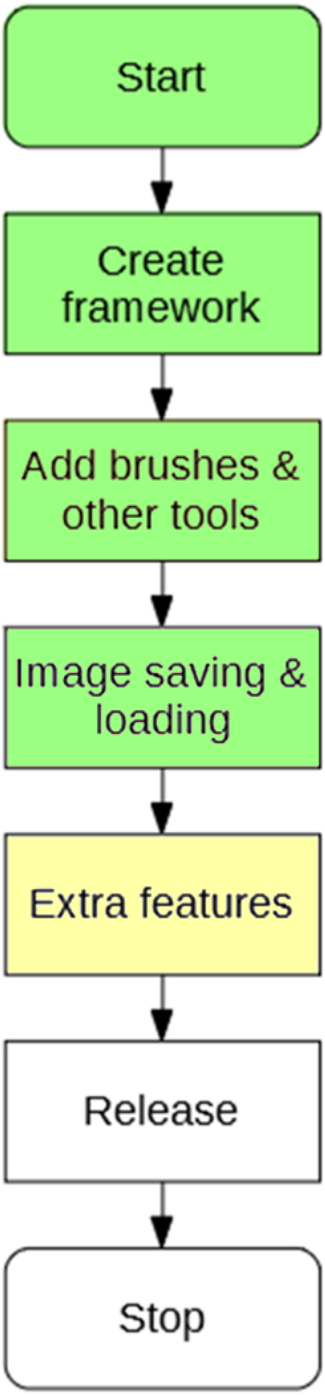
Section 5 – Extra Features

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

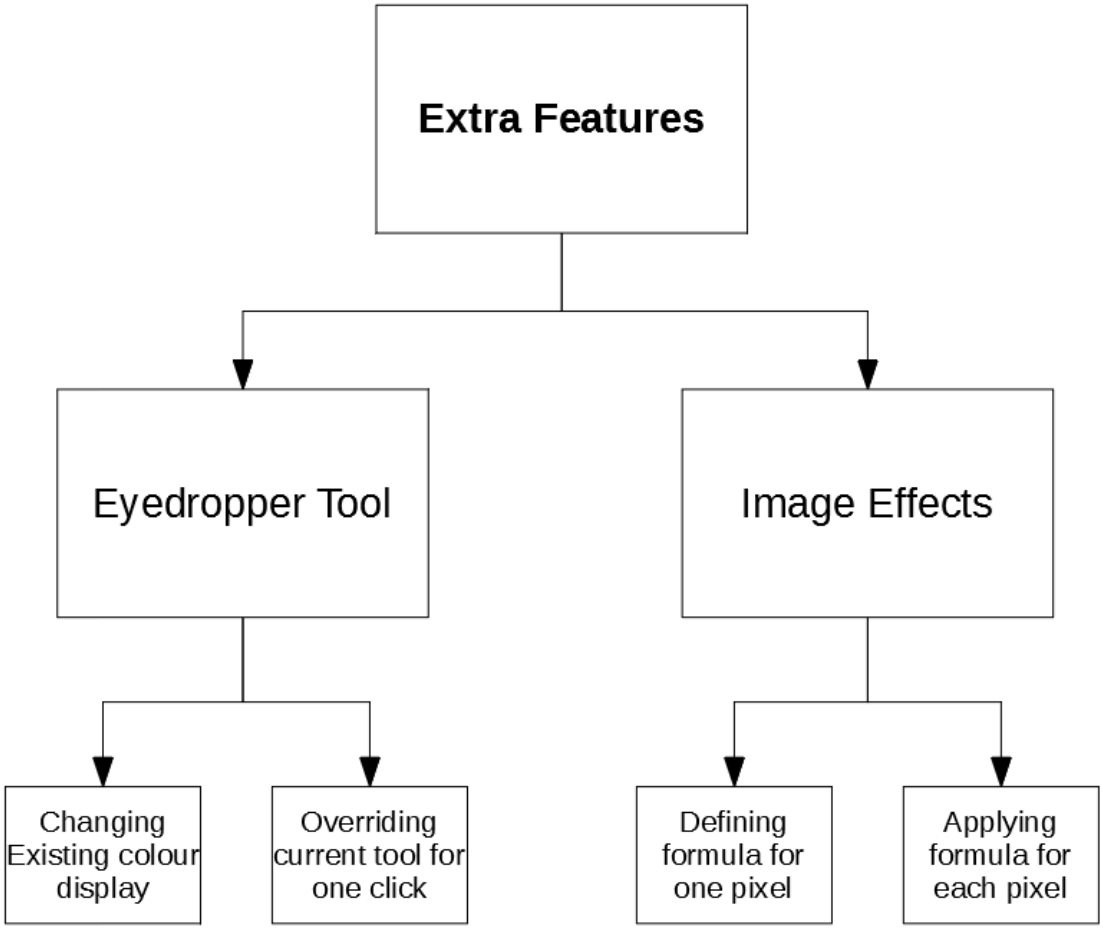
To be done this section

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

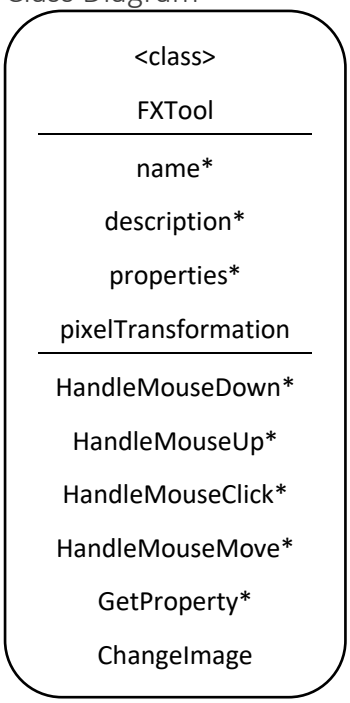
Method	Params	Return type	Justification
HandleMouseDown	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseUp	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseClick	FilePoint clickLocation, MouseButton button	None	Inherited from ITool
HandleMouseMove	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



Properties

Property	Datatype	Justification
name	String	Inherited from ITool
description	String	Inherited from ITool
properties	List of ToolProperties	Inherited from ITool
pixelTransformation	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

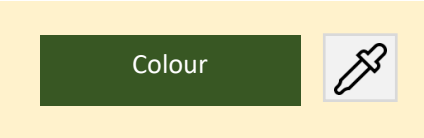
Property	Datatype	Justification
name*	String	Inherited from IToolProperty
value	String	Stores the value of the property
options	Array of String	Stores the possible options to choose from.

5.1.4 Algorithm Design

Algorithm 5.1 Changing existing colour display

The current display for a colour looks like:

This should be changed to look like:



Where the eyedropper button initiates the tool.

Algorithm 5.2 Overriding current tool for one click

When created, the ReturnToTool property of the EyeDropper tool can be set to the tool that was selected. Then, upon clicking:

```
HandleMouseClick(clickLocation) {
      colour = image.GetColour(clickLocation)
      editingProperty.value = colour
      workspace.currentTool = returnToTool
      workspace.displayTool()
}
```

Redisplays the tool as the colour has changed

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
}
```

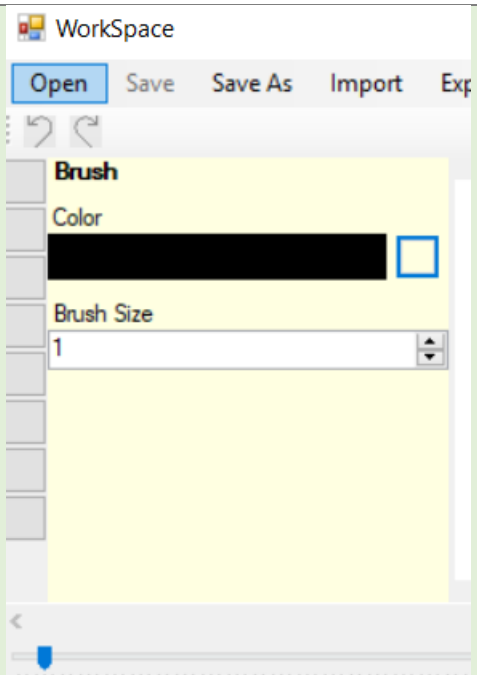
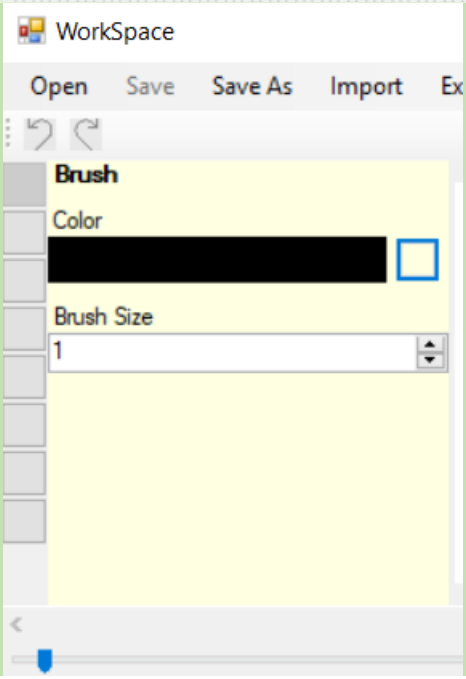
Algorithm 5.3 & Algorithm 5.4 Unit Test

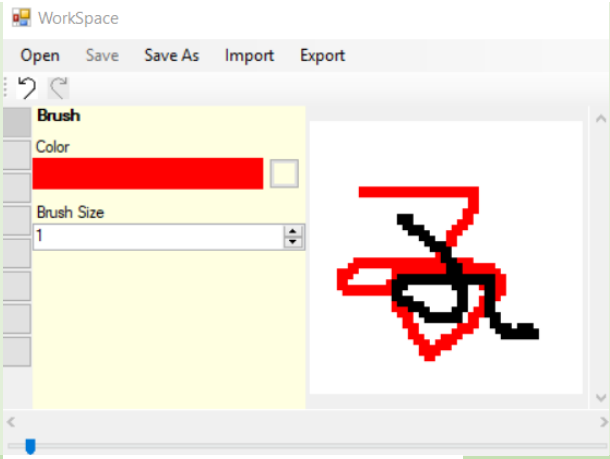
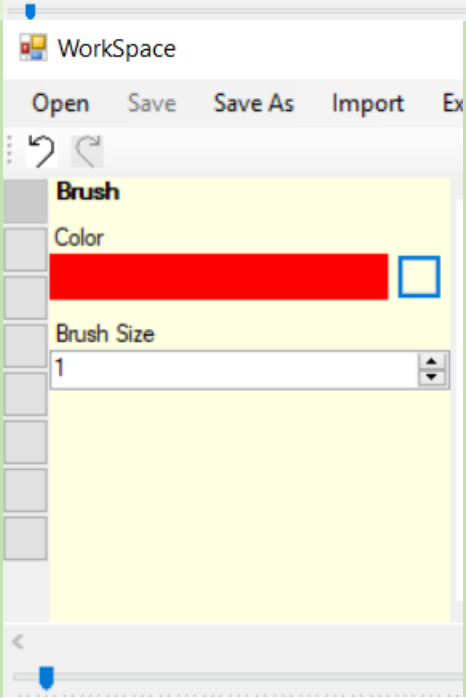
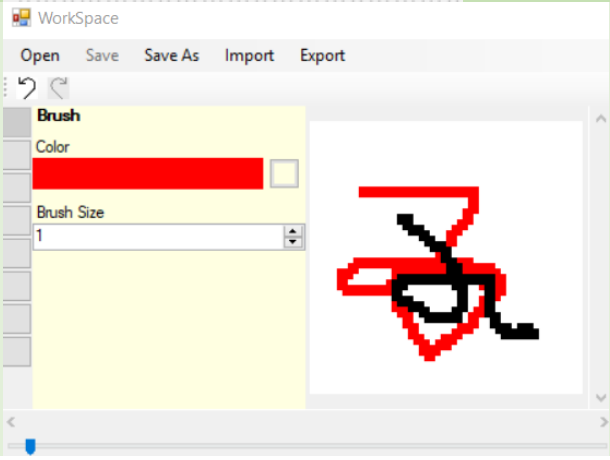
Test	*ID*	*Expected Result*	*Comment*
Apply grayscale transformation	1	The image becomes shades of grey	Tests that the grayscale transformation functions correctly
Apply black & white transformation	2	The image becomes black and white	Tests that black and white transformation functions correctly
Apply 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

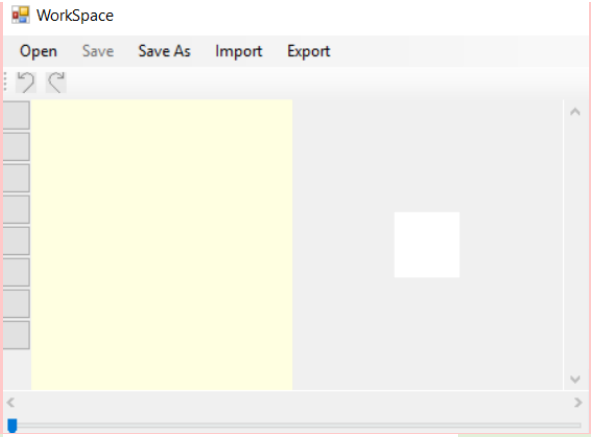
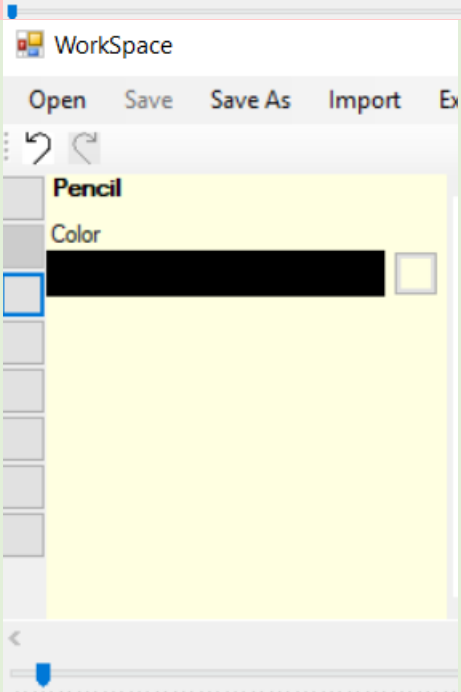
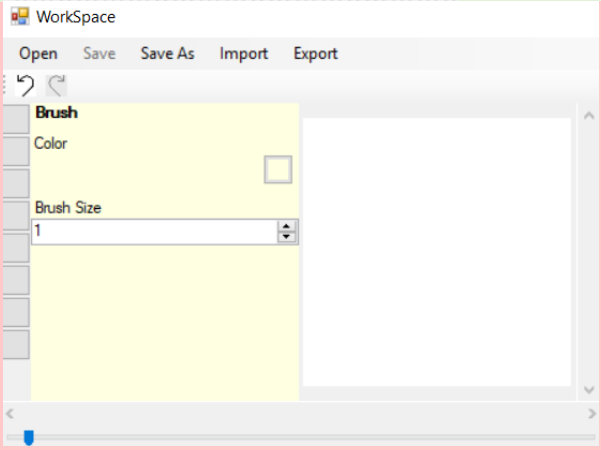
5.2 Development

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

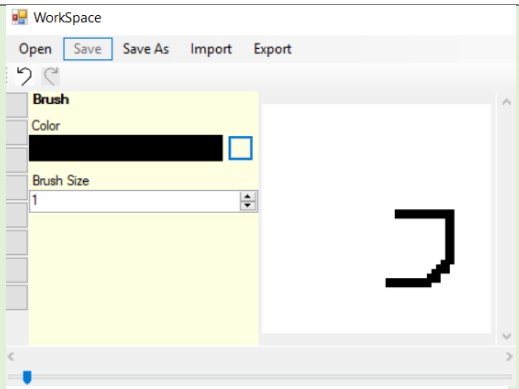
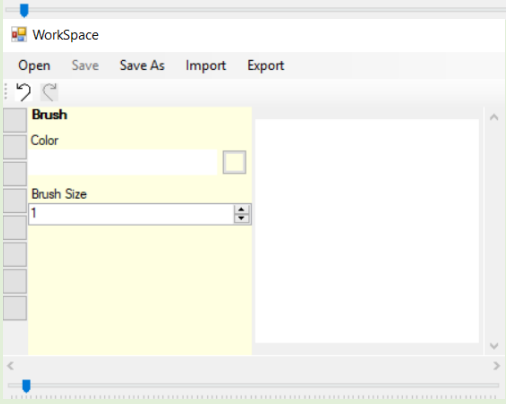
The error is caused by the program not checking whether the eyedropper is currently selected. It can be fixed by adding this check:

```
if (currentTool is EyedropperTool) {
    currentTool = ((EyedropperTool)currentTool).returnToTool;
} else {
    EyedropperTool newTool = new EyedropperTool("","",this,(ColorProperty)((Button)sender).Tag,currentTool);
    currentTool = newTool;
}
```

Fixing Error #8

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

```
color = (color == Color.Transparent) ? Color.White : color;
```

Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
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:

```
public class PixelTransformation
{
    public delegate Color Transformation(Color input);
    public Transformation transform;
```

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

```
public static PixelTransformation GrayScale {
    get {
        PixelTransformation transformation = new PixelTransformation();
        transformation.transform = delegate(Color input) {
            return Color.White;
        };
        return transformation;
    }
}
```

In this case, the colour is returned as white always. Thus when applying this transformation the code is very simple, the delegate is called and passed the colour it needs:

```
for (int x = 0; x < myWorkspace.image.fileWidth; x++) {
    for (int y = 0; y < myWorkspace.image.fileHeight; y++) {
        Color oldColor = myWorkspace.image.GetPixel(x,y);
        Color newColor = currentTransformation.transform(oldColor);
        myWorkspace.image.SetPixel(x,y,newColor);
    }
}
```

This closely follows the previously laid out design:

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

Implementing Grayscale

The grayscale can be implemented in accordance to the designed algorithm:

```
Grayscale(colour) {
      total = colour.R + colour.G + colour.B
      average = total / 3
      return new Color(average,average,average)
}
```

R, G and B all use the same average value

```
transformation.transform = delegate(Color input) {
    int total = input.R + input.G + input.B;
    int average = total / 3;
    return Color.FromArgb(average,average,average);
};
```

Implementing Black & White

```
BlackAndWhite(colour) {
      total = colour.R + colour.G + colour.B
      average = total / 3
      IF average < 127 THEN
            return White
      ELSE
            return Black
      END IF
}
```

However, this algorithm can be improved as the total does not need to be divided by 3, the total can be compared to 127 * 3:

```
transformation.transform = delegate(Color input) {
    int total = input.R + input.G + input.B;
    if (total < 381) {
        return Color.Black;
    } else {
        return Color.White;
    }
};
```

Implementing Invert

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

17/01/2020 Final Changes

Transparency Saving error

While doing Beta Testing with clients, an error was discovered that was not found in Alpha Testing. When saving layers, the Red, Green and Blue values 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.

```
currentColor = layer.pixels[x,y].Color;
stream.WriteByte(currentColor.R);
stream.WriteByte(currentColor.G);
stream.WriteByte(currentColor.B);
stream.WriteByte(currentColor.A);
```

However this means that the file format has been changed during beta testing – however my clients would like their older SIMP files to remain compatible.

Introducing File Versioning

Now that the file saving has been changed, the header has also been changed to denote that it is a different type of SIMP file. (SIM2 = SIMP2)

```
//SIMP check digits
stream.WriteByte((byte)'S');
stream.WriteByte((byte)'I');
stream.WriteByte((byte)'M');
stream.WriteByte((byte)'2');
```

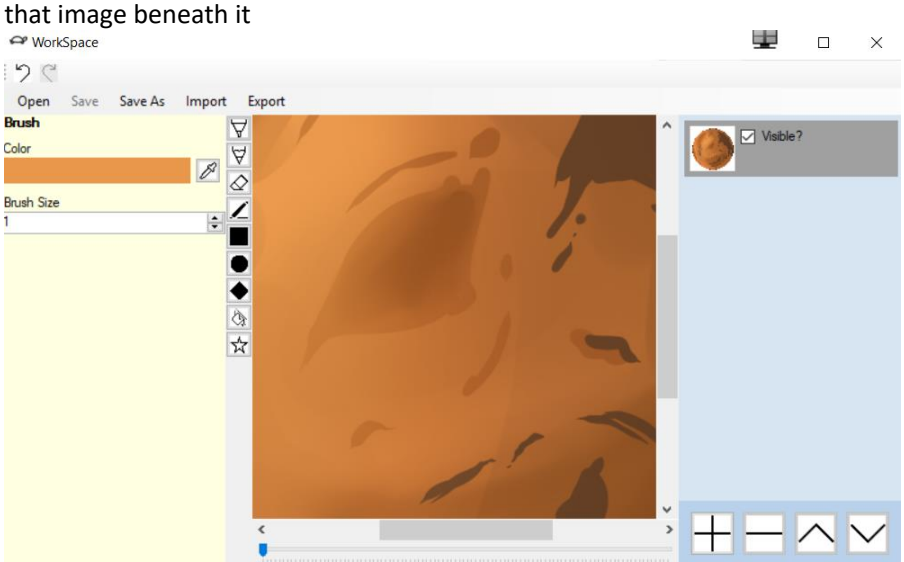
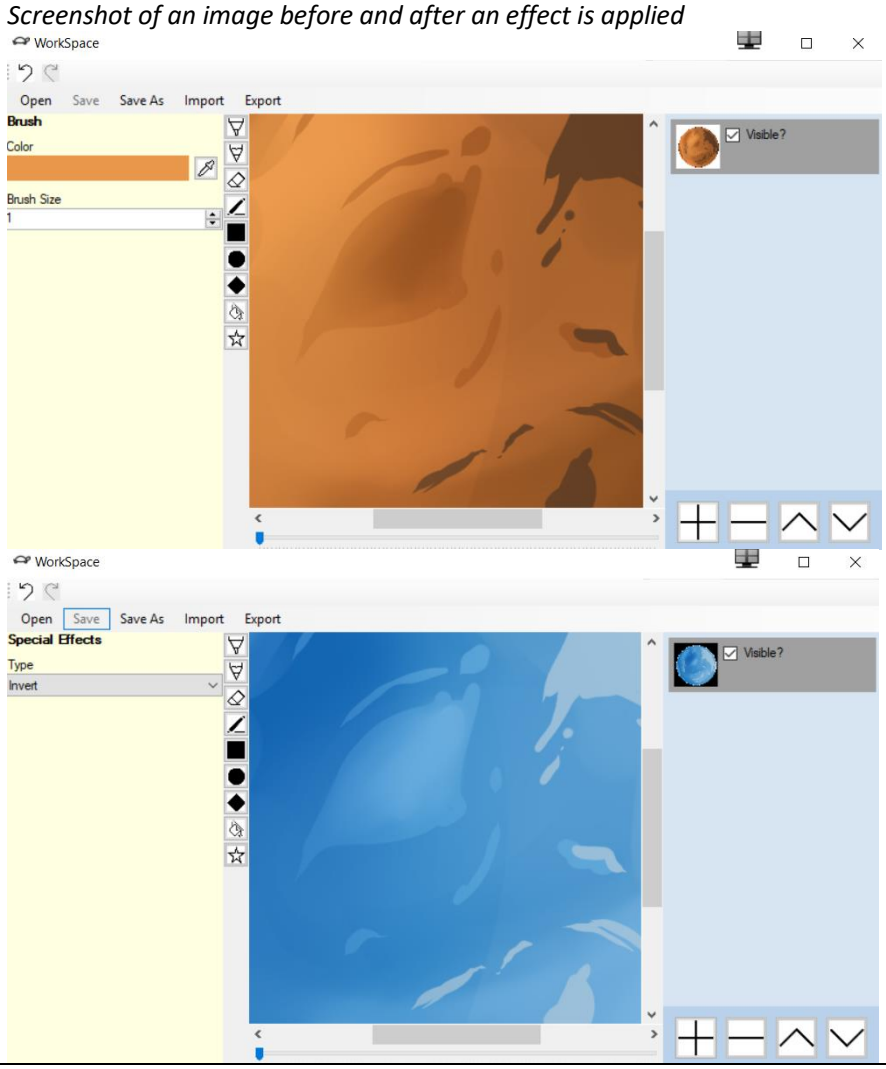
Then, when loading, the image, there is a switch case to decide what sort of file it is:

```
switch (checkString) {
    case "SIMP": //simp format v1 loading


    case "SIM2": //simp format v2 loading


    default:
        MessageBox.Show("File cannot be loaded. \n\nSIMP data could not be found within this file."
        return null;
```

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

```
R = SafeRead(stream);
G = SafeRead(stream);
B = SafeRead(stream);
A = SafeRead(stream);
newLayer.pixels[x,y] = new SolidBrush(Color.FromArgb(A,R,G,B));
```

Success Criteria Evaluation

Feature	Proof	Code
Section B – Other Editing Tools		
Eyedropper tool	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	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