, Color>();  
            newPixels = **new** Dictionary<FilePoint, Color>();  
        }  
          
        /// <**summary**>  
        /// Record a new pixel change  
        /// <**/summary**>  
        /// <**param** **name="pixelLocation"**>The location of this change<**/param**>  
        /// <**param** **name="oldColour"**>What it was before<**/param**>  
        /// <**param** **name="newColour"**>What it changes to<**/param**>  
        **public** void **AddPixel**(FilePoint pixelLocation, Color oldColour, Color newColour) {  
            // saves the old and new colours in the dictionary  
            oldPixels[pixelLocation] = oldColour;  
            newPixels[pixelLocation] = newColour;  
        }

        /// <**summary**>  
        /// Set the pixels  
        /// <**/summary**>  
        /// <**param** **name="workspace"**><**/param**>  
        **public** void **Do**(Workspace workspace) {  
            // Records the layer this action is performed on if necessary  
            **if** (layerPerformedOn == **null**) {  
                layerPerformedOn = workspace.image.currentLayer;  
            } **else** { // if there is already a layer recorded  
                // switch the current selected layer to the one this action is bound to  
                workspace.image.currentLayer = layerPerformedOn;  
            }  
              
            // goes through all the pixels and sets them  
            **foreach** (KeyValuePair<FilePoint,Color> pixel **in** newPixels) {  
                workspace.image.**SetPixel**(pixel.Key,pixel.Value);  
            }  
              
            workspace.**UpdateDisplayBox**(**true**,**false**);  
        }  
          
        **public** void **Undo**(Workspace workspace) {  
            // there must be a recorded layer now so no need to check  
            workspace.image.currentLayer = layerPerformedOn;  
            **foreach** (KeyValuePair<FilePoint,Color> pixel **in** oldPixels) {  
                workspace.image.**SetPixel**(pixel.Key,pixel.Value);  
            }  
              
            workspace.**UpdateDisplayBox**(**true**,**false**);  
        }  
    }  
}

## Properties

### EPropertyType

**namespace** SIMP.Properties {  
    /// <**summary**>  
    /// A type of a property  
    /// <**/summary**>  
    **public** **enum** PropertyType {  
        Normal, // Shown  
        ReadOnly, // Shown but cannot be edited  
        Hidden // Not shown  
    }  
}

### ColorProperty

**namespace** SIMP.Properties  
{  
    /// <**summary**>  
    /// Property that stores a colour  
    /// <**/summary**>  
    **public** class ColorProperty :IProperty  
    {  
        **public** **ColorProperty**(string name, Color **value**, PropertyType propertyType, Workspace myWorkspace)  
        {  
            **this**.name = name;  
            **this**.**value** = **value**;  
            **this**.myWorkspace = myWorkspace;  
            **this**.propertyType = propertyType;  
              
            **this**.onInteract = delegate(Object sender, EventArgs e) {  
                RGBPicker newPicker = **new** **RGBPicker**((Color)**this**.**value**,ColorCallback);  
                newPicker.**Show**();  
            };  
        }  
          
        // when this colour is updated  
        **private** void **ColorCallback**(Color newColour) {  
            **this**.**value** = newColour;  
            // make sure the tool display is updated  
            myWorkspace.**ShowTool**();  
        }  
    }  
}

### IProperty

**namespace** SIMP.Properties  
{  
    /// <**summary**>  
    /// Superclass for all properties  
    /// <**/summary**>  
    **public** abstract class IProperty  
    {  
        **public** string name;  
        **public** object **value**;  
        **public** EventHandler onInteract;  
        **public** Workspace myWorkspace;  
        **public** PropertyType propertyType;  
    }  
}

### DecimalProperty

**namespace** SIMP.Properties  
{  
    /// <**summary**>  
    /// Property that stores a Decimal  
    /// <**/summary**>  
    **public** class DecimalProperty : IProperty  
    {  
        **public** **decimal** min;  
        **public** **decimal** max;  
          
        **public** **DecimalProperty**(string name, **decimal** **value**, **decimal** min, **decimal** max, PropertyType propertyType, Workspace myWorkspace)  
        {  
            **this**.name = name;  
            **this**.**value** = **value**;  
            **this**.min = min;  
            **this**.max = max;  
            **this**.myWorkspace = myWorkspace;  
            **this**.propertyType = propertyType;  
              
            **this**.onInteract = delegate(Object sender, EventArgs e) {  
                **this**.**value** = (**decimal**)((NumericUpDown)sender).Value;  
                myWorkspace.**ShowTool**();  
            };  
        }  
    }  
}

### NumericalProperty

**namespace** SIMP.Properties  
{  
    /// <**summary**>  
    /// Property that stores a Number  
    /// <**/summary**>  
    **public** class NumericalProperty : IProperty  
    {  
        **public** **int** min;  
        **public** **int** max;  
          
        **public** **NumericalProperty**(string name, **int** **value**, **int** min, **int** max, PropertyType propertyType, Workspace myWorkspace)  
        {  
            **this**.name = name;  
            **this**.**value** = **value**;  
            **this**.min = min;  
            **this**.max = max;  
            **this**.myWorkspace = myWorkspace;  
            **this**.propertyType = propertyType;  
              
            // When the number box is changed  
            **this**.onInteract = delegate(Object sender, EventArgs e) {  
                // Update the internal value  
                **this**.**value** = (**int**)((NumericUpDown)sender).Value;  
                myWorkspace.**ShowTool**();  
            };  
        }  
    }  
}  
    

### ComboProperty

**namespace** SIMP.Properties  
{

 /// <**summary**>  
    /// Property that stores potential strings  
    /// <**/summary**>  
    **public** class ComboProperty : IProperty  
    {  
        **public** string[] options;  
          
        **public** **ComboProperty**(string name, **int** startIndex, string[] options, PropertyType propertyType, Workspace myWorkspace)  
        {  
            **this**.name = name;  
            **this**.**value** = **value**;  
            **this**.options = options;  
            **this**.**value** = options[startIndex];  
            **this**.myWorkspace = myWorkspace;  
            **this**.propertyType = propertyType;  
              
            // When the list box is changed  
            **this**.onInteract = delegate(Object sender, EventArgs e) {  
                // Update the internal value  
                **this**.**value** = (string)((ComboBox)sender).SelectedItem;  
                myWorkspace.**ShowTool**();  
            };  
        }  
    }  
}

## Shapes

### IPicturePoint

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A generic point on the image  
    /// <**/summary**>  
    **public** interface IPicturePoint  
    {  
        FilePoint **ToFilePoint**();  
        DisplayPoint **ToDisplayPoint**();  
    }  
}

### DisplayPoint

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A point relative to the display window, e.g 0,0 is top right of window  
    /// <**/summary**>  
    **public** class DisplayPoint  
    {  
        **public** **int** displayX;  
        **public** **int** displayY;  
          
        **public** **DisplayPoint**(**int** displayX, **int** displayY)  
        {  
            **this**.displayX = displayX;  
            **this**.displayY = displayY;  
        }  
          
        **public** **int** **GetDisplayValue**(EAxis axis) {  
            **switch** (axis) {  
                **case** EAxis.X:  
                    return displayX;  
                      
                **case** EAxis.Y:  
                    return displayY;  
                      
                **default**:  
                    **throw** **new** **Exception**("Invalid value for Axis");  
            }  
        }  
          
        **public** override string **ToString**()  
        {  
            return string.**Format**("[DisplayPoint DisplayX={0}, DisplayY={1}]", displayX, displayY);  
        }  
    }  
}

### FilePoint

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A point relative to the loaded image, e.g 0,0 is top right of image  
    /// <**/summary**>  
    **public** class FilePoint  
    {  
        **public** **int** fileX;  
        **public** **int** fileY;  
          
        **public** **FilePoint**(**int** fileX, **int** fileY)  
        {  
            **this**.fileX = fileX;  
            **this**.fileY = fileY;  
        }  
          
        **public** **FilePoint**(Point p) : **this**(p.X,p.Y) {  
            // no extra implementation  
        }  
          
        /// <**summary**>  
        /// Clones from an existing FilePoint  
        /// <**/summary**>  
        **public** **FilePoint**(FilePoint fp) : **this**(fp.fileX,fp.fileY) {  
            // no extra implementation  
        }  
          
        **public** **int** **GetFileValue**(EAxis axis) {  
            **switch** (axis) {  
                **case** EAxis.X:  
                    return fileX;  
                      
                **case** EAxis.Y:  
                    return fileY;  
                      
                **default**:  
                    **throw** **new** **Exception**("Invalid value for Axis");  
            }  
        }  
          
        **public** override string **ToString**()  
        {  
            return string.**Format**("[FilePoint FileX={0}, FileY={1}]", fileX, fileY);  
        }  
        

**public** override **int** **GetHashCode**()  
        {  
            return **base**.**GetHashCode**();  
        }  
  
         **public** override **bool** **Equals**(object obj)  
        {  
             FilePoint other = (FilePoint)obj;  
             return other.fileX == fileX && other.fileY == fileY;  
        }  
    }  
}

### IPictureRectangle

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A generic rectangle made of points  
    /// <**/summary**>  
    **public** interface IPictureRectangle  
    {  
        FileRectangle **ToFileRectangle**();  
        DisplayRectangle **ToDisplayRectangle**();  
    }  
}

### DisplayRectangle

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A rectangle made of display points  
    /// <**/summary**>  
    **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** DisplayRectangle **ToDisplayRectangle**() {  
            return **new** **DisplayRectangle**(  
                \_displayTopLeftCorner.**GetDisplayValue**(EAxis.X),  
                \_displayTopLeftCorner.**GetDisplayValue**(EAxis.Y),  
                \_displayBottomRightCorner.**GetDisplayValue**(EAxis.X),  
                \_displayBottomRightCorner.**GetDisplayValue**(EAxis.Y)  
            );  
        }  
          
        **public** FileRectangle **ToFileRectangle**() {  
            // not needed  
            return **null**;  
        }  
          
        **public** DisplayPoint **GetTopLeftCorner**() {  
            return \_displayTopLeftCorner;  
        }  
          
        **public** DisplayPoint **GetBottomRightCorner**() {  
            return \_displayBottomRightCorner;  
        }  
          
        **public** **int** **GetDisplaySize**(EAxis axis) {  
            return \_displayBottomRightCorner.**GetDisplayValue**(axis) - \_displayTopLeftCorner.**GetDisplayValue**(axis);  
        }  
    }  
}

### FileRectangle

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A rectangle made of display points  
    /// <**/summary**>  
    **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 **ToFileRectangle**() {  
            return **new** **FileRectangle**(  
                \_fileTopLeftCorner.**GetFileValue**(EAxis.X),  
                \_fileTopLeftCorner.**GetFileValue**(EAxis.Y),  
                \_fileBottomRightCorner.**GetFileValue**(EAxis.X),  
                \_fileBottomRightCorner.**GetFileValue**(EAxis.Y)  
            );  
        }  
          
        **public** DisplayRectangle **ToDisplayRectangle**() {  
            // not needed  
            return **null**;  
        }  
          
        **public** FilePoint **GetTopLeftCorner**() {  
            return \_fileTopLeftCorner;  
        }  
          
        **public** FilePoint **GetBottomRightCorner**() {  
            return \_fileBottomRightCorner;  
        }  
          
        **public** **int** **GetFileSize**(EAxis axis) {  
            return \_fileBottomRightCorner.**GetFileValue**(axis) - \_fileTopLeftCorner.**GetFileValue**(axis);  
        }  
    }  
}

## Tools

### ITool

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// A generic tool - describes the tasks a tool must handle  
    /// <**/summary**>  
    **public** abstract class ITool  
    {  
        **public** string name;  
        **public** string description;  
        **public** List<IProperty> properties; // properties that are displayed on side panel  
        **public** Workspace myWorkspace;  
        **public** System.Drawing.Image icon; // icon displayed in button  
          
        **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) {  
            **foreach** (IProperty property **in** properties) {  
                **if** (property.name.**Equals**(propertyName)) {  
                    return property;  
                }  
            }  
            **throw** **new** **KeyNotFoundException**("Couldn't find property " + propertyName);  
        }  
          
        //blanktool  
        **public** static ITool BlankTool;  
          
        static **ITool**() {  
            BlankTool = **new** **BlankTool**();  
        }  
    }  
}

### BlankTool

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// A null tool  
    /// Used for situations where a dummy is needed  
    /// <**/summary**>  
    **internal** class BlankTool : ITool  
    {  
        **internal** **BlankTool**()  
        {  
        }  
          
        // no implementation  
        // nothing happens ever  
        **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){}  
    }  
}

### EraserTool

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// Tool for erasing pixels - setting them to transparent  
    /// <**/summary**>  
    **public** class EraserTool : LineTool  
    {  
        **public** **EraserTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon)  
            : **base**(name,description,myWorkspace,icon)  
        {  
            **this**.properties.**Clear**(); // removes all previous properties then adds wanted ones  
            **this**.properties.**Add**(**new** **ColorProperty**("Color",Color.Transparent,PropertyType.Hidden,myWorkspace));  
            **this**.properties.**Add**(**new** **NumericalProperty**("Brush Size",5,1,20,PropertyType.Normal,myWorkspace));  
            **this**.icon = icon;  
        }  
    }  
}

### FXTool

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// Tool for apply effects to image  
    /// <**/summary**>  
    **public** class FXTool : ITool  
    {  
        **public** **FXTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon)  
        {  
            **this**.name = name;  
            **this**.description = description;  
            **this**.myWorkspace = myWorkspace;  
              
            **this**.properties = **new** List<SIMP.Properties.IProperty>();  
            **this**.properties.**Add**(**new** **ComboProperty**("Type",0,**new** string[] {"Grayscale","Black & White","Invert"}, PropertyType.Normal,myWorkspace));  
            **this**.icon = icon;  
        }  
          
        **public** override void **HandleMouseClick**(FilePoint clickLocation, MouseButtons button) {  
            PixelTransformation currentTransformation = **null**;  
              
            // checks what the combo box was set to  
            **switch** (**this**.**GetProperty**("Type").**value**.**ToString**()) {  
                **case** "Grayscale":  
                    currentTransformation = PixelTransformation.GrayScale;  
                    break;  
                **case** "Black & White":  
                    currentTransformation = PixelTransformation.BlackAndWhite;  
                    break;  
                **case** "Invert":  
                    currentTransformation = PixelTransformation.Invert;  
                    break;  
            }  
            

            PixelAction action = **new** **PixelAction**();  
              
            **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);  
                    // dont try to edit transparent pixels - leave them as is  
                    // otherwise background can look weird  
                    **if** (oldColor == Color.Transparent) {  
                        continue;  
                    }  
                    // applies the transformation delegate  
                    Color newColor = currentTransformation.**transform**(oldColor);  
                    action.**AddPixel**(**new** **FilePoint**(x,y),oldColor,newColor);  
                }  
            }  
              
            myWorkspace.**PerformAction**(action);  
            myWorkspace.**UpdateDisplayBox**(**true**,**true**);  
            myWorkspace.**UpdateLayersSelector**(**true**);  
        }  
          
        **public** override void **HandleMouseDown**(FilePoint clickLocation, MouseButtons button) {   
            // no implementation  
        }  
          
        **public** override void **HandleMouseUp**(FilePoint clickLocation, MouseButtons button) {  
            // no implementation  
        }  
          
        **public** override void **HandleMouseMove**(FilePoint oldLocation, FilePoint newLocation) {  
            // no implementation  
        }  
    }  
}

### PixelTransformation

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// A transformation applied to an input colour that returns an output colour  
    /// <**/summary**>  
    **public** class PixelTransformation  
    {  
        **public** delegate Color **Transformation**(Color input);  
        **public** Transformation transform;  
          
        // averages the R, G, and B values  
        **public** static PixelTransformation GrayScale {  
            get {  
                PixelTransformation transformation = **new** **PixelTransformation**();  
                transformation.transform = delegate(Color input) {  
                    **int** total = input.R + input.G + input.B;  
                    **int** average = total / 3;  
                    return Color.**FromArgb**(average,average,average);  
                };  
                return transformation;  
            }  
        }  
          
        // averages the R, G and B values and checks if they are above a certain threshold  
        **public** static PixelTransformation BlackAndWhite {  
            get {  
                PixelTransformation transformation = **new** **PixelTransformation**();  
                transformation.transform = delegate(Color input) {  
                    **int** total = input.R + input.G + input.B;  
                    // 381 is roughly (255 + 255 + 255) / 2, or mid-colour  
                    **if** (total < 381) {  
                        return Color.Black;  
                    } **else** {  
                        return Color.White;  
                    }  
                };  
                return transformation;  
            }  
        }  
        

        // subtracts each R, G and B value from 255  
        **public** static PixelTransformation Invert {  
            get {  
                PixelTransformation transformation = **new** **PixelTransformation**();  
                transformation.transform = delegate(Color input) {  
                    **int** newR = 255 - input.R;  
                    **int** newG = 255 - input.G;  
                    **int** newB = 255 - input.B;  
                    return Color.**FromArgb**(newR,newG,newB);  
                };  
                return transformation;  
            }  
        }  
    }  
}

### LineTool

**namespace** SIMP.Tools  
{  
    /// <**summary**>  
    /// The default brush  
    /// Not to be confused with the LineTool for drawing a straight line between two points  
    /// <**/summary**>  
    **public** class LineTool : ITool  
    {  
        **private** **bool** \_mouseDown;  
        **private** List<FilePoint> currentBrush;  
        **private** Dictionary<FilePoint, Color> setPixels;  
        **private** List<string> setHashes;  
        **private** Dictionary<FilePoint, Color> strokePixels;  
        **private** List<string> strokeHashes;  
          
        **public** **LineTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon)  
        {  
            **this**.name = name;  
            **this**.description = description;  
            **this**.myWorkspace = myWorkspace;  
            **this**.currentBrush = **new** List<FilePoint>();  
              
            **this**.properties = **new** System.Collections.Generic.List<IProperty>();  
            **this**.properties.**Add**(**new** **ColorProperty**("Color",Color.Black,PropertyType.Normal,myWorkspace));  
            **this**.properties.**Add**(**new** **NumericalProperty**("Brush Size",1,1,20,PropertyType.Normal,myWorkspace));  
              
            **this**.setPixels = **new** Dictionary<FilePoint, Color>();  
            **this**.setHashes = **new** List<string>();  
            **this**.strokePixels = **new** Dictionary<FilePoint, Color>();  
            **this**.strokeHashes = **new** List<string>();  
            **this**.icon = icon;  
        }  
          
        **public** override void **HandleMouseDown**(FilePoint clickLocation, MouseButtons button) {  
            // clickLocation may be null - off image  
            **if** (clickLocation != **null**) {  
                **GenerateBrush**();  
                **StampBrush**(clickLocation.fileX,clickLocation.fileY);  
                myWorkspace.**UpdateDisplayBox**(**true**,**false**);  
            }  
            \_mouseDown = **true**;  
        }  
        

**public** override void **HandleMouseUp**(FilePoint clickLocation, MouseButtons button) {  
            \_mouseDown = **false**;  
              
            Color myColor = (Color)**GetProperty**("Color").**value**;  
            PixelAction newAction = **new** **PixelAction**();  
              
            **foreach** (KeyValuePair<FilePoint,Color> pixel **in** strokePixels) {  
                newAction.**AddPixel**(pixel.Key,pixel.Value,myColor);  
            }  
            strokePixels.**Clear**();  
            strokeHashes.**Clear**();  
              
            newAction.layerPerformedOn = myWorkspace.image.currentLayer;  
            // records a seperate action that is the whole stroke  
            // this is so when undoing the entire stroke is undone  
            myWorkspace.**RecordAction**(newAction);  
            myWorkspace.**UpdateLayersSelector**(**false**);  
        }  
          
        **public** override void **HandleMouseClick**(FilePoint clickLocation, MouseButtons button) {  
            // no implementation  
        }  
          
        **public** override void **HandleMouseMove**(FilePoint oldLocation, FilePoint newLocation) {  
            **if** (\_mouseDown) {  
                Color myColor = (Color)**GetProperty**("Color").**value**;  
                **if** (**IsAdjacent**(oldLocation,newLocation)) {  
                    **StampBrush**(newLocation.fileX,newLocation.fileY);  
                } **else** {  
                    **StampBrush**(newLocation.fileX,newLocation.fileY);  
                    **DrawLine**(oldLocation,newLocation);  
                }  
                myWorkspace.**UpdateDisplayBox**(**true**,**false**);  
            }  
        }  
          
        **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**;  
        }  
        

**private** void **DrawLine**(FilePoint point1, FilePoint point2) {  
            // bresenhem's  
            **int** x1,x2,y1,y2;  
            **if** (point1.fileX <= point2.fileX) {  
                x1 = point1.fileX;  
                y1 = point1.fileY;  
                  
                x2 = point2.fileX;  
                y2 = point2.fileY;  
            } **else** {  
                x1 = point2.fileX;  
                y1 = point2.fileY;  
                  
                x2 = point1.fileX;  
                y2 = point1.fileY;  
            }  
              
            **if** (Math.**Abs**((x2 - x1)) > Math.**Abs**(y2 - y1)) {  
                **int** A = 2 \* Math.**Abs**(y2 - y1);  
                **int** B = A - (2 \* Math.**Abs**(x2 - x1));  
                **int** P = A - (x2 - x1);  
                  
                **int** currentY = y1;  
                **for** (**int** currentX = x1+1; currentX < x2; currentX++) {  
                    **if** (P < 0) {  
                        P += A;  
                    } **else** {  
                        **if** (y2 > y1) {  
                            currentY++;  
                        } **else** {  
                            currentY--;  
                        }  
                        P += B;  
                    }  
                    **StampBrush**(currentX,currentY);  
                }  
            }

**else** {  
                **if** (y1 > y2) {  
                    **int** temp = y1;  
                    y1 = y2;  
                    y2 = temp;  
                    temp = x1;  
                    x1 = x2;  
                    x2 = temp;  
                }  
                **int** A = 2 \* Math.**Abs**(x2 - x1);  
                **int** B = A - (2 \* Math.**Abs**(y2 - y1));  
                **int** P = A - (y2 - y1);  
                  
                **int** currentX = x1;  
                **for** (**int** currentY = y1+1; currentY < y2; currentY++) {  
                    **if** (P < 0) {  
                        P += A;  
                    } **else** {  
                        **if** (x2 > x1) {  
                            currentX++;  
                        } **else** {  
                            currentX--;  
                        }  
                        P += B;  
                    }  
                    **StampBrush**(currentX,currentY);  
                }  
            }  
        }  
          
        **private** void **GenerateBrush**() {  
            **this**.currentBrush = **new** List<FilePoint>();  
              
            **float** radius = Convert.**ToSingle**(**GetProperty**("Brush Size").**value**);  
              
            **for** (**float** x = radius \* -1; x < radius; x++) {  
                **for** (**float** y = radius \* -1; y < radius; y++) {  
                    **if** (**IsPointInCircle**(x + 0.5f, y + 0.5f, radius)) {  
                        currentBrush.**Add**(**new** **FilePoint**((**int**)x,(**int**)y));  
                    }  
                }  
            }  
        }  
        

**private** **bool** **IsPointInCircle**(**float** x, **float** y, **float** radius) {  
            // returns whether this condition is true or false  
            return ((x\*x)+(y\*y)) <= (radius \* radius);  
        }  
          
        **private** void **StampBrush**(**int** x, **int** y) {  
            Color myColor = (Color)**GetProperty**("Color").**value**;  
            Color currentColor;  
            FilePoint stampPoint;  
              
            **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;  
                }  
                currentColor = myWorkspace.image.**GetPixel**(stampPoint);  
                **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**());  
                    }  
                }  
            }  
              
            PixelAction newAction = **new** **PixelAction**();  
              
            **foreach** (KeyValuePair<FilePoint,Color> pixel **in** setPixels) {  
                newAction.**AddPixel**(pixel.Key,pixel.Value,myColor);  
            }  
            setPixels.**Clear**();  
            setHashes.**Clear**();  
              
            myWorkspace.**PerformActionSilent**(newAction);  
        }  
    }  
}

### ShapeTools

#### IPixelTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// A generic pixel based tool  
    /// <**/summary**>  
    **public** abstract class IPixelTool : ITool  
    {  
        **public** **IPixelTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) {  
            **this**.name = name;  
            **this**.description = description;  
            **this**.myWorkspace = myWorkspace;  
            **this**.properties = **new** System.Collections.Generic.List<IProperty>();  
            **this**.shapePoints = **new** List<FilePoint>();  
            **this**.icon = icon;  
        }  
          
        **internal** List<FilePoint> shapePoints;  
          
        **internal** abstract void **GenShape**();  
          
        **internal** abstract void **DrawShape**();  
          
        **internal** abstract void **AddShapePoint**(**int** x, **int** y);  
    }  
}

#### IShapeTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// A pixel tool that generates a shape  
    /// Implements everything except shape generation  
    /// This leaves shape generation up to children  
    /// <**/summary**>  
    **public** abstract class IShapeTool : IPixelTool  
    {  
        **public** **IShapeTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon)  
            : **base**(name,description,myWorkspace,icon) {  
            **this**.point1 = **null**;  
            **this**.point2 = **null**;  
            **this**.properties.**Add**(**new** **ColorProperty**("Color",Color.Black,PropertyType.Normal,myWorkspace));  
        }  
          
        **internal** FilePoint point1;  
        **internal** FilePoint point2;  
          
        **public** override void **HandleMouseMove**(FilePoint oldLocation, FilePoint newLocation)  
        {  
            // do nothing upon mouse move  
        }  
          
        **public** override void **HandleMouseClick**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            **if** (point1 == **null**) {  
                point1 = clickLocation;  
            } **else** {  
                point2 = clickLocation;  
                **ResolvePoints**();  
                **GenShape**();  
                **DrawShape**();  
                point1 = **null**;  
            }  
        }  
          
        **public** override void **HandleMouseUp**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            // do nothing upon mouse up  
        }  
        

**public** override void **HandleMouseDown**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            // do nothing upon mouse down  
        }  
          
        **internal** override void **AddShapePoint**(**int** x, **int** y)  
        {  
            shapePoints.**Add**(**new** **FilePoint**(x,y));  
        }  
          
        **internal** override void **DrawShape**()  
        {  
            Color myColor = (Color)**GetProperty**("Color").**value**;  
            PixelAction newAction = **new** **PixelAction**();  
              
            **foreach** (FilePoint shapePoint **in** shapePoints) {  
                newAction.**AddPixel**(shapePoint,myWorkspace.image.**GetPixel**(shapePoint),myColor);  
            }  
              
            myWorkspace.**PerformAction**(newAction);  
            shapePoints.**Clear**();  
            myWorkspace.**UpdateLayersSelector**(**false**);  
        }  
          
        // virtual as LineTool needs to override it  
        **internal** virtual 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);  
        }  
    }  
}

#### CircleTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// Tool for drawing a circle shape  
    /// <**/summary**>  
    **public** class CircleTool : IShapeTool  
    {  
        **public** **CircleTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) : **base** (name,description,myWorkspace,icon)  
        { }  
          
        /// <**summary**>  
        /// Generates the shape from its current parameters and stores it in a list of points  
        /// <**/summary**>  
        **internal** override void **GenShape**()  
        {  
            // Centre Location is halfway between the points clicked on the image  
            **double** centreLocX = ((point1.fileX + point2.fileX) / 2);  
            **double** centreLocY = ((point1.fileY + point2.fileY) / 2);  
            // Doesn't square root as result of the distance calculation also outputs a squared number  
            **double** widthSquared = Math.**Pow**((point2.fileX - (point1.fileX-1))/2,2);  
            **double** heightSquared = Math.**Pow**((point2.fileY - (point1.fileY-1))/2,2);  
              
            // Looks through every possible candidate point  
            **for** (**int** x = point1.fileX; x <= point2.fileX; x++) {  
                **for** (**int** y = point1.fileY; y <= point2.fileY; y++) {  
                    // if its distance to the centre is less than the radius of the circle  
                    **if** ((Math.**Pow**(x-centreLocX,2) / widthSquared) +  
                        (Math.**Pow**(y-centreLocY,2) / heightSquared) <= 1) {  
                        // its a point in the circle, add it  
                        **AddShapePoint**(x,y);  
                    }  
                }  
            }  
        }  
    }  
}

#### DiamondTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// Tool for drawing a diamond shape  
    /// <**/summary**>  
    **public** class DiamondTool : IShapeTool  
    {  
        **public** **DiamondTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) : **base** (name,description,myWorkspace,icon)  
        { }  
          
        /// <**summary**>  
        /// Generates the shape from its current parameters and stores it in a list of points  
        /// <**/summary**>  
        **internal** override void **GenShape**()  
        {  
            // This code is almost identical to the circle code, except not squared  
            // this results in a diamond shape  
            **double** centreLocX = ((point1.fileX + point2.fileX) / 2);  
            **double** centreLocY = ((point1.fileY + point2.fileY) / 2);  
            **double** width = (point2.fileX - point1.fileX)/2;  
            **double** height = (point2.fileY - point1.fileY)/2;  
              
            **for** (**int** x = point1.fileX; x <= point2.fileX; x++) {  
                **for** (**int** y = point1.fileY; y <= point2.fileY; y++) {  
                    **if** ((Math.**Abs**(((**double**)x + 0.5)-centreLocX) / width) +  
                        (Math.**Abs**(((**double**)y + 0.5)-centreLocY) / height) <= 1) {  
                        **AddShapePoint**(x,y);  
                    }  
                }  
            }  
        }  
    }  
}

#### LineTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// Tool for drawing a line between two points  
    /// <**/summary**>  
    **public** class LineTool : IShapeTool  
    {  
        **public** **LineTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) : **base** (name,description,myWorkspace,icon)  
        {  
        }  
          
        **internal** override void **ResolvePoints**() {  
            // do nothing  
            // this stops points being resolved, stopping the IShapeTool implementation  
        }  
          
        **internal** override void **GenShape**()  
        {  
            // makes the the start and end points are part of the line  
            **AddShapePoint**(point1.fileX,point1.fileY);  
            **AddShapePoint**(point2.fileX,point2.fileY);  
            // bresenhem's  
            **int** x1,x2,y1,y2;  
            **if** (point1.fileX <= point2.fileX) {  
                x1 = point1.fileX;  
                y1 = point1.fileY;  
                  
                x2 = point2.fileX;  
                y2 = point2.fileY;  
            } **else** {  
                x1 = point2.fileX;  
                y1 = point2.fileY;  
                  
                x2 = point1.fileX;  
                y2 = point1.fileY;  
            }  
            

**if** (Math.**Abs**((x2 - x1)) > Math.**Abs**(y2 - y1)) {  
                **int** A = 2 \* Math.**Abs**(y2 - y1);  
                **int** B = A - (2 \* Math.**Abs**(x2 - x1));  
                **int** P = A - (x2 - x1);  
                  
                **int** currentY = y1;  
                **for** (**int** currentX = x1+1; currentX < x2; currentX++) {  
                    **if** (P < 0) {  
                        P += A;  
                    } **else** {  
                        **if** (y2 > y1) {  
                            currentY++;  
                        } **else** {  
                            currentY--;  
                        }  
                        P += B;  
                    }  
                    **AddShapePoint**(currentX,currentY);  
                }  
            } **else** {  
                **if** (y1 > y2) {  
                    **int** temp = y1;  
                    y1 = y2;  
                    y2 = temp;  
                    temp = x1;  
                    x1 = x2;  
                    x2 = temp;  
                }  
                **int** A = 2 \* Math.**Abs**(x2 - x1);  
                **int** B = A - (2 \* Math.**Abs**(y2 - y1));  
                **int** P = A - (y2 - y1);  
                  
                **int** currentX = x1;  
                **for** (**int** currentY = y1+1; currentY < y2; currentY++) {  
                    **if** (P < 0) {  
                        P += A;  
                    } **else** {  
                        **if** (x2 > x1) {  
                            currentX++;  
                        } **else** {  
                            currentX--;  
                        }  
                        P += B;  
                    }  
                    **AddShapePoint**(currentX,currentY);  
                }  
            }  
        }  
    }  
}

#### FillTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// Tool for filling a large area of identical color  
    /// <**/summary**>  
    **public** class FillTool : IPixelTool  
    {  
        **private** FilePoint point1;  
        **private** List<FilePoint> fillPoints;  
        **private** List<string> hashedPoints;  
        **private** Queue<FilePoint> pointQueue;  
        **private** Color targetColor;  
          
        **public** **FillTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) : **base** (name,description,myWorkspace,icon)  
        {   
            **this**.fillPoints = **new** List<FilePoint>();  
            **this**.pointQueue = **new** Queue<FilePoint>();  
            **this**.hashedPoints = **new** List<string>();  
            **this**.properties.**Add**(**new** **ColorProperty**("Color",Color.Black,PropertyType.Normal,myWorkspace));  
        }  
          
        **internal** override void **AddShapePoint**(**int** x, **int** y)  
        {  
            // doesnt need to do anything  
        }  
          
        **internal** override void **GenShape**()  
        {  
            // Adds the starting point to the list of points  
            pointQueue.**Enqueue**(point1);  
            FilePoint currentPoint;  
              
            // While there are still points left to be checked  
            **while** (pointQueue.Count > 0) {  
                currentPoint = pointQueue.**Dequeue**();  
                // Attempt to add the 4 tiles around it  
                **AddTile**(**new** **FilePoint**(currentPoint.fileX+1,currentPoint.fileY));  
                **AddTile**(**new** **FilePoint**(currentPoint.fileX,currentPoint.fileY+1));  
                **AddTile**(**new** **FilePoint**(currentPoint.fileX-1,currentPoint.fileY));  
                **AddTile**(**new** **FilePoint**(currentPoint.fileX,currentPoint.fileY-1));  
            }  
        }

        /// <**summary**>  
        /// May or may not add the point  
        /// <**/summary**  
        **private** void **AddTile**(FilePoint point) {  
            **if** (**CheckAddTile**(point)) {  
                // check any points around this one  
                pointQueue.**Enqueue**(point);  
                shapePoints.**Add**(point);  
                // to make searching faster - a list of strings is used to check whether a point has been added  
                hashedPoints.**Add**(point.**ToString**());  
            }  
        }  
          
        **private** **bool** **CheckAddTile**(FilePoint point) {  
            // if the list hash doesnt already contain the point  
            **if** (hashedPoints.**Contains**(point.**ToString**())) {  
                return **false**;  
            }  
              
            // if out of bounds              
            **if** (point.fileX < 0 ||  
                point.fileY < 0 ||  
                point.fileX >= myWorkspace.image.fileWidth ||  
                   point.fileY >= myWorkspace.image.fileHeight) {  
                return **false**;  
            }  
              
            // if not of the same color as the start color  
            **if** (myWorkspace.image.**GetPixel**(point) != targetColor) {  
                return **false**;  
            }  
              
            return **true**;  
        }  
        

        /// <**summary**>  
        /// Prints the generated shape onto canvas  
        /// <**/summary**>  
        **internal** override void **DrawShape**()  
        {  
            Color myColor = (Color)**GetProperty**("Color").**value**;  
            PixelAction newAction = **new** **PixelAction**();  
              
            **foreach** (FilePoint shapePoint **in** shapePoints) {  
                newAction.**AddPixel**(shapePoint,myWorkspace.image.**GetPixel**(shapePoint),myColor);  
            }  
              
            myWorkspace.**PerformAction**(newAction);  
            shapePoints.**Clear**();  
            myWorkspace.**UpdateLayersSelector**(**false**);  
        }  
          
        **public** override void **HandleMouseMove**(FilePoint oldLocation, FilePoint newLocation)  
        {  
            //none  
        }  
          
        **public** override void **HandleMouseClick**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            targetColor = myWorkspace.image.**GetPixel**(clickLocation);  
            // reset everything  
            fillPoints = **new** List<FilePoint>();  
            pointQueue = **new** Queue<FilePoint>();  
            hashedPoints = **new** List<string>();  
            point1 = **new** **FilePoint**(clickLocation.fileX,clickLocation.fileY);  
            **GenShape**();  
            **DrawShape**();  
        }  
          
        **public** override void **HandleMouseUp**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            //none  
        }  
          
        **public** override void **HandleMouseDown**(FilePoint clickLocation, System.Windows.Forms.MouseButtons button)  
        {  
            //none  
        }  
    }  
}

#### RectangleTool

**namespace** SIMP.Tools.ShapeTools  
{  
    /// <**summary**>  
    /// Tool for drawing a rectangle  
    /// <**/summary**>  
    **public** class RectangleTool : IShapeTool  
    {  
        **public** **RectangleTool**(string name, string description, Workspace myWorkspace, System.Drawing.Image icon) : **base** (name,description,myWorkspace,icon)  
        { }  
          
        **internal** override void **GenShape**() {  
            // simplest generation - add everything between the two points  
            **for** (**int** x = point1.fileX; x <= point2.fileX; x++) {  
                **for** (**int** y = point1.fileY; y <= point2.fileY; y++) {  
                    **AddShapePoint**(x,y);  
                }  
            }  
        }  
    }  
}

## Image

**namespace** SIMP  
{  
    /// <**summary**>  
    /// A loaded image  
    /// <**/summary**>  
    **public** class Image : IPictureRectangle  
    {  
        **public** **int** fileWidth, fileHeight;  
        **public** ZoomSettings zoomSettings;  
        **public** Workspace attachedWorkspace;  
        **public** List<FilePoint> changedPixels;  
        **public** Bitmap lastImage;  
        **public** List<Layer> layers;  
        **public** Layer currentLayer;  
          
        **public** **int** displayWidth { get {  
            return fileWidth \* zoomSettings.zoom;  
        }}  
        **public** **int** displayHeight { get {  
            return fileHeight \* zoomSettings.zoom;  
        }}  
          
        **public** **Image**(**int** width, **int** height, Workspace sender)  
        {  
            // prevent invalid widths & heights  
            **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**.fileWidth = width;  
            **this**.fileHeight = height;  
            **this**.attachedWorkspace = sender;  
            // centre is set to start in the middle of the image  
            **this**.zoomSettings = **new** **ZoomSettings**(**new** **FilePoint**(width/2,height/2));  
            **this**.changedPixels = **new** List<FilePoint>();  
              
            //layer setup  
            **this**.layers = **new** List<Layer>();  
            **this**.layers.**Add**(**new** **Layer**(fileWidth,fileHeight));  
            **this**.currentLayer = layers[0];  
        }  
          
        /// <**summary**>  
        /// Gets the colour of a pixel  
        /// <**/summary**>  
        /// <**param** **name="x"**>X coord of pixel to get<**/param**>  
        /// <**param** **name="y"**>Y coord of pixel to get<**/param**>  
        /// <**returns**>The colour of the selected pixel<**/returns**>  
        **public** Color **GetPixel**(**int** x, **int** y) {  
            return **GetPixel**(**new** **FilePoint**(x,y));  
        }  
          
        /// <**summary**>  
        /// Gets the colour of a pixel  
        /// <**/summary**>  
        /// <**param** **name="x"**>X coord of pixel to get<**/param**>  
        /// <**param** **name="y"**>Y coord of pixel to get<**/param**>  
        /// <**returns**>The colour of the selected pixel<**/returns**>  
        **public** Color **GetPixel**(FilePoint filePoint) {  
            return currentLayer.pixels[filePoint.fileX,filePoint.fileY].Color;  
        }  
          
        /// <**summary**>  
        /// Sets the colour of a pixel  
        /// <**/summary**>  
        /// <**param** **name="filePoint"**>File Point of pixel to set<**/param**>  
        /// <**param** **name="colour"**>The colour to set that pixel to<**/param**>  
        **public** void **SetPixel**(FilePoint filePoint, Color colour) {  
            **try** {  
                currentLayer.pixels[filePoint.fileX,filePoint.fileY] = **new** **SolidBrush**(colour);  
                changedPixels.**Add**(**new** **FilePoint**(filePoint));  
            } **catch** (IndexOutOfRangeException) {}  
        }  
          
        /// <**summary**>  
        /// Sets the colour of a pixel  
        /// <**/summary**>  
        /// <**param** **name="x"**>X coord of pixel to set<**/param**>  
        /// <**param** **name="y"**>Y coord of pixel to set<**/param**>  
        /// <**param** **name="colour"**>The colour to set that pixel to<**/param**>  
        **public** void **SetPixel**(**int** x, **int** y, Color colour) {  
            **SetPixel**(**new** **FilePoint**(x,y),colour);  
        }  
          
        /// <**summary**>  
        /// Sets the colour of a pixel  
        /// <**/summary**>  
        /// <**param** **name="displayPoint"**>The coords of pixel to set<**/param**>  
        /// <**param** **name="colour"**>The colour to set that pixel to<**/param**>  
        **public** void **SetPixel**(DisplayPoint displayPoint, Color colour) {  
            FilePoint filePoint = **DisplayPointToFilePoint**(displayPoint);  
            **SetPixel**(filePoint,colour);  
        }  
        

        /// <**summary**>  
        /// Gets a representation of this image  
        /// <**/summary**>  
        /// <**returns**>A System.Drawing.Image representation<**/returns**>  
        **public** System.Drawing.Image **GetDisplayImage**(**int** width, **int** height, **bool** full) {  
            zoomSettings.**CalcDisplayCentreLocation**(width,height);  
              
            // finds locations of where it needs to draw to and from  
            FilePoint topLeftPoint = **new** **FilePoint**(  
                zoomSettings.fileCentreLocation.fileX - (**int**)Math.**Ceiling**(Math.**Ceiling**((**decimal**)width/(**decimal**)zoomSettings.zoom)/2),  
                zoomSettings.fileCentreLocation.fileY - (**int**)Math.**Ceiling**(Math.**Ceiling**((**decimal**)height/(**decimal**)zoomSettings.zoom)/2)  
            );  
            FilePoint bottomRightPoint = **new** **FilePoint**(  
                zoomSettings.fileCentreLocation.fileX + (**int**)Math.**Ceiling**(Math.**Ceiling**((**decimal**)width/(**decimal**)zoomSettings.zoom)/2),  
                zoomSettings.fileCentreLocation.fileY + (**int**)Math.**Ceiling**(Math.**Ceiling**((**decimal**)height/(**decimal**)zoomSettings.zoom)/2)  
            );  
              
            // clamp both potential points inside the image  
            **ClampFilePoint**(**ref** topLeftPoint);  
            **ClampFilePoint**(**ref** bottomRightPoint);  
              
            // also clamp the centre location in the image  
            **ClampCentreLocation**(width,height);  
              
            // adjusts the centre location if the image is odd  
            **if** (width == displayWidth && fileWidth%2 == 1) {  
                zoomSettings.displayCentreLocation.displayX -= zoomSettings.zoom/2;  
            }  
              
            **if** (height == displayHeight && fileHeight%2 == 1) {  
                zoomSettings.displayCentreLocation.displayY -= zoomSettings.zoom/2;  
            }  
              
            **if** (full) { // if the full image is requested  
                Bitmap newImage = **new** **Bitmap**(width,height);  
                Graphics GFX = Graphics.**FromImage**(newImage);  
                GFX.**Clear**(Color.White);  
                **DrawFullImage**(topLeftPoint,bottomRightPoint,GFX);  
                lastImage = newImage;  
                return newImage;  
            } **else** { // else if only changes since last display is requested  
                Graphics GFX = Graphics.**FromImage**(lastImage);  
                **DrawChangedImage**(GFX);  
                return lastImage;  
            }  
              
        }  
          
        **private** void **DrawFullImage**(FilePoint topLeftPoint, FilePoint bottomRightPoint, Graphics GFX) {  
            // the file location of the top right  
            DisplayPoint topLeftDisplayPoint = **FilePointToDisplayPoint**(topLeftPoint);  
              
            // a point that is used within the loop  
            DisplayPoint currentPoint = **new** **DisplayPoint**(0,0);  
              
            // resets the X  
            currentPoint.displayX = topLeftDisplayPoint.displayX;  
              
            // 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++) {  
                      
                    **foreach** (Layer layer **in** layers) {  
                        //don't draw invisible layer  
                        **if** (!layer.visible) {  
                            continue;  
                        }  
                        // repeats through layers until a solid pixel is found  
                        **if** (layer.pixels[x,y].Color.A != 0) {  
                            // 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**(layer.pixels[x,y],currentPoint.displayX,currentPoint.displayY,zoomSettings.zoom,zoomSettings.zoom);  
                            break; // a pixel has been drawn - draw no more here  
                        }  
                    }  
                      
                    // 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;  
            }  
        }  
        

**private** void **DrawChangedImage**(Graphics GFX) {  
            DisplayPoint displayChangedLoc;  
            **bool** setPixel;  
            **foreach** (FilePoint changedPixel **in** changedPixels) {  
                // finds this pixels location  
                displayChangedLoc = **FilePointToDisplayPoint**(changedPixel);  
                setPixel = **false**;  
                **foreach** (Layer layer **in** layers) {  
                    //don't draw invisible layer  
                    **if** (!layer.visible) {  
                        continue;  
                    }  
                      
                    // repeats through layers until a solid pixel is found  
                    **if** (layer.pixels[changedPixel.fileX,changedPixel.fileY].Color != Color.Transparent) {  
                        GFX.**FillRectangle**(layer.pixels[changedPixel.fileX,changedPixel.fileY],  
                                          displayChangedLoc.displayX,displayChangedLoc.displayY,  
                                          zoomSettings.zoom,zoomSettings.zoom);  
                        setPixel = **true**;  
                        break;  
                    }  
                }  
                // if none of the layers mention that pixel  
                **if** (!setPixel) {  
                    // make sure that its set to white - if the pixel has been erased  
                    // this is not needed when displaying the full image as the image starts as a blank canvas  
                    // but here the previous image is being edited so must make sure this is changed  
                    GFX.**FillRectangle**(  
                        **new** **SolidBrush**(Color.White),  
                        displayChangedLoc.displayX,displayChangedLoc.displayY,  
                        zoomSettings.zoom,zoomSettings.zoom  
                    );  
                }  
            }  
            changedPixels.**Clear**();  
        }  
          
        **public** void **SetBarValues**(ScrollBar barHorizontal, ScrollBar barVertical) {  
            // determines how many missing centre locations there are  
            **int** missingCrossexXAxis = **GetMissingCrossesInAxis**(attachedWorkspace.displayBox.Width);  
            **int** missingCrossexYAxis = **GetMissingCrossesInAxis**(attachedWorkspace.displayBox.Height);  
              
            // updates horizontal bar  
            barHorizontal.LargeChange = missingCrossexXAxis;  
            barHorizontal.Maximum = fileWidth-1;  
            barHorizontal.Value = **CentreToBar**(EAxis.X);  
              
            // updates vertical bar  
            barVertical.LargeChange = missingCrossexYAxis;  
            barVertical.Maximum = fileHeight-1;  
            barVertical.Value = **CentreToBar**(EAxis.Y);  
        }  
          
        /// <**summary**>  
        /// Change the zoomSettings centre location based on the value of the bar  
        /// Used for when a bar is changed by user and needs this to be reflected in image  
        /// <**/summary**>  
        /// <**param** **name="newValue"**>Value bar has been changed to<**/param**>  
        /// <**param** **name="axis"**>Axis of the bar<**/param**>  
        **public** void **CentreFromBar**(**int** newValue, EAxis axis) {  
            **int** firstCrossPosition;  
            **switch** (axis) {  
                **case** EAxis.X:  
                    // check design docs for a more full explanation of the reasoning behind these sums  
                    firstCrossPosition = (**int**)Math.**Floor**(Math.**Floor**((**decimal**)attachedWorkspace.displayBox.Width/(**decimal**)zoomSettings.zoom)/2);  
                    zoomSettings.fileCentreLocation = **new** **FilePoint**(firstCrossPosition + newValue, zoomSettings.fileCentreLocation.fileY);  
                    break;  
                **case** EAxis.Y:  
                    firstCrossPosition = (**int**)Math.**Floor**(Math.**Floor**((**decimal**)attachedWorkspace.displayBox.Height/(**decimal**)zoomSettings.zoom)/2);  
                    zoomSettings.fileCentreLocation = **new** **FilePoint**(zoomSettings.fileCentreLocation.fileX, firstCrossPosition + newValue);  
                    break;  
            }  
        }  
          
        /// <**summary**>  
        /// Determine what value the bar should have  
        /// <**/summary**>  
        /// <**param** **name="axis"**>Axis<**/param**>  
        /// <**returns**>Integer representing what to set the bar's value to<**/returns**>  
        **public** **int** **CentreToBar**(EAxis axis) {  
            **int** firstCrossPosition;  
            **switch** (axis) {  
                **case** EAxis.X:  
                    firstCrossPosition = **GetFirstCrossPosition**(attachedWorkspace.displayBox.Width);  
                    return zoomSettings.fileCentreLocation.fileX - firstCrossPosition;  
                **case** EAxis.Y:  
                    firstCrossPosition = **GetFirstCrossPosition**(attachedWorkspace.displayBox.Height);  
                    return zoomSettings.fileCentreLocation.fileY - firstCrossPosition;  
            }  
            return 0;  
        }  
          
        **private** **int** **GetFirstCrossPosition**(**int** axisSize) {  
            return (**int**)Math.**Floor**(Math.**Floor**((**decimal**)axisSize/(**decimal**)zoomSettings.zoom)/2);  
        }  
          
        **private** **int** **GetMissingCrossesInAxis**(**int** axisSize) {  
            return (**int**)Math.**Floor**(Math.**Floor**((**decimal**)axisSize/(**decimal**)zoomSettings.zoom)/2)\*2;  
        }  
          
        /// <**summary**>  
        /// Returns a DisplayRectangle representation of the image  
        /// <**/summary**>  
        **public** DisplayRectangle **ToDisplayRectangle**() {  
            return **new** **DisplayRectangle**(0,0,displayWidth,displayHeight);  
        }  
          
        /// <**summary**>  
        /// Returns a FileRectangle representation of the image  
        /// <**/summary**>  
        **public** FileRectangle **ToFileRectangle**() {  
            return **new** **FileRectangle**(0,0,fileWidth,fileHeight);  
        }  
          
        /// <**summary**>  
        /// Converts a FilePoint to a DisplayPoint  
        /// <**/summary**>  
        /// <**param** **name="filePoint"**>FilePoint to convert<**/param**>  
        /// <**returns**>The FilePoint converted into a DisplayPoint<**/returns**>  
        **public** DisplayPoint **FilePointToDisplayPoint**(FilePoint filePoint) {  
            // checks whether the point is in bounds  
            **if** (  
                filePoint.fileX < 0 || filePoint.fileX >= fileWidth  
                || filePoint.fileY < 0 || filePoint.fileY >= fileHeight  
            ) {  
                **throw** **new** **IndexOutOfRangeException**();  
            }  
              
            // finds displacement  
            **int** displacementX = filePoint.fileX - zoomSettings.fileCentreLocation.fileX;  
            **int** displacementY = filePoint.fileY - zoomSettings.fileCentreLocation.fileY;  
              
            // factors zoom  
            displacementX \*= zoomSettings.zoom;  
            displacementY \*= zoomSettings.zoom;  
              
            **if** (zoomSettings.displayCentreLocation.displayX == 5) {  
                zoomSettings.displayCentreLocation.displayX = 4;  
            }  
              
            // returns the new point  
            return **new** **DisplayPoint**(  
                // this SHOULD be 4, not 5  
                // perhaps the code to cacluate where the centre location is on odd number size images isn't quite correct.. hmmm  
                zoomSettings.displayCentreLocation.displayX + displacementX,  
                zoomSettings.displayCentreLocation.displayY + displacementY  
            );  
        }  
          
        /// <**summary**>  
        /// Converts a DisplayPoint to a FilePoint  
        /// <**/summary**>  
        /// <**param** **name="displayPoint"**>DisplayPoint to convert<**/param**>  
        /// <**returns**>The DisplayPoint converted into a FilePoint<**/returns**>  
        **public** 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)  
            );  
        }  
        /// <**summary**>  
        /// edits the filepoint such that it is in the image bounds  
        /// <**/summary**>  
        **private** void **ClampFilePoint**(**ref** FilePoint filePoint) {  
            **if** (filePoint.fileX < 0) {  
                filePoint.fileX = 0;  
            }  
            **if** (filePoint.fileX > fileWidth) {  
                filePoint.fileX = fileWidth;  
            }  
              
            **if** (filePoint.fileY < 0) {  
                filePoint.fileY = 0;  
            }  
            **if** (filePoint.fileY > fileHeight) {  
                filePoint.fileY = fileHeight;  
            }  
        }  
        

        /// <**summary**>  
        /// changes the centrelocation when the image size reduces  
        /// this is so centre location is always somewhere on the image  
        /// <**/summary**>  
        **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);  
            }  
              
            // makes sure on odd sized images the centre is correct when zoomed out completely  
            **if** (displayWidth == width) {  
                zoomSettings.displayCentreLocation.displayX = width/2;  
            }  
              
            **if** (displayHeight == height) {  
                zoomSettings.displayCentreLocation.displayX = width/2;  
            }  
        }  
    }  
}

## Layer

**namespace** SIMP  
{  
    /// <**summary**>  
    /// Description of Layer.  
    /// <**/summary**>  
    **public** class Layer  
    {  
        **public** SolidBrush[,] pixels;  
        **public** **bool** visible;  
          
        **public** **Layer**(**int** width, **int** height)  
        {  
            // dimension array  
            pixels = **new** SolidBrush[width,height];  
            visible = **true**;  
              
            // populate array  
            **for** (**int** x = 0; x < width; x++) {  
                **for** (**int** y = 0; y < height; y++) {  
                    pixels[x,y] = **new** **SolidBrush**(Color.Transparent);  
                }  
            }  
        }  
    }  
}

## MainForm

**namespace** SIMP  
{  
    /// <**summary**>  
    /// Description of MainForm.  
    /// <**/summary**>  
    **public** partial class MainForm : Form  
    {  
        **public** Color testColor;  
        **public** Color testColor2;  
          
        **public** **MainForm**()  
        {  
            **InitializeComponent**();  
              
            // populates numeric boxes with the needed constants  
            numWidth.Maximum = SimpConstants.IMAGE\_MAX\_WIDTH;  
            numWidth.Value = (**decimal**)SimpConstants.IMAGE\_DEFAULT\_WIDTH;  
              
            numHeight.Maximum = SimpConstants.IMAGE\_MAX\_HEIGHT;  
            numHeight.Value = (**decimal**)SimpConstants.IMAGE\_DEFAULT\_HEIGHT;  
        }  
          
        void **BtnCreateClick**(object sender, EventArgs e)  
        {  
            Form newForm = **new** **Workspace**((**int**)numWidth.Value,(**int**)numHeight.Value);  
            newForm.**Show**();  
        }  
          
        void **BtnOpenClick**(object sender, EventArgs e)  
        {  
            // prompts the user for location of file to open  
            DialogResult result = diaOpen.**ShowDialog**();  
            **if** (result == DialogResult.OK) { // if they pressed ok and not cancel or close  
                **using** (FileStream stream = **new** **FileStream**(diaOpen.FileName,FileMode.Open)) {  
                    Workspace newWorkspace = Workspace.**OpenFile**(stream);  
                    **if** (newWorkspace != **null**) {  
                        newWorkspace.**MarkSavedTo**(diaOpen.SafeFileName,diaOpen.FileName);  
                    }  
                }  
            }  
        }  
          
        void **BtnImportClick**(object sender, EventArgs e)  
        {  
            DialogResult result = diaImport.**ShowDialog**();  
            **if** (result == DialogResult.OK) {  
                // creates a bitmap from that file location  
                Bitmap fileImage = **new** **Bitmap**(diaImport.FileName);  
                // if the area of the image is larger than the area of allowed maximums  
                **if** (fileImage.Width \* fileImage.Height > SimpConstants.IMAGE\_MAX\_WIDTH \* SimpConstants.IMAGE\_MAX\_HEIGHT  
                    // or if one of the dimensions is too large  
                   || fileImage.Width > SimpConstants.IMAGE\_MAX\_WIDTH  
                   || fileImage.Height > SimpConstants.IMAGE\_MAX\_HEIGHT  
                  ) {  
                    MessageBox.**Show**("Image was too large to be imported!","Image Size Error!",MessageBoxButtons.OK,MessageBoxIcon.Error);  
                    return;  
                }  
                  
                Workspace newForm = **new** **Workspace**(fileImage.Width,fileImage.Height);  
                  
                // creates an action that prints the opened image onto the internal image  
                PixelAction action = **new** **PixelAction**();  
                **for** (**int** x = 0; x < fileImage.Width; x++) {  
                    **for** (**int** y = 0; y < fileImage.Height; y++) {  
                        action.**AddPixel**(**new** **FilePoint**(x,y),Color.White,fileImage.**GetPixel**(x,y));  
                    }  
                }  
                  
                // performs it silently so you cant undo  
                newForm.**PerformActionSilent**(action);  
                  
                newForm.**Show**();  
            }  
        }  
    }  
}

## RGBPicker

**namespace** SIMP  
{  
    /// <**summary**>  
    /// Description of RGBPicker.  
    /// <**/summary**>  
    **public** partial class RGBPicker : Form  
    {  
        **private** **bool** \_updating;  
        **private** HSVColor \_startColor;  
        **private** HSVColor \_currentColor;  
        **private** ColorCallback \_updateCallback;  
        **private** static RGBPicker currentPicker = **null**;  
        **private** static List<Color> previousColors;  
          
        static **RGBPicker**() {  
            previousColors = **new** List<Color>();  
        }  
          
        **public** **RGBPicker**(Color startColor, ColorCallback updateCallback)  
        {  
            **InitializeComponent**();  
              
            // makes it so there is only one RGBPicker open at once  
            **if** (currentPicker != **null**) {  
                currentPicker.**Close**();  
            }  
            currentPicker = **this**;  
              
            \_startColor = SIMPRGB.**RGBtoHSV**(startColor);  
            \_currentColor = SIMPRGB.**RGBtoHSV**(startColor);  
            \_updating = **false**;  
            \_updateCallback = updateCallback;  
            SIMPRGB.cachedColor = startColor;  
              
            **Update**(**true**);  
        }  
        

        #**region** Update  
        /// <**summary**>  
        /// Updates all the displays  
        /// For when a value has been changed  
        /// <**/summary**>  
        /// <**param** **name="updateNums"**>Whether to update the number boxes or not<**/param**>  
        **private** void **Update**(**bool** updateNums) {  
            \_updating = **true**;  
              
            **UpdateColourSquare**();  
            **UpdateColourIndicator**();  
            **UpdateColourStrip**();  
            **UpdateColourDisplays**();  
            **UpdateColourInputs**(updateNums);  
            **UpdatePreviousColours**();  
              
            \_updating = **false**;  
        }  
          
        **double** oldHue = -1;  
        System.Drawing.Image oldImage;  
        **private** void **UpdateColourSquare**() {  
            **if** (\_currentColor.H.**Equals**(oldHue)) {  
                return;  
            }  
            **double** hue = \_currentColor.H;  
            Bitmap newImage = **new** **Bitmap**(100,100);  
              
            // generate eaxh pixel  
            **for** (**double** x = 0; x < 100; x++) {  
                **for** (**double** y = 0; y < 100; y++) {  
                    // uses 1-() in order to flip in y axis  
                    // divide by 100 as the S and V need to between 0 and 1  
                    Color currentColour = SIMPRGB.**HSVtoRGB**(hue,x/100,1-(y/100));  
                    newImage.**SetPixel**((**int**)x,(**int**)y,currentColour);  
                }  
            }  
              
            oldHue = hue;  
            oldImage = newImage;  
            picColourSquare.Image = newImage;  
        }  
          
        **private** void **UpdateColourIndicator**() {  
            System.Drawing.Image editImage = **new** **Bitmap**(oldImage);  
            Graphics GFX = Graphics.**FromImage**(editImage);  
              
            // if V (1-Y) is greater than 0.5 (at lower part of image) make indicator black  
            Color indicatorColor = (\_currentColor.V > 0.5) ? Color.Black : Color.White;  
            GFX.**FillRectangle**(**new** **SolidBrush**(indicatorColor),(**int**)(\_currentColor.S\*100),0,1,100);  
            GFX.**FillRectangle**(**new** **SolidBrush**(indicatorColor),0,(**int**)((1-\_currentColor.V)\*100),100,1);  
              
            picColourSquare.Image = editImage;  
        }  
          
        **private** void **UpdateColourStrip**() {  
            Bitmap newImage = **new** **Bitmap**(20,100);  
            Graphics GFX = Graphics.**FromImage**(newImage);  
              
            **for** (**float** h = 0; h < 100; h++) {  
                // h\*3.6 as H needs to be between 0-360, and h is between 0-100  
                GFX.**FillRectangle**(**new** **SolidBrush**(SIMPRGB.**HSVtoRGB**(h\*3.6,1,1)),0,h,20,1);  
            }  
              
            picColourStrip.Image = newImage;  
        }  
          
        /// <**summary**>  
        /// Color Displays are the two boxes telling you your current and previous colour  
        /// <**/summary**>  
        **private** void **UpdateColourDisplays**() {  
            // startColor  
            Bitmap startImage = **new** **Bitmap**(135,24);  
            Graphics startGFX = Graphics.**FromImage**(startImage);  
            startGFX.**FillRectangle**(**new** **SolidBrush**(\_startColor.**ToColor**()),0,0,135,24);  
            picStartColour.Image = startImage;  
              
            // currentColor  
            Bitmap currentImage = **new** **Bitmap**(135,24);  
            Graphics currentGFX = Graphics.**FromImage**(currentImage);  
            currentGFX.**FillRectangle**(**new** **SolidBrush**(\_currentColor.**ToColor**()),0,0,135,24);  
            picCurrentColour.Image = currentImage;  
        }  
          
        **private** void **UpdateColourInputs**(**bool** updateNums) {  
            Color displayColor = \_currentColor.**ToColor**();  
              
            **if** (updateNums) {  
                // its not always wanted to update the number boxes  
                // if the number box was just changed, then they may twitch back  
                // due to the fact that the R from the number box is converted to HSV notation, then BACK to RGB  
                // so some information is lost - leading to innacuracies  
                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);  
        }  
          
        **private** void **UpdatePreviousColours**() {  
            Bitmap newImage = **new** **Bitmap**(47,290);  
            Graphics GFX = Graphics.**FromImage**(newImage);  
            GFX.**Clear**(SystemColors.ControlDark);  
            **int** currentY = 0;  
            // traverse through the list and fill boxes  
            **foreach** (Color color **in** previousColors) {  
                GFX.**FillRectangle**(**new** **SolidBrush**(color),0,currentY,47,20);  
                currentY+=30;  
            }  
            picRecentColours.Image = newImage;  
        }  
        #**endregion**  
          
        #**region** Value changes  
        void **NumRValueChanged**(object sender, EventArgs e)  
        {  
            **if** (\_updating) {  
                return;  
            }  
              
            \_currentColor = SIMPRGB.**RGBtoHSV**((**double**)numR.Value,(**double**)numG.Value,(**double**)numB.Value);  
              
            // DONT update the number boxes  
            **Update**(**false**);  
        }  
          
        void **NumGValueChanged**(object sender, EventArgs e)  
        {  
            **if** (\_updating) {  
                return;  
            }  
              
            \_currentColor = SIMPRGB.**RGBtoHSV**((**double**)numR.Value,(**double**)numG.Value,(**double**)numB.Value);  
              
            **Update**(**false**);  
        }  
          
        void **NumBValueChanged**(object sender, EventArgs e)  
        {  
            **if** (\_updating) {  
                return;  
            }  
              
            \_currentColor = SIMPRGB.**RGBtoHSV**((**double**)numR.Value,(**double**)numG.Value,(**double**)numB.Value);  
              
            **Update**(**false**);  
        }  
          
        void **PicColourSquareClick**(object sender, EventArgs e)  
        {  
            MouseEventArgs me = (MouseEventArgs)e;  
              
            \_currentColor.S = ((**float**)me.Location.X)/100;  
            \_currentColor.V = 1-((**float**)me.Location.Y)/100;  
              
            **Update**(**true**);  
        }  
          
        void **PicColourStripClick**(object sender, EventArgs e)  
        {  
            MouseEventArgs me = (MouseEventArgs)e;  
              
            // change the hue and update everything  
            **float** hue = ((**float**)me.Location.Y)\*3.6f;  
            \_currentColor.H = hue;  
              
            **Update**(**true**);  
        }  
          
        void **PicColourSquareMouseMove**(object sender, MouseEventArgs e)  
        {  
            **if** (e.Button != MouseButtons.None) {  
                Point newPoint = e.Location;  
                // binds the click location into the box  
                // this is so if you move the mouse outside the box the selector still 'follows' it  
                **if** (e.Location.X < 0) {  
                    newPoint = **new** **Point**(0,e.Location.Y);  
                }  
                **if** (e.Location.X >= picColourSquare.Width) {  
                    newPoint = **new** **Point**(picColourSquare.Width,e.Location.Y);  
                }  
                **if** (e.Location.Y < 0) {  
                    newPoint = **new** **Point**(newPoint.X,0);  
                }  
                **if** (e.Location.Y >= picColourSquare.Height) {  
                    newPoint = **new** **Point**(newPoint.X,picColourSquare.Height);  
                }  
                // just calls the click function - saves repeating code  
                **PicColourSquareClick**(sender,**new** **MouseEventArgs**(e.Button,e.Clicks,newPoint.X,newPoint.Y,e.Delta));  
            }  
        }  
          
        void **PicColourStripMouseMove**(object sender, MouseEventArgs e)  
        {  
            **if** (e.Button != MouseButtons.None) {  
                // same reasoining as PicColorSquareMouseMove  
                Point newPoint = e.Location;  
                **if** (e.Location.X < 0) {  
                    newPoint = **new** **Point**(0,e.Location.Y);  
                }  
                **if** (e.Location.X >= picColourStrip.Width) {  
                    newPoint = **new** **Point**(picColourStrip.Width,e.Location.Y);  
                }  
                **if** (e.Location.Y < 0) {  
                    newPoint = **new** **Point**(newPoint.X,0);  
                }  
                **if** (e.Location.Y >= picColourStrip.Height) {  
                    newPoint = **new** **Point**(newPoint.X,picColourStrip.Height);  
                }  
                **PicColourStripClick**(sender,**new** **MouseEventArgs**(e.Button,e.Clicks,newPoint.X,newPoint.Y,e.Delta));  
            }  
        }  
        #**endregion**  
          
        void **BtnSaveClick**(object sender, EventArgs e)  
        {  
            // calls the callback to be handled by caller  
            **\_updateCallback**(\_currentColor.**ToColor**());  
            **AddPreviousColor**(\_currentColor.**ToColor**());  
              
            **Close**();  
        }  
          
        void **BtnCloseClick**(object sender, EventArgs e)  
        {  
            **Close**();  
        }  
        

        #**region** previouscolors  
        void **AddPreviousColor**(Color color) {  
            previousColors.**Insert**(0,color);  
              
            **if** (previousColors.Count > 10) {  
                previousColors.**RemoveAt**(10); // dequeue color at end and throw it away  
            }  
        }  
          
        void **PicRecentColoursClick**(object sender, EventArgs e)  
        {  
            MouseEventArgs me = (MouseEventArgs)e;  
            **int** colorY = me.Y/30;  
            **if** (me.Y%30 < 20) {  
                **if** (previousColors.Count > colorY) {  
                    \_currentColor = SIMPRGB.**RGBtoHSV**(previousColors[colorY]);  
                    **Update**(**true**);  
                }  
            }  
        }  
        #**endregion**  
    }  
      
    **public** delegate void **ColorCallback**(Color newColour);  
      
    static class SIMPRGB {  
        **public** static Color cachedColor;  
          
        /// <**summary**>  
        /// Converts a HSV colour to its RGB equivalent  
        /// <**/summary**>  
        **public** static Color **HSVtoRGB**(**double** h, **double** s, **double** v) {  
            // see documentation for the details of algorithm used  
            **double** c = s \* v;  
            **double** x = c \* (1-Math.**Abs**(((h/60)%2)-1));  
            **double** m = v - c;  
            **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);  
        }  
          
        // allows you to call RGBtoHSV with either a colour or three numbers  
        **public** static HSVColor **RGBtoHSV**(Color c) {  
            return **RGBtoHSV**((**double**)c.R,(**double**)c.G,(**double**)c.B);  
        }  
          
        **public** static HSVColor **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;  
            h = 0;  
            **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 \* (((R-G)/Δ)+4);  
            }  
            **if** (h<0) {  
                h += 360;  
            }  
              
            **if** (Cmax == 0) {  
                s = 0;  
            } **else** {  
                s = Δ/Cmax;  
            }  
              
            v = Cmax;  
              
            return **new** **HSVColor**(h,s,v);  
        }  
          
        /// <**summary**>  
        /// finds the largest number out of a list  
        /// <**/summary**>  
        **private** static **double** **Max**(**params** **double**[] nums) {  
            **double** max = nums[0];  
              
            **for** (**int** i = 1; i < nums.Length; i++) {  
                **if** (nums[i] > max) {  
                    max = nums[i];  
                }  
            }  
              
            return max;  
        }  
          
        /// <**summary**>  
        /// finds the smallest number out of a list  
        /// <**/summary**>  
        **private** static **double** **Min**(**params** **double**[] nums) {  
            **double** min = nums[0];  
              
            **for** (**int** i = 1; i < nums.Length; i++) {  
                **if** (nums[i] < min) {  
                    min = nums[i];  
                }  
            }  
              
            return min;  
        }  
          
        **public** static Color **SetHue**(**this** Color color, **float** hue) {  
            return **HSVtoRGB**(hue,color.**GetSaturation**(),color.**GetBrightness**());  
        }  
          
        **public** static Color **SetSaturation**(**this** Color color, **float** saturation) {  
            return **HSVtoRGB**(color.**GetHue**(),saturation,color.**GetBrightness**());  
        }  
          
        **public** static Color **SetLightness**(**this** Color color, **float** lightness) {  
            return **HSVtoRGB**(color.**GetHue**(),color.**GetSaturation**(),lightness);  
        }  
    }

    /// <**summary**>  
    /// Structure similar to the Color struct which stores info about an HSV colour  
    /// <**/summary**>  
    **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;  
        }  
          
        **public** Color **ToColor**() {  
            return SIMPRGB.**HSVtoRGB**(H,S,V);  
        }  
          
        **public** override string **ToString**()  
        {  
            return string.**Format**("[HSVColor H={0}, S={1}, V={2}]", H, S, V);  
        }  
  
    }  
}

## SIMPConstants

**namespace** SIMP {  
    **public** static class SimpConstants {  
        **public** static **int** IMAGE\_MAX\_WIDTH = 1000;  
        **public** static **int** IMAGE\_MAX\_HEIGHT = 1000;  
          
        **public** static **int** IMAGE\_DEFAULT\_WIDTH = 50;  
        **public** static **int** IMAGE\_DEFAULT\_HEIGHT = 50;  
          
        **public** static **int** WINDOWS\_TOP\_BAR\_HEIGHT;    // different depending on OS  
        **public** static **int** WINDOWS\_BOTTOM\_BAR\_HEIGHT; // generated at runtime  
        **public** static **int** WINDOWS\_LEFT\_BAR\_WIDTH;  
        **public** static **int** WINDOWS\_RIGHT\_BAR\_WIDTH;  
          
        **public** static **int** WORKSPACE\_LEFT\_PADDING = 24;  
        **public** static **int** WORKSPACE\_TOP\_PADDING = 49;  
        **public** static **int** WORKSPACE\_RIGHT\_PADDING = 220;  
        **public** static **int** WORKSPACE\_BOTTOM\_PADDING = 40;  
          
        **public** static **int** WORKSPACE\_REFRESH\_PERIOD = 10;  
          
        **public** static **int** PROPERTY\_LABEL\_HEIGHT = 15;  
        **public** static **int** PROPERTY\_GAP\_HEIGHT = 0;  
        **public** static **int** PROPERTY\_FIELD\_HEIGHT = 23;  
        **public** static **int** PROPERTY\_SPACER\_HEIGHT = 10;  
        **public** static **int** PROPERTY\_HEIGHT; // the sum of the previous 4 properties  
          
        /// <**summary**>  
        /// Called when SimpConstants is first used, sets up the bar height constants  
        /// <**/summary**>  
        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;  
              
            PROPERTY\_HEIGHT = PROPERTY\_LABEL\_HEIGHT + PROPERTY\_GAP\_HEIGHT + PROPERTY\_FIELD\_HEIGHT + PROPERTY\_SPACER\_HEIGHT;  
        }  
    }  
}

## Workspace

**namespace** SIMP  
{  
    /// <**summary**>  
    /// Description of WorkSpace.  
    /// <**/summary**>  
    **public** partial class Workspace : Form  
    {  
        #**region** vars  
        **public** SIMP.Image image;  
        **public** **int** width, height;  
        **public** Form attachedForm;  
        **public** PictureBox displayBox;  
        **public** **bool** shiftPressed = **false**, controlPressed = **false**;  
        **public** List<ITool> tools;  
        **public** ITool currentTool;  
        **public** string savedName;  
        **public** **bool** savedOnDisk;  
          
        **private** **int** \_savedFormWidth, \_savedFormHeight;  
        **private** **int** \_updateCount = 0;  
        **private** **int** \_leftPadding, \_rightPadding, \_topPadding, \_bottomPadding;  
        **private** Stack<IAction> pastActions;  
        **private** Stack<IAction> futureActions;  
        **private** string \_savedPath;  
          
        **public** **int** leftPadding {  
            get {  
                return \_leftPadding;  
            }  
            set {  
                // makes sure the minimum size of the form is updated if you change the padding  
                \_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**();  
            }  
        }  
        #**endregion**  
        

        #**region** constructor  
        **public** **Workspace**()  
        {  
            **InitializeComponent**();  
              
            **Constructor**(  
                SimpConstants.IMAGE\_DEFAULT\_WIDTH,  
                SimpConstants.IMAGE\_DEFAULT\_HEIGHT  
            );  
        }  
          
        **public** **Workspace**(**int** width, **int** height) {  
            **InitializeComponent**();  
              
            **Constructor**(width,height);  
        }  
          
        **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;  
              
            // Needs to gen controls on layer selector  
            **GenLayersSelector**();  
              
            // Sets the dimensions of the workspace to the dimensions of the form  
            **CalculateDimensions**();  
              
            // Updates the form  
            **UpdateDisplayBox**(**true**,**true**);  
              
            // Adds scroll event  
            **this**.MouseWheel += **new** **MouseEventHandler**(WorkspaceMouseWheel);  
              
            **AddTools**();  
              
            // Adds buttons  
            **int** buttonY = 0;  
            currentTool = tools[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);  
                newButton.BackgroundImage = tool.icon;  
                newButton.BackgroundImageLayout = ImageLayout.Stretch;  
                newButton.BackColor = Color.White;  
                **if** (currentTool == tool) {  
                    newButton.Enabled = **false**;  
                }  
                buttonY+=24;  
                  
                panTools.Controls.**Add**(newButton);  
            }  
            **ShowTool**();  
              
            pastActions = **new** Stack<IAction>();  
            futureActions = **new** Stack<IAction>();  
              
            // As this was just generated, has not yet been saved  
            savedName = "Unnamed";  
            savedOnDisk = **false**;  
        }  
          
        **private** void **AddTools**() {  
            // Defines tools  
            tools = **new** List<ITool>();  
              
            // predefined tools  
            tools.**Add**(**new** SIMP.Tools.**LineTool**("Brush","Simple Brush",**this**,btnPenIcon.BackgroundImage));  
            tools.**Add**(**new** **SinglePixelLineTool**("Pencil","Precise Brush",**this**,btnPencilIcon.BackgroundImage));  
            tools.**Add**(**new** **EraserTool**("Eraser","Rub Stuff Out",**this**,btnEraserIcon.BackgroundImage));  
            tools.**Add**(**new** SIMP.Tools.ShapeTools.**LineTool**("Line Tool","Draw a staight line",**this**,btnLineIcon.BackgroundImage));  
            tools.**Add**(**new** **RectangleTool**("Rectangle","Draws a rectangle",**this**,btnRectangleIcon.BackgroundImage));  
            tools.**Add**(**new** **CircleTool**("Circle","Draws a circle (or rectangle)",**this**,btnCircleIcon.BackgroundImage));  
            tools.**Add**(**new** **DiamondTool**("Diamond","Draws diamonds",**this**,btnDiamondIcon.BackgroundImage));  
            tools.**Add**(**new** **FillTool**("Fill","Fills a shape",**this**,btnFillIcon.BackgroundImage));  
            tools.**Add**(**new** **FXTool**("Special Effects","Apply cool changes to the image",**this**,btnFXIcon.BackgroundImage));  
        }  
        #**endregion**  
        

        #**region** display  
          
        /// <**summary**>  
        /// Resizes, Relocates and (if redraw) updates image in the displayBox  
        /// <**/summary**>  
        /// <**param** **name="redraw"**><**/param**>  
        **public** void **UpdateDisplayBox**(**bool** redraw, **bool** full) {  
            // 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,full);  
            }  
              
            // Updates the progress bars  
            **UpdateBar**(EAxis.X,barHorizontal);  
            **UpdateBar**(EAxis.Y,barVertical);  
              
            \_updateCount++;  
              
            // after x amount of updates, garbage collect  
            // every time the picture box's image is changed, the old one is not disposed of  
            **if** (\_updateCount >= SimpConstants.WORKSPACE\_REFRESH\_PERIOD) {  
                \_updateCount = 0;  
                GC.**Collect**();  
            }  
        }  
          
        /// <**summary**>  
        /// Updates the width and height of the displayBox  
        /// <**/summary**>  
        **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;  
            }  
        }  
          
        /// <**summary**>  
        /// Updates the position of the displayBox  
        /// <**/summary**>  
        **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.displayWidth) / 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.displayHeight) / 2) + topPadding;  
                    break;  
            }  
            displayBox.Location = **new** **Point**(X,Y);  
        }  
        

        /// <**summary**>  
        /// Called when the heartbeat timer ticks  
        /// <**/summary**>  
        **private** void **HeartbeatTick**(object sender, EventArgs e)  
        {  
            **if** (**HasSizeChanged**()) {  
                **CalculateDimensions**();  
                **UpdateDisplayBox**(**true**,**true**);  
            }  
        }  
          
        /// <**summary**>  
        /// Calculates what the width and height of the Workspace should be  
        /// <**/summary**>  
        **private** void **CalculateDimensions**() {  
            width = DisplayRectangle.Width - (leftPadding + rightPadding);  
            height = DisplayRectangle.Height - (topPadding + bottomPadding);  
        }  
          
        **private** void **WorkspaceResizeEnd**(object sender, EventArgs e)  
        {  
            //UpdateDisplayBox(true);  
        }  
          
        /// <**summary**>  
        /// If the form size is different to the last recorded ones  
        /// <**/summary**>  
        /// <**returns**><**/returns**>  
        **private** **bool** **HasSizeChanged**() {  
            // if width has changed  
            **if** (\_savedFormWidth != DisplayRectangle.Width) {  
                \_savedFormWidth = DisplayRectangle.Width;  
                return **true**;  
            }  
              
            // if height has changed  
            **if** (\_savedFormHeight != DisplayRectangle.Height) {  
                \_savedFormHeight = DisplayRectangle.Height;  
                return **true**;  
            }  
              
            return **false**;  
        }  
        

        /// <**summary**>  
        /// Checks whether the image is smaller or larger than the form in this Axis  
        /// <**/summary**>  
        /// <**param** **name="axis"**>Axis to check in<**/param**>  
        /// <**returns**><**/returns**>  
        **private** EAxisMode **CheckImageSize**(EAxis axis) {  
            **int** axisSize;  
            **int** padding;  
            **switch** (axis) {  
                **case** EAxis.X:  
                    // determines the max size of the relevent axis and the amount of padding to deduct  
                    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");  
            }  
              
            // gets the display size of the relevent axis (taking zoom into account)  
            **int** imageAxisSize = image.**ToDisplayRectangle**().**GetDisplaySize**(axis);  
              
            // if the size of form is bigger than the image's size plus padding  
            **if** (axisSize >= (imageAxisSize + padding)) {  
                // then the image is smaller than form  
                return EAxisMode.ImageTooSmall;  
            } **else** {  
                return EAxisMode.ImageTooLarge;  
            }  
        }  
          
        **private** void **SetMinimumSize**() {  
            **this**.MinimumSize = **new** **Size**(\_leftPadding + \_rightPadding + SimpConstants.WINDOWS\_RIGHT\_BAR\_WIDTH + SimpConstants.WINDOWS\_LEFT\_BAR\_WIDTH + 100,  
                                        \_topPadding + \_bottomPadding + SimpConstants.WINDOWS\_TOP\_BAR\_HEIGHT + SimpConstants.WINDOWS\_BOTTOM\_BAR\_HEIGHT + 100);  
        }  
          
        #**endregion**  
        

        #**region** bars  
          
        /// <**summary**>  
        /// When the Value of the Zoom bar changes  
        /// <**/summary**>  
        void **BarZoomScroll**(object sender, EventArgs e)  
        {  
            image.zoomSettings.zoom = barZoom.Value;  
              
            // everything changes when zoom changes so just redisplay everything  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        **private** void **UpdateBar**(EAxis axis, ScrollBar bar) {  
            // checks whether this bar should be visible or not  
            // it might not be if there is nothing to scroll  
            **SetBarVisiblity**(axis,bar);  
              
            image.**SetBarValues**(barHorizontal,barVertical);  
              
        }  
          
        **private** void **SetBarVisiblity**(EAxis axis, ScrollBar bar) {  
            **switch** (**CheckImageSize**(axis)) {  
                **case** EAxisMode.ImageTooLarge:  
                    bar.Enabled = **true**;  
                    break;  
                **case** EAxisMode.ImageTooSmall:  
                    bar.Enabled = **false**;  
                    break;  
            }  
        }  
          
        void **BarHorizontalValueChanged**(object sender, EventArgs e)  
        {  
            image.**CentreFromBar**(barHorizontal.Value,EAxis.X);  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        void **BarVerticalValueChanged**(object sender, EventArgs e)  
        {  
            image.**CentreFromBar**(barVertical.Value,EAxis.Y);  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        #**endregion**  
        

        #**region** mouse  
          
        void **DisplayBoxMouseDown**(object sender, MouseEventArgs e)  
        {  
            // calls the handler of the current tool - the tool handles it from there  
            DisplayPoint clickLocation = **new** **DisplayPoint**(e.Location.X,e.Location.Y);  
            currentTool.**HandleMouseDown**(image.**DisplayPointToFilePoint**(clickLocation),e.Button);  
        }  
          
        void **DisplayBoxMouseUp**(object sender, MouseEventArgs e)  
        {  
            DisplayPoint clickLocation = **new** **DisplayPoint**(e.Location.X,e.Location.Y);  
            currentTool.**HandleMouseUp**(image.**DisplayPointToFilePoint**(clickLocation),e.Button);  
        }  
          
        // .net doesnt store the old location - guess I will instead  
        DisplayPoint oldLocation = **new** **DisplayPoint**(0,0);  
        void **DisplayBoxMouseMove**(object sender, MouseEventArgs e)  
        {  
            DisplayPoint newLocation = **new** **DisplayPoint**(e.Location.X,e.Location.Y);  
            currentTool.**HandleMouseMove**(image.**DisplayPointToFilePoint**(oldLocation),image.**DisplayPointToFilePoint**(newLocation));  
              
            oldLocation = newLocation;  
        }  
          
        void **DisplayBoxClick**(object sender, EventArgs e)  
        {  
            MouseEventArgs me = (MouseEventArgs)e;  
            DisplayPoint clickLocation = **new** **DisplayPoint**(me.Location.X,me.Location.Y);  
            currentTool.**HandleMouseClick**(image.**DisplayPointToFilePoint**(clickLocation),me.Button);  
        }  
  
        void **WorkspaceMouseWheel**(object sender, MouseEventArgs e)  
        {  
            // 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) {  
                    // makes sure not scrolling TOO far  
                    **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--;  
                    }  
                }  
            }  
        }  
          
        /// <**summary**>  
        /// Not for any direct function, but records when keys like shift or alt are pressed or unpressed  
        /// <**/summary**>  
        void **WorkspaceKeyDown**(object sender, KeyEventArgs e)  
        {  
            **if** (e.Shift) {  
                shiftPressed = **true**;  
            }  
            **if** (e.Control) {  
                controlPressed = **true**;  
                  
                // though does make ctrl+Z undo and ctrl+Y redo  
                **if** (e.KeyCode == Keys.Z && btnUndo.Enabled) {  
                    **BtnUndoClick**(**null**,**null**);  
                }  
                **if** (e.KeyCode == Keys.Y && btnRedo.Enabled) {  
                    **BtnRedoClick**(**null**,**null**);  
                }  
            }  
        }  
          
        void **WorkspaceKeyUp**(object sender, KeyEventArgs e)  
        {  
            **if** (e.KeyCode == Keys.ShiftKey) {  
                shiftPressed = **false**;  
            }  
            **if** (e.KeyCode == Keys.ControlKey) {  
                controlPressed = **false**;  
            }  
        }  
          
        #**endregion**  
          
        #**region** tools  
          
        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;  
                }  
            }  
              
            // if no tool is selected - hide the tool menu  
            **if** (currentTool != ITool.BlankTool) {  
                **ShowTool**();  
            } **else** {  
                **HideTool**();  
            }  
              
        }  
        

**bool** toolsOpen = **false**;  
        **public** void **ShowTool**() {  
            **if** (currentTool.name.**Equals**("Eyedropper")) { // never attempt to display eyedropper  
                return;  
            }  
              
            leftPadding = SimpConstants.WORKSPACE\_LEFT\_PADDING + 200;  
            panToolProperties.Visible = **true**;  
            panToolProperties.Controls.**Clear**();  
              
            Label lblToolName = **new** **Label**();  
            lblToolName.Location = **new** **Point**(0, 0);  
            lblToolName.Size = **new** **Size**(191, 23);  
            lblToolName.Text = currentTool.name;  
            lblToolName.Font = **new** **Font**(lblToolName.Font,FontStyle.Bold);  
            panToolProperties.Controls.**Add**(lblToolName);  
              
            **int** currentY = lblToolName.Height;  
            **int** width = panToolProperties.Width;  
            **foreach** (IProperty property **in** currentTool.properties) {  
                // doesn't show invisbles  
                **if** (property.propertyType == PropertyType.Hidden) {  
                    continue;  
                }  
                  
                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;  
                  
                // generates controls to put inside picture box  
                **if** (property **is** ColorProperty) {  
                    ColorProperty colorProperty = (ColorProperty)property;  
                    PictureBox newPicture = **new** **PictureBox**();  
                    newPicture.Size = **new** **Size**(width - 31,SimpConstants.PROPERTY\_FIELD\_HEIGHT);  
                    newPicture.Location = **new** **Point**(0,currentY);  
                    newPicture.BackColor = (Color)property.**value**;  
                    newPicture.Click += colorProperty.onInteract;  
                    newPicture.Cursor = Cursors.Hand;  
                    panToolProperties.Controls.**Add**(newPicture);  
                    Button btnEyedropper = **new** **Button**();  
                    btnEyedropper.Size = **new** **Size**(23,23);  
                    btnEyedropper.Location = **new** **Point**(width - 27,currentY);  
                    btnEyedropper.Click += **new** **EventHandler**(btnEyedropper\_Click);  
                    btnEyedropper.Tag = colorProperty;  
                    btnEyedropper.BackColor = (Color)colorProperty.**value**;  
                    btnEyedropper.BackgroundImage = btnEyedropperIcon.BackgroundImage;  
                    btnEyedropper.BackgroundImageLayout = ImageLayout.Stretch;  
                    panToolProperties.Controls.**Add**(btnEyedropper);  
                } **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 = (**int**)numericalProperty.**value**;  
                    newNum.ValueChanged += numericalProperty.onInteract;  
                    panToolProperties.Controls.**Add**(newNum);  
                } **else** **if** (property **is** DecimalProperty) {  
                    NumericUpDown newNum = **new** **NumericUpDown**();  
                    DecimalProperty DecimalProperty = (DecimalProperty)property;  
                    newNum.Size = **new** **Size**(width,SimpConstants.PROPERTY\_FIELD\_HEIGHT);  
                    newNum.Location = **new** **Point**(0,currentY);  
                    newNum.Value = DecimalProperty.min;  
                    newNum.Minimum = DecimalProperty.min;  
                    newNum.Maximum = DecimalProperty.max;  
                    newNum.Value = (**decimal**)DecimalProperty.**value**;  
                    newNum.ValueChanged += DecimalProperty.onInteract;  
                    panToolProperties.Controls.**Add**(newNum);  
                } **else** **if** (property **is** ComboProperty) {  
                    ComboBox newCombo = **new** **ComboBox**();  
                    ComboProperty comboProperty = (ComboProperty)property;  
                    newCombo.Size = **new** **Size**(width,SimpConstants.PROPERTY\_FIELD\_HEIGHT);  
                    newCombo.Location = **new** **Point**(0,currentY);  
                    **foreach** (string option **in** comboProperty.options) {  
                        newCombo.Items.**Add**(option);  
                    }  
                    newCombo.SelectedItem = comboProperty.**value**;  
                    newCombo.SelectedIndexChanged += comboProperty.onInteract;  
                    newCombo.DropDownStyle = ComboBoxStyle.DropDownList;  
                    panToolProperties.Controls.**Add**(newCombo);  
                }  
                currentY += SimpConstants.PROPERTY\_FIELD\_HEIGHT;  
                  
                currentY += SimpConstants.PROPERTY\_SPACER\_HEIGHT;  
            }  
              
            **CalculateDimensions**();  
              
            **if** (!toolsOpen) {  
                toolsOpen = **true**;  
                **UpdateDisplayBox**(**true**,**true**);  
            }  
        }  
  
        void **btnEyedropper\_Click**(object sender, EventArgs e)  
        {  
            **if** (currentTool **is** EyedropperTool) {  
                currentTool = ((EyedropperTool)currentTool).returnToTool;  
            } **else** {  
                EyedropperTool newTool = **new** **EyedropperTool**("Eyedropper","",**this**,(ColorProperty)((Button)sender).Tag,currentTool);  
                currentTool = newTool;      
            }  
        }  
          
        **private** void **HideTool**() {  
            leftPadding = SimpConstants.WORKSPACE\_LEFT\_PADDING;  
            panToolProperties.Visible = **false**;  
            **CalculateDimensions**();  
            **UpdateDisplayBox**(**true**,**true**);  
            toolsOpen = **false**;  
        }  
          
        #**endregion**  
          
        #**region** action  
        /// <**summary**>  
        /// Perform an action and record it  
        /// <**/summary**>  
        **public** void **PerformAction**(IAction action) {  
            **PerformActionSilent**(action);  
            **RecordAction**(action);  
        }  
          
        **public** void **PerformActionSilent**(IAction action) {  
            // perform an action on this workspace  
            action.**Do**(**this**);  
        }  
          
        **public** void **RecordAction**(IAction action) {  
            pastActions.**Push**(action);  
            futureActions.**Clear**();  
            // cannot redo now - just did a new action  
            btnRedo.Enabled = **false**;  
            // however can definitely undo  
            btnUndo.Enabled = **true**;  
        }  
        

        void **BtnUndoClick**(object sender, EventArgs e)  
        {  
            IAction lastAction = pastActions.**Pop**();  
              
            lastAction.**Undo**(**this**);  
              
            futureActions.**Push**(lastAction);  
            btnRedo.Enabled = **true**;  
              
            **if** (pastActions.Count == 0) {  
                btnUndo.Enabled = **false**;  
            }  
              
            // updates the layers display if needed  
            **GenLayersSelector**();  
        }  
          
        void **BtnRedoClick**(object sender, EventArgs e)  
        {  
            IAction futureAction = futureActions.**Pop**();  
              
            futureAction.**Do**(**this**);  
              
            pastActions.**Push**(futureAction);  
            btnUndo.Enabled = **true**;  
              
            **if** (futureActions.Count == 0) {  
                btnRedo.Enabled = **false**;  
            }  
              
            **GenLayersSelector**();  
        }  
        #**endregion**  
          
        #**region** off bounds image  
        // handles stuff such that if you start a stroke by clicking on the workspace, should still be handled by image  
        void **WorkspaceMouseDown**(object sender, MouseEventArgs e)  
        {  
            currentTool.**HandleMouseDown**(**null**,e.Button);  
        }  
          
        void **WorkspaceMouseUp**(object sender, MouseEventArgs e)  
        {  
            currentTool.**HandleMouseUp**(**null**,e.Button);  
        }  
        

        void **WorkspaceMouseMove**(object sender, MouseEventArgs e)  
        {  
            Rectangle displayRectangle = **new** **Rectangle**(displayBox.Location,displayBox.Size);  
              
            **if** (displayRectangle.**Contains**(e.Location)) {  
                Point clickLocation = e.Location;  
                clickLocation.**Offset**(displayBox.Location.X \* -1,  
                                     displayBox.Location.Y \* -1);  
                oldLocation = **new** **DisplayPoint**(clickLocation.X,clickLocation.Y);  
                **DisplayBoxMouseMove**(displayBox,**new** **MouseEventArgs**(  
                    e.Button,  
                    e.Clicks,  
                    clickLocation.X,  
                    clickLocation.Y,  
                    e.Delta)  
                );  
            }  
        }  
        #**endregion**  
          
        #**region** layer display  
        **private** void **GenLayersSelector**() {  
            **int** y=5;  
            panLayerSelector.Controls.**Clear**();  
            panLayerSelector.Controls.**Add**(panLayerButtons);  
            previewImages = **new** Dictionary<Layer, PictureBox>();  
              
            **foreach** (Layer layer **in** image.layers) {  
                Panel layerPanel = **new** **Panel**();  
                layerPanel.Location = **new** **Point**(5,y);  
                layerPanel.Size = **new** **Size**(panLayers.Width-10,50);  
                // makes currently selected layer dark  
                **if** (layer == image.currentLayer) {  
                    layerPanel.BackColor = SystemColors.ControlDark;  
                } **else** {  
                    layerPanel.BackColor = SystemColors.Control;  
                }  
                layerPanel.Tag = layer;  
                layerPanel.Click += **new** **EventHandler**(PanLayerClick);  
                  
                // add controls to the new layer  
                // picture box displaying preview  
                PictureBox layerPreview = **new** **PictureBox**();  
                layerPreview.Location = **new** **Point**(5,5);  
                layerPreview.Size = **new** **Size**(40,40);  
                layerPreview.BackColor = Color.White;  
                layerPanel.Controls.**Add**(layerPreview);  
                previewImages[layer] = layerPreview;  
                  
                // check box showing visible  
                CheckBox visibleBox = **new** **CheckBox**();  
                visibleBox.Location = **new** **Point**(50,5);  
                visibleBox.Size = **new** **Size**(100,20);  
                visibleBox.Text = "Visible?";  
                visibleBox.Checked = layer.visible;  
                visibleBox.Tag = layer;  
                visibleBox.CheckedChanged += **new** **EventHandler**(ChkLayerChecked);  
                layerPanel.Controls.**Add**(visibleBox);  
                  
                panLayerSelector.Controls.**Add**(layerPanel);  
                y+=55;  
            }  
              
            **UpdateLayersSelector**(**true**);  
            **UpdateLayerButtons**();  
        }  
          
        Dictionary<Layer,PictureBox> previewImages;  
        **public** void **UpdateLayersSelector**(**bool** full) {  
            // if requests to update EVERY layer  
            **if** (full) {  
                **foreach** (Layer layer **in** image.layers) {  
                    **UpdateLayer**(layer);  
                }  
            } **else** { // or just the currently displayed one  
                // this is to save performance as most changes just edit one layer at a time  
                **UpdateLayer**(image.currentLayer);  
            }  
        }  
          
        **private** void **UpdateLayerButtons**() {  
            btnRemoveLayer.Enabled = image.layers.Count > 1;  
              
            // disables move layer up button if the current layer is at start of list  
            btnLayerUp.Enabled = image.layers.**IndexOf**(image.currentLayer) != 0;  
              
            // disables move layer down button if the current layer is at end of list  
            btnLayerDown.Enabled = image.layers.**IndexOf**(image.currentLayer) != (image.layers.Count - 1);  
        }  
          
        **private** void **UpdateLayer**(Layer layer) {  
            System.Drawing.Image newImage = **new** System.Drawing.**Bitmap**(40,40);  
            Graphics GFX = Graphics.**FromImage**(newImage);  
            // checks 40 pixels along the image  
            // if actual image is 400 pixels long, it checks every 10  
            // an inexpensive form of compression  
            **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;      
        }  
          
        void **PanLayerClick**(object sender, EventArgs e)  
        {  
            // when a panel containing a layer is pressed  
            // the tag of each panel is a reference to its layer  
            image.currentLayer = (Layer)(((Panel)sender).Tag);  
            **GenLayersSelector**();  
        }  
          
        void **ChkLayerChecked**(object sender, EventArgs e) {  
            ((Layer)(((CheckBox)sender).Tag)).visible = ((CheckBox)sender).Checked;  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        void **BtnNewLayerClick**(object sender, EventArgs e)  
        {  
            Layer newLayer = **new** **Layer**(image.fileWidth,image.fileHeight);  
            image.layers.**Insert**(image.layers.**IndexOf**(image.currentLayer),newLayer);  
            image.currentLayer = newLayer;  
              
            **GenLayersSelector**();  
        }  
          
        void **BtnRemoveLayerClick**(object sender, EventArgs e)  
        {  
            image.layers.**Remove**(image.currentLayer);  
            // changes current layer to just the start of the list  
            image.currentLayer = image.layers[0];  
              
            **GenLayersSelector**();  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        void **BtnLayerUpClick**(object sender, EventArgs e)  
        {  
            **int** layerPos = image.layers.**IndexOf**(image.currentLayer);  
            // remove this layer from the list and insert it one place below  
            image.layers.**Remove**(image.currentLayer);  
            image.layers.**Insert**(layerPos-1,image.currentLayer);  
              
            **GenLayersSelector**();  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
          
        void **BtnLayerDownClick**(object sender, EventArgs e)  
        {  
            **int** layerPos = image.layers.**IndexOf**(image.currentLayer);  
            //opposite of layer up  
            image.layers.**Remove**(image.currentLayer);  
            image.layers.**Insert**(layerPos+1,image.currentLayer);  
              
            **GenLayersSelector**();  
            **UpdateDisplayBox**(**true**,**true**);  
        }  
        #**endregion**  
          
        #**region** saving / loading controls  
        void **BtnOpenClick**(object sender, EventArgs e)  
        {  
            DialogResult result = diaOpen.**ShowDialog**();  
            **if** (result == DialogResult.OK) {  
                **using** (FileStream stream = **new** **FileStream**(diaOpen.FileName,FileMode.Open)) {  
                    **OpenFile**(stream);  
                }  
            }  
        }  
          
        void **BtnSaveClick**(object sender, EventArgs e)  
        {  
            // save button for if you have already saved before  
            // \_savedPath contains the path you entered when you pressed 'save as'  
            **using** (FileStream stream = **new** **FileStream**(\_savedPath,FileMode.OpenOrCreate)) {  
                **SaveFile**(stream);  
                  
                MessageBox.**Show**("Saved!","Success!",MessageBoxButtons.OK,MessageBoxIcon.Information);  
            }  
        }  
          
        void **BtnSaveAsClick**(object sender, EventArgs e)  
        {  
            DialogResult result = diaSave.**ShowDialog**();  
            **if** (result == DialogResult.OK) {  
                **using** (FileStream stream = **new** **FileStream**(diaSave.FileName,FileMode.OpenOrCreate)) {  
                    **SaveFile**(stream);  
                      
                    **MarkSavedTo**(Path.**GetFileName**(diaSave.FileName),diaSave.FileName);  
                    MessageBox.**Show**("Saved!","Success!",MessageBoxButtons.OK,MessageBoxIcon.Information);  
                }  
            }  
        }  
          
        void **SaveFile**(FileStream stream) {  
            //SIMP check digits  
            //format v2 thus SIM2  
            stream.**WriteByte**((**byte**)'S');  
            stream.**WriteByte**((**byte**)'I');  
            stream.**WriteByte**((**byte**)'M');  
            stream.**WriteByte**((**byte**)'2');  
                  
            //header  
            stream.**WriteByte**((**byte**)(image.fileWidth / 256)); // saves width and height into two bytes  
            stream.**WriteByte**((**byte**)(image.fileWidth % 256)); // this is so you can save images larger than 255 in size  
            stream.**WriteByte**((**byte**)(image.fileHeight / 256));  
            stream.**WriteByte**((**byte**)(image.fileHeight % 256));  
            stream.**WriteByte**((**byte**)image.layers.Count); // not much point doing the same for layers - 255 layers is impossible  
              
            Color currentColor;  
            // now the header stuff is present, write every colour sequentially  
            **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);  
                        stream.**WriteByte**(currentColor.A);  
                    }  
                }  
            }  
        }  
          
        **public** static Workspace **OpenFile**(FileStream stream) {  
            string checkString = "";  
            **for** (**int** i = 0; i < 4; i++) {  
                checkString += (**char**)**SafeRead**(stream);  
            }  
            **int** width,height,layerCount,R,G,B,A = 0;  
            Workspace newWorkspace;  
            Image image;  
            List<Layer> newLayers;  
              
            **switch** (checkString) {  
                **case** "SIMP": //simp format v1 loading  
                    width = (**SafeRead**(stream) \* 256) + **SafeRead**(stream);  
                    height = (**SafeRead**(stream) \* 256) + **SafeRead**(stream);  
                    layerCount = **SafeRead**(stream);  
                      
                    newWorkspace = **new** **Workspace**(width,height);  
                      
                    image = **new** **Image**(width,height,newWorkspace);  
                    newWorkspace.image = image;  
                    newLayers = **new** List<Layer>();  
                      
                    **for** (**int** i = 0; i < layerCount; i++) {  
                        Layer newLayer = **new** **Layer**(width,height);  
                        **for** (**int** x = 0; x < width; x++) {  
                            **for** (**int** y = 0; y < height; y++) {  
                                // reads all the colours off  
                                R = **SafeRead**(stream);  
                                G = **SafeRead**(stream);  
                                B = **SafeRead**(stream);  
                                Color newColor = Color.**FromArgb**(R,G,B);  
                                //patches the transparency issue  
                                **if** (newColor.R == 255 &&  
                                    newColor.G == 255 &&  
                                    newColor.B == 255 &&  
                                    newColor.A == 255  
                                   ) {  
                                    newColor = Color.Transparent;  
                                }  
                                newLayer.pixels[x,y] = **new** **SolidBrush**(newColor);  
                            }  
                        }  
                        newLayers.**Add**(newLayer);  
                    }  
                      
                    image.layers = newLayers;  
                    newWorkspace.image = image;  
                    newWorkspace.image.currentLayer = newWorkspace.image.layers[0];  
                      
                    newWorkspace.**Show**();  
                    newWorkspace.**GenLayersSelector**();  
                      
                    return newWorkspace;  
                      
                **case** "SIM2": //simp format v2 loading - transparency is saved along with RGB  
                    width = (stream.**ReadByte**() \* 256) + stream.**ReadByte**();  
                    height = (stream.**ReadByte**() \* 256) + stream.**ReadByte**();  
                    layerCount = stream.**ReadByte**();  
                      
                    newWorkspace = **new** **Workspace**(width,height);  
                      
                    image = **new** **Image**(width,height,newWorkspace);  
                    newWorkspace.image = image;  
                    newLayers = **new** List<Layer>();  
                      
                    **for** (**int** i = 0; i < layerCount; i++) {  
                        Layer newLayer = **new** **Layer**(width,height);  
                        **for** (**int** x = 0; x < width; x++) {  
                            **for** (**int** y = 0; y < height; y++) {  
                                R = **SafeRead**(stream);  
                                G = **SafeRead**(stream);  
                                B = **SafeRead**(stream);  
                                A = **SafeRead**(stream);  
                                newLayer.pixels[x,y] = **new** **SolidBrush**(Color.**FromArgb**(A,R,G,B));  
                            }  
                        }  
                        newLayers.**Add**(newLayer);  
                    }  
                      
                    image.layers = newLayers;  
                    newWorkspace.image = image;  
                    newWorkspace.image.currentLayer = newWorkspace.image.layers[0];  
                      
                    newWorkspace.**Show**();  
                    newWorkspace.**GenLayersSelector**();  
                      
                    return newWorkspace;  
                      
                **default**: // if no valid SIMP header found  
                    MessageBox.**Show**("File cannot be loaded. \n\nSIMP data could not be found within this file.","Loading Error",MessageBoxButtons.OK,MessageBoxIcon.Error);  
                    return **null**;  
            }  
        }  
          
        void **BtnImportClick**(object sender, EventArgs e)  
        {  
            DialogResult result = diaImport.**ShowDialog**();  
            **if** (result == DialogResult.OK) {  
                Bitmap file = **new** **Bitmap**(diaImport.FileName);  
                Layer newLayer = **new** **Layer**(image.fileWidth,image.fileHeight);  
                  
                **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));  
                    }  
                }  
                  
                image.layers.**Insert**(0,newLayer);  
                image.currentLayer = newLayer;  
                  
                **this**.**GenLayersSelector**();  
                **this**.**UpdateDisplayBox**(**true**,**true**);  
            }  
        }  
          
        void **BtnExportClick**(object sender, EventArgs e)  
        {  
            DialogResult result = diaExport.**ShowDialog**();  
            **int** tempZoom = image.zoomSettings.zoom;  
            // zoom is temporarily changed to 1 so image is exported not zoomed in  
            image.zoomSettings.zoom = 1;  
            **if** (result == DialogResult.OK) {  
                // requests a display image from the image class  
                System.Drawing.Image saveImage = image.**GetDisplayImage**(image.fileWidth,image.fileHeight,**true**);  
                // in-built saving function - System.Drawing.Image#Save  
                saveImage.**Save**(diaExport.FileName);  
                image.zoomSettings.zoom = tempZoom;  
                MessageBox.**Show**("Exported!","Success!",MessageBoxButtons.OK,MessageBoxIcon.Information);  
            }  
        }  
          
        **private** static **int** **SafeRead**(FileStream stream) {  
            **int** result = stream.**ReadByte**();  
            **if** (result == -1) {  
                // throws a catchable error if -1 is returned  
                **throw** **new** **EndOfStreamException**();  
            } **else** {  
                return result;  
            }  
        }  
          
        // when 'save as' executes successfully  
        // enabled the needed buttons and changes form name  
        **public** void **MarkSavedTo**(string name, string path) {  
            savedName = name;  
            \_savedPath = path;  
            savedOnDisk = **true**;  
              
            **this**.Text = savedName;  
            **this**.btnSave.Enabled = **true**;  
        }  
        #**endregion**  
    }  
}

## ZoomSettings

**namespace** SIMP  
{  
    /// <**summary**>  
    /// settings about the zoom of the image  
    /// <**/summary**>  
    **public** **struct** ZoomSettings  
    {  
        **public** **int** zoom;  
        **public** FilePoint fileCentreLocation;  
        **public** DisplayPoint displayCentreLocation;  
          
        **public** **ZoomSettings**(FilePoint centreLocation) {  
            **this**.zoom = 1;  
            **this**.fileCentreLocation = centreLocation;  
            **this**.displayCentreLocation = **null**;  
        }  
          
        /// <**summary**>  
        /// Generate the displayCentreLocation depending on the width and height of the image  
        /// <**/summary**>  
        **public** void **CalcDisplayCentreLocation**(**int** width, **int** height) {  
            **int** newX = width/2;  
            **int** newY = height/2;  
            displayCentreLocation = **new** **DisplayPoint**(newX,newY);  
        }  
    }  
}