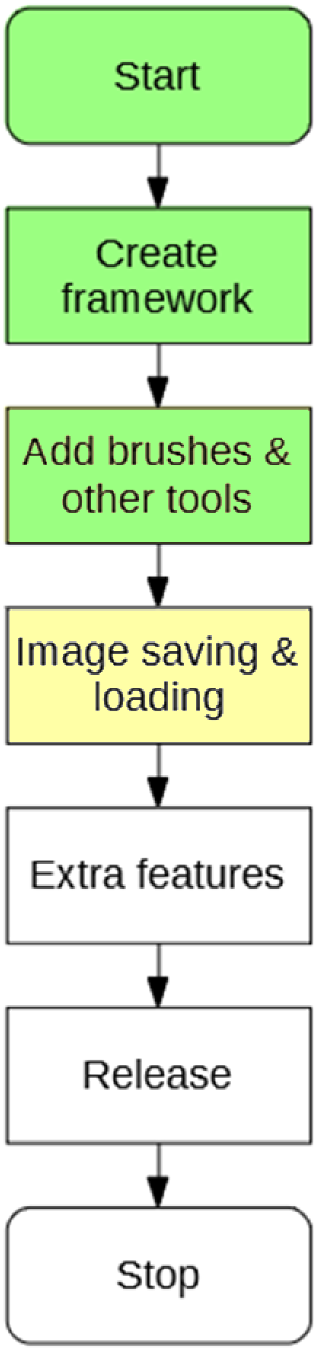
Section 4 – Saving & Loading

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 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.

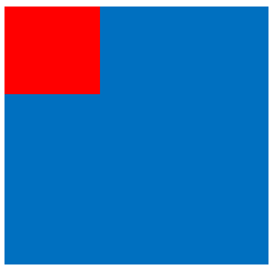
So thus the code for exporting is:

```
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:

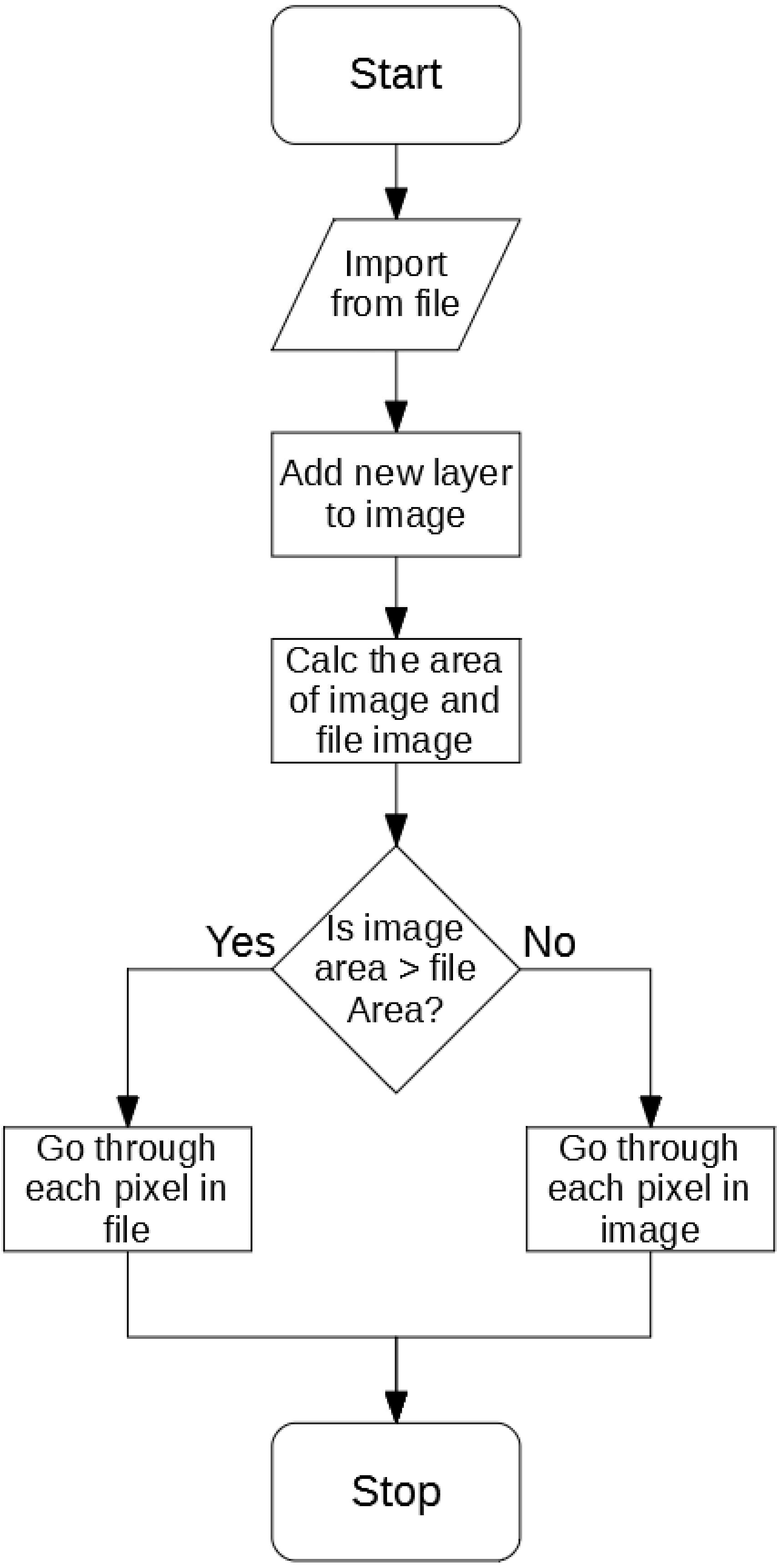
For example, if the red rectangle is the image, and the blue rectangle is the file trying to be imported:



Then it is clearly impractical to loop through each pixel of the file.

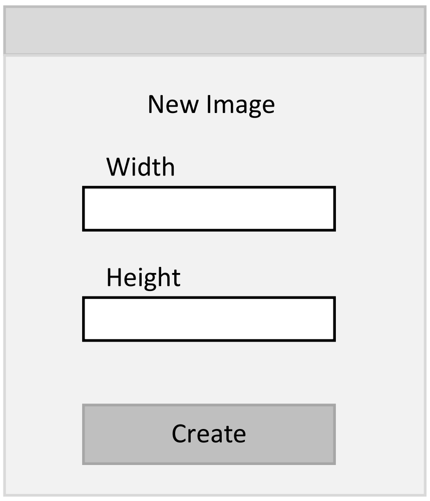
To decide which way around to iterate, a **heuristic** can be used. By finding the area of the image and file, a good guess can be made as to which one to iterate through (the one with the smaller area). This can be used to make sure that file loading is fast for larger images.

Thus the algorithm becomes:


Start
Import from file
Add new layer to image
Calc the area of image and file image
Is image area > file Area?
Yes
No
Go through each pixel in file
Go through each pixel in image
Stop

4.1.6 New GUI design

When creating images like this, some new GUIs may be needed. The most important one being the dialogue to create an image:

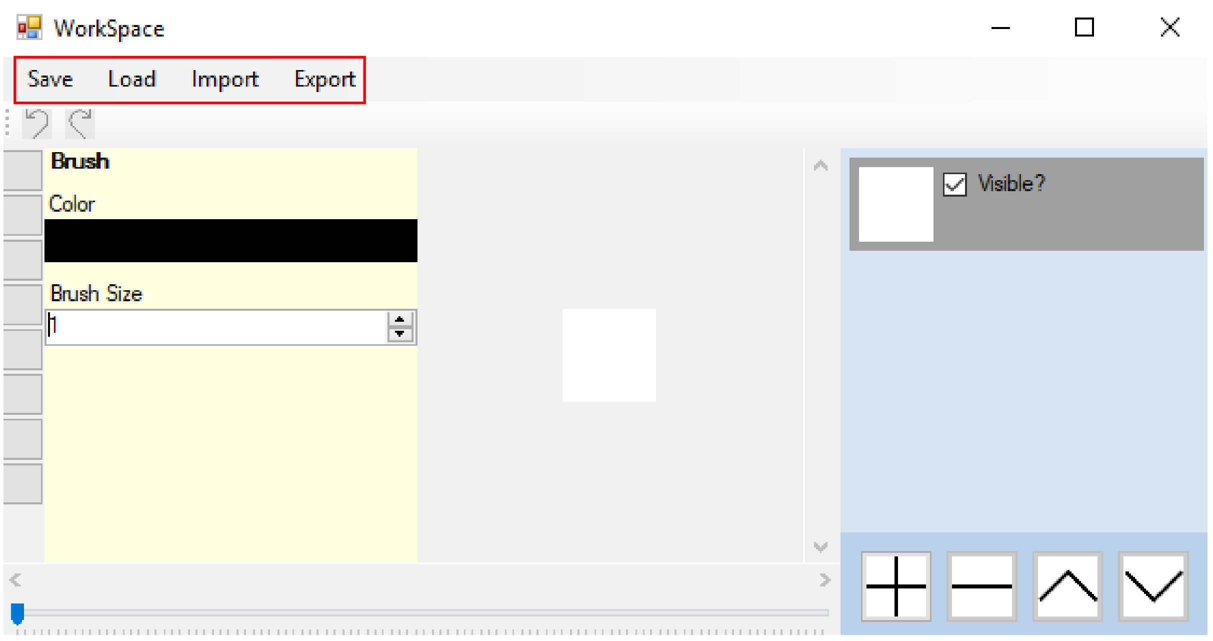


This will be started whenever a new image is needed to be made.

4.2 Development

04/12/2019 Form Design

The buttons necessary for the designed functions have been added to the form:



Stakeholders agreed that the top was a reasonable place to put those buttons, similar to other programs.

06/12/2019 Saving File

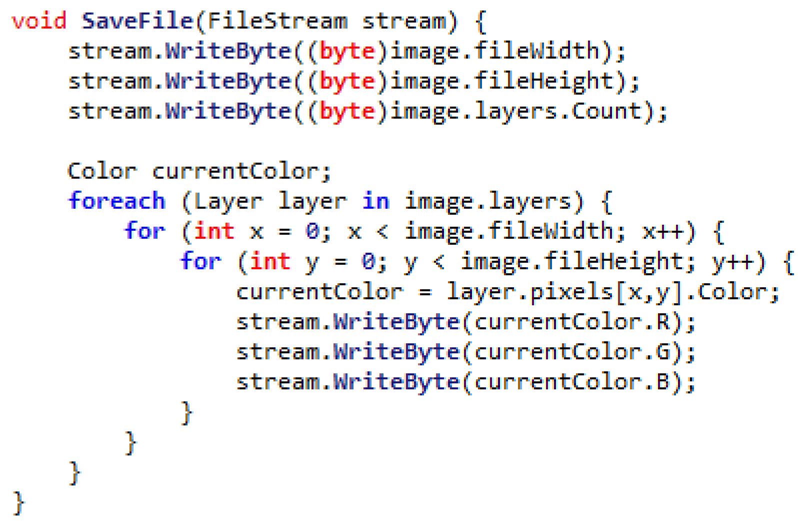
Writing algorithm

When saving the file, there are two potential classes that can be used for saving to a file; StreamWriter or FileStream.

StreamWriter allows saving of ASCII code to a file, FileStream saves raw binary. StreamWriter produces more easily readable files, but is less efficient as all numbers are encoded in ASCII. **I have decided** to use the FileStream class to reduce the file size of SIMP files and increase loading speed.

After this, the file saving algorithm can be implemented:

```
PROCEDURE WriteBody(file) {
      FOR EACH layer IN image.layers
            FOR x = 0 TO image.width
                  FOR y = 0 to image.height
                        color = layer.GetColor(x,y)
                        file.Write(color.R)
                        file.Write(color.G)
                        file.Write(color.B)
                  NEXT y
            NEXT x
      NEXT layer
}
```

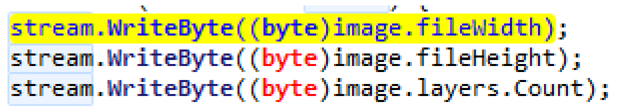


```
void SaveFile(FileStream stream) {
    stream.WriteByte((byte)image.fileWidth);
    stream.WriteByte((byte)image.fileHeight);
    stream.WriteByte((byte)image.layers.Count);

    Color currentColor;
    foreach (Layer layer in image.layers) {
        for (int x = 0; x < image.fileWidth; x++) {
            for (int y = 0; y < image.fileHeight; y++) {
                currentColor = layer.pixels[x,y].Color;
                stream.WriteByte(currentColor.R);
                stream.WriteByte(currentColor.G);
                stream.WriteByte(currentColor.B);
            }
        }
    }
}
```

Header Error

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

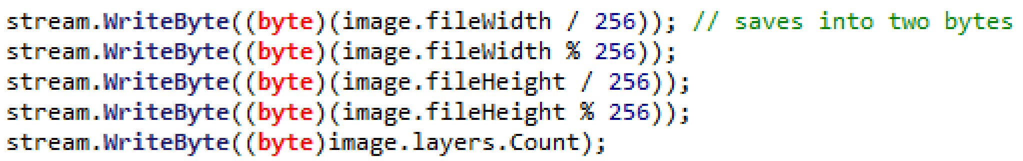


```
stream.WriteByte((byte)image.fileWidth);
stream.WriteByte((byte)image.fileHeight);
stream.WriteByte((byte)image.layers.Count);
```

This can be fixed by using two bytes each to store the width and height. This means that maximum size is increased from 255 to 255^2 (65,025), which is much larger than the maximum size. Whilst this may waste space for smaller images, the extra space needed is so minute that this is unlikely to be an issue.

It's assumed that 255 layers will never be created, so the same sort of error checking is not needed there.

This makes the new code for saving:

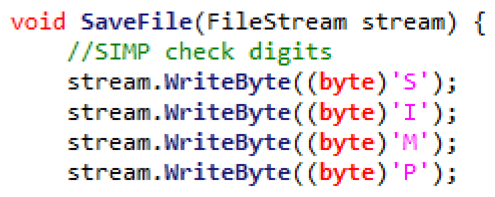


```
stream.WriteByte((byte)(image.fileWidth / 256)); // saves into two bytes
stream.WriteByte((byte)(image.fileWidth % 256));
stream.WriteByte((byte)(image.fileHeight / 256));
stream.WriteByte((byte)(image.fileHeight % 256));
stream.WriteByte((byte)image.layers.Count);
```

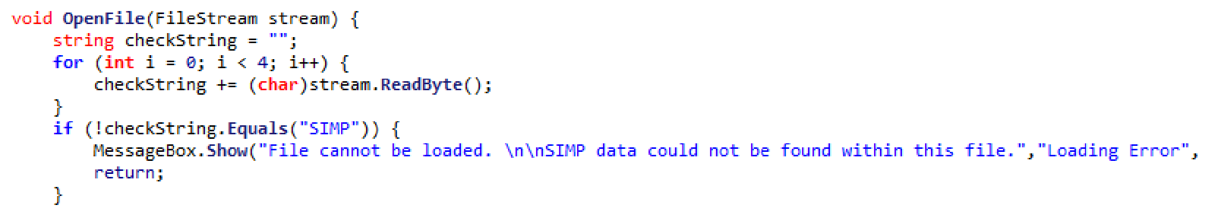
9/12/2019 More robust importing

Image importing has now been implemented according to 4.1.2, but it is currently not very robust. This especially happens when a non-SIMP file is attempted to be loaded.

This can be resolved by attaching the values 0x53, 0x49, 0x4D, 0x50 to the start of the file. This is the ASCII code for 'SIMP'. Thus when the file is attempting to be loaded, the program will check the first four bytes to see if they are 'SIMP'.



```
void SaveFile(FileStream stream) {
    //SIMP check digits
    stream.WriteByte((byte)'S');
    stream.WriteByte((byte)'I');
    stream.WriteByte((byte)'M');
    stream.WriteByte((byte)'P');
```



```
void OpenFile(FileStream stream) {
    string checkString = "";
    for (int i = 0; i < 4; i++) {
        checkString += (char)stream.ReadByte();
    }
    if (!checkString.Equals("SIMP")) {
        MessageBox.Show("File cannot be loaded. \n\nSIMP data could not be found within this file.","Loading Error",
        return;
    }
```

This means non SIMP files will not be loaded.

10/12/2019 Importing & Exporting

Image importing and exporting can now be introduced.

These both use C# base classes for importing & exporting, as the algorithms tied with PNG, JPEG and BMP are complex. Implementing these would be far outside the scope of the project, so it is suitable to use them here.

11/12/2019 Stakeholder feedback

After giving the program to my stakeholders, they had the following feedback:

"Images should be imported to be centred in the image"

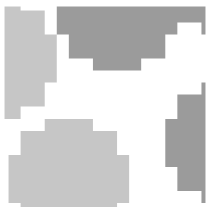
"Exporting while the image is zoomed doesn't work"

"Image should be the same size as the canvas"

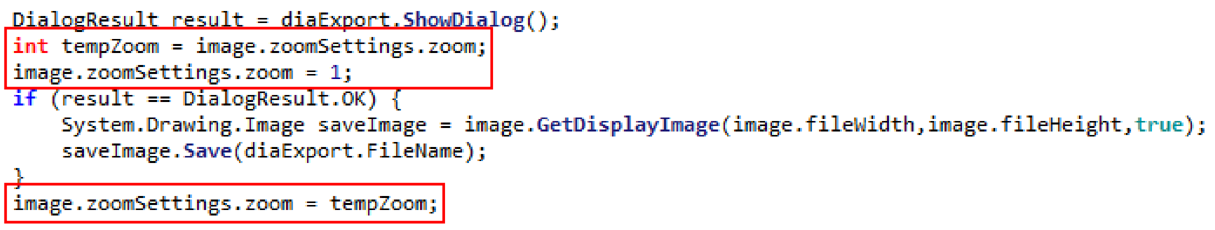
"You should be able to open an image from the start page"

Fixing Exporting

When the image is exported whilst the image is zoomed in, a zoomed in portion of the image was exported instead:



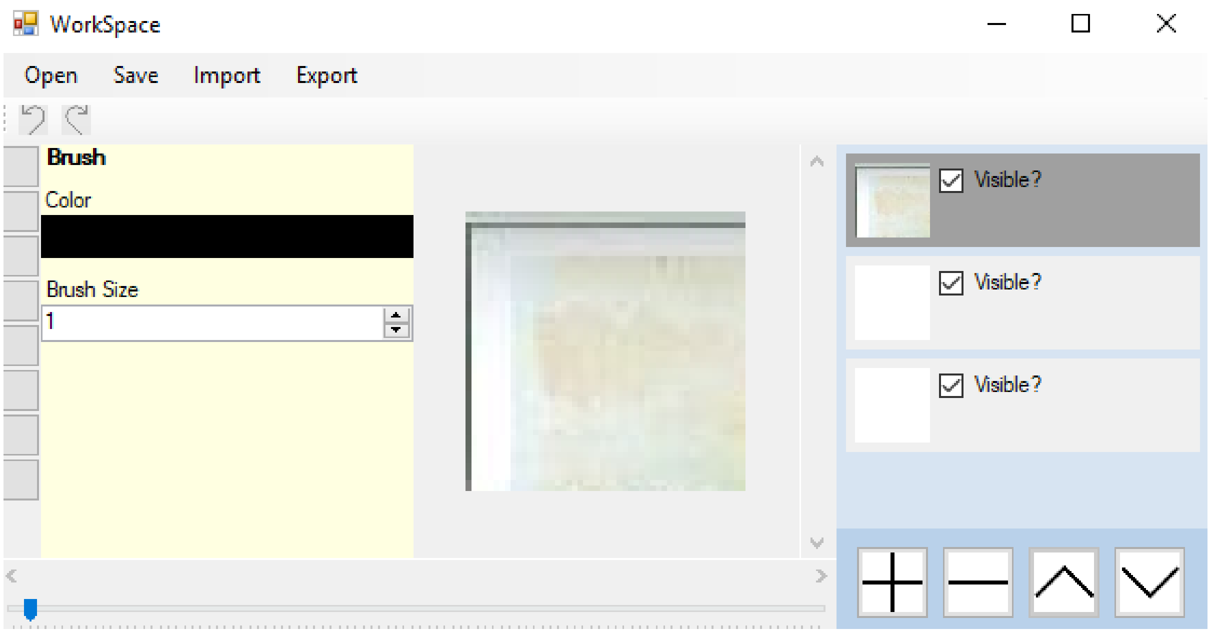
To fix this, the zoom of the image is temporarily set to 1, then exported. This means the proper image is always exported:



```
DialogResult result = diaExport.ShowDialog();
int tempZoom = image.zoomSettings.zoom;
image.zoomSettings.zoom = 1;
if (result == DialogResult.OK) {
    System.Drawing.Image saveImage = image.GetDisplayImage(image.fileWidth,image.fileHeight,true);
    saveImage.Save(diaExport.FileName);
}
image.zoomSettings.zoom = tempZoom;
```

Fixing Importing

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



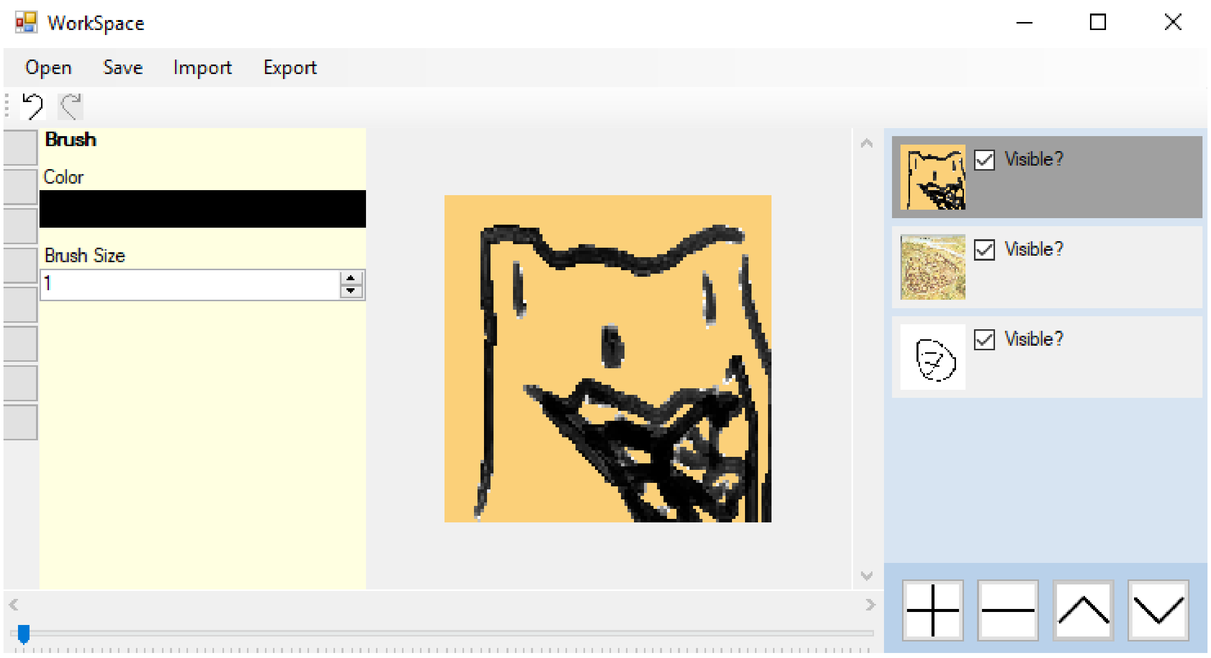
So, a different importing system needs to be used. A similar system to the one used for drawing the layer preview, where a foreach is used on each pixel of the image.

```
DrawIcon() {
        FOR x = 1 TO iconHeight
                FOR y = 1 TO iconHeight
                        imageX = (x * imageWidth) / iconWidth
                        imageY = (y * imageHeight) / iconHeight
                        DrawRectangle(x,y,1,1,colours[imageX,imageY])
                NEXT
        NEXT
}
```

Changing the code for importing to:

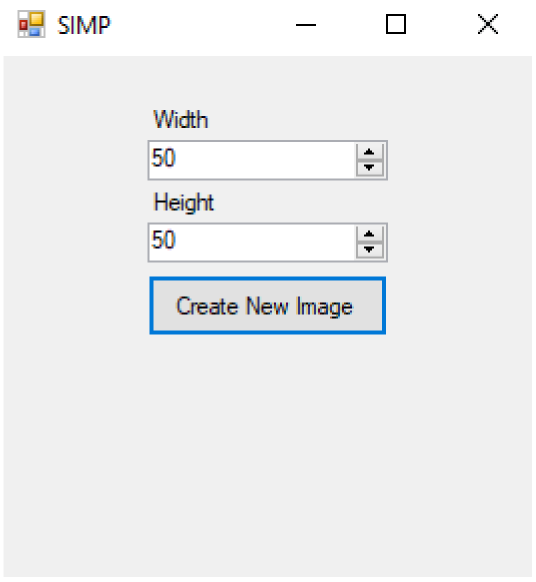
```
for (int x = 0; x < image.fileWidth; x++) {
    for (int y = 0; y < image.fileHeight; y++) {
        int imageX = (x * file.Width) / image.fileWidth;
        int imageY = (y * file.Height) / image.fileHeight;
        newLayer.pixels[x,y] = new SolidBrush(file.GetPixel(imageX,imageY));
    }
}
```

This means that images can now be properly imported:

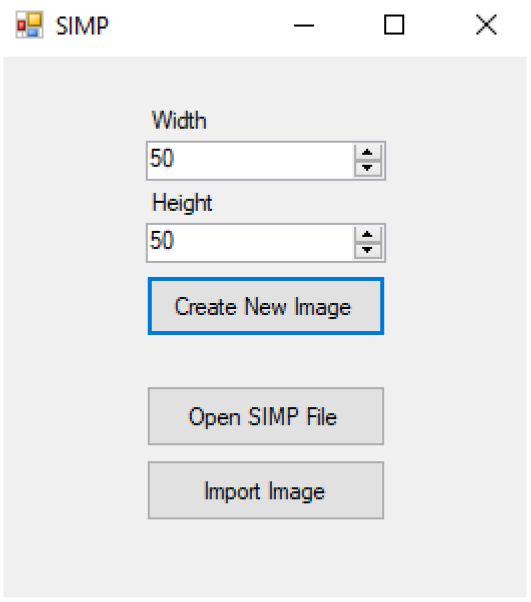


Improving Start page

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



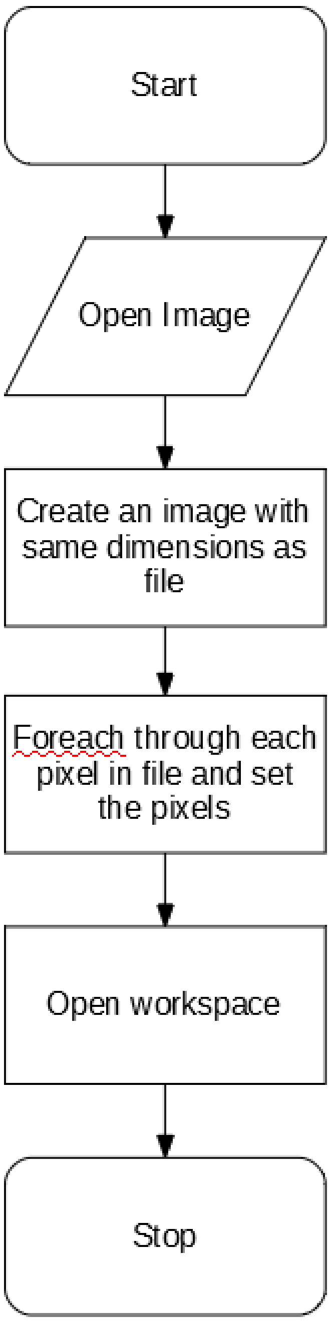
So, a few more buttons have been requested by the stakeholders:



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

```
void BtnOpenClick(object sender, EventArgs e)
{
    DialogResult result = diaOpen.ShowDialog();
    if (result == DialogResult.OK) {
        using (FileStream stream = new FileStream(diaOpen.FileName,FileMode.Open)) {
            Workspace.OpenFile(stream);
        }
    }
}
```

However importing an image will need to be designed slightly differently



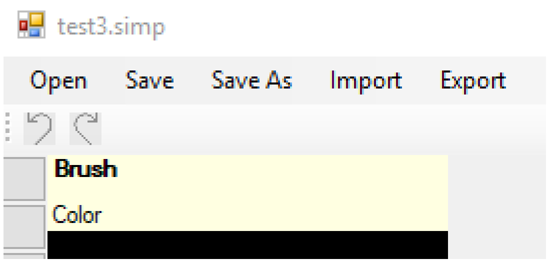
Start
Open Image
Create an image with same dimensions as file
Foreach through each pixel in file and set the pixels
Open workspace
Stop

After doing this, the start page is fully functional.

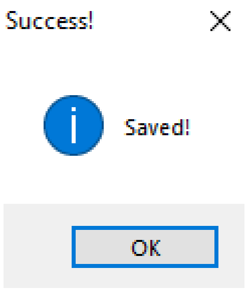
14/12/2019 Improvements to saving

Implementing Save As

The top bar currently only contains a Save button, which always makes a new file. This is not always useful if the user wants to save updated changes. The top bar has been updated to now include Save and Save As buttons:



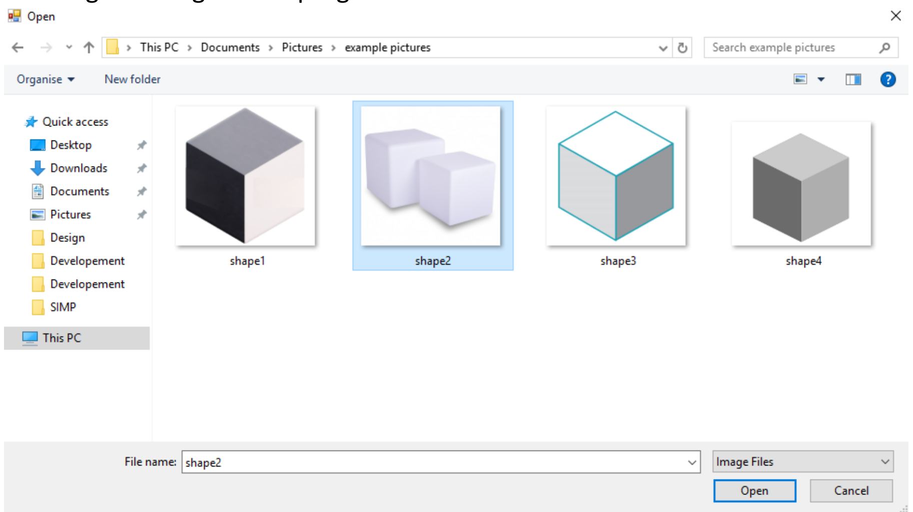
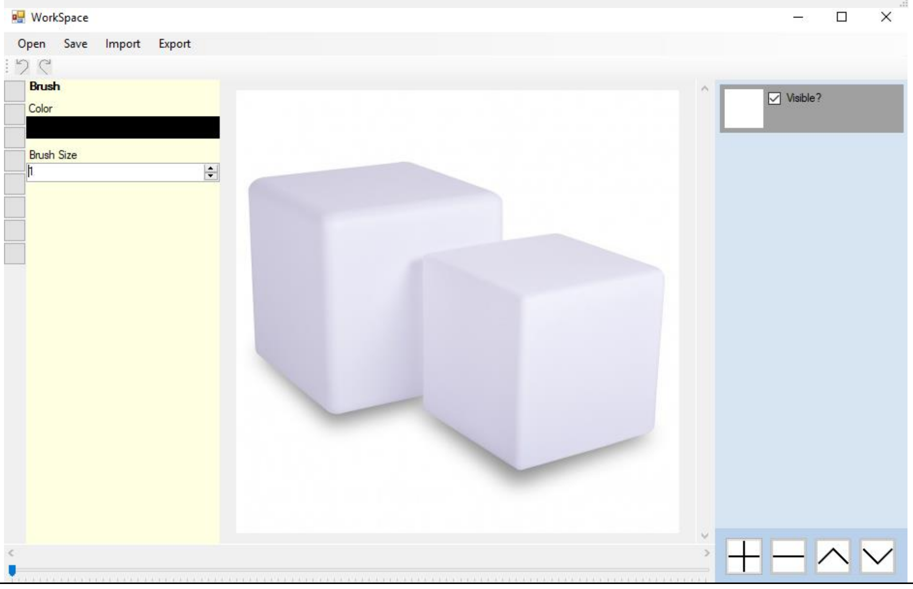
Notification boxes have also been added upon saving or exporting an image:

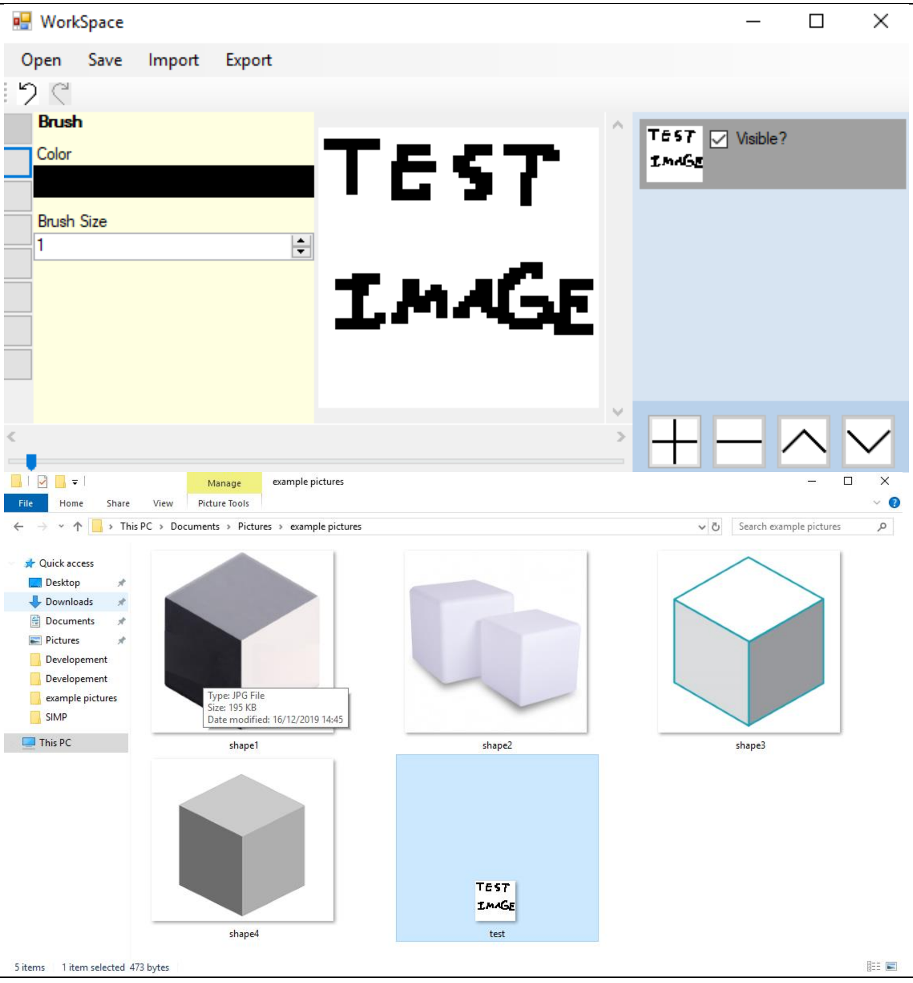
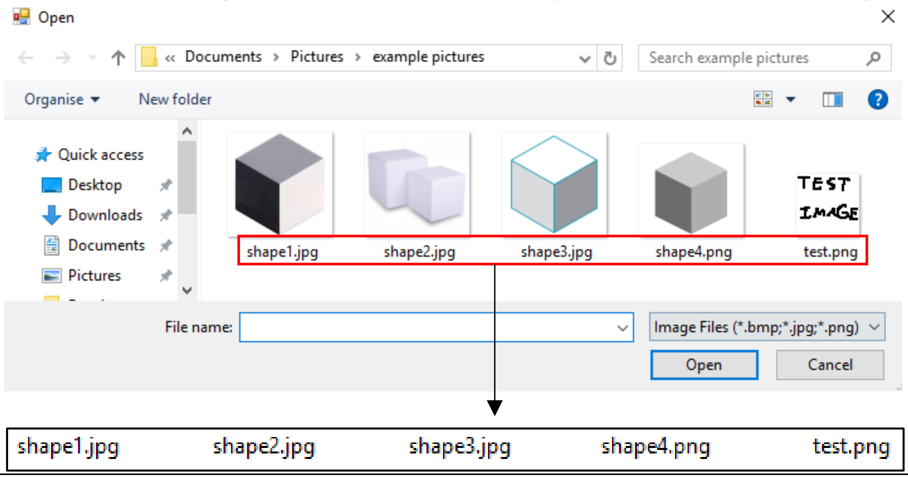

Success!

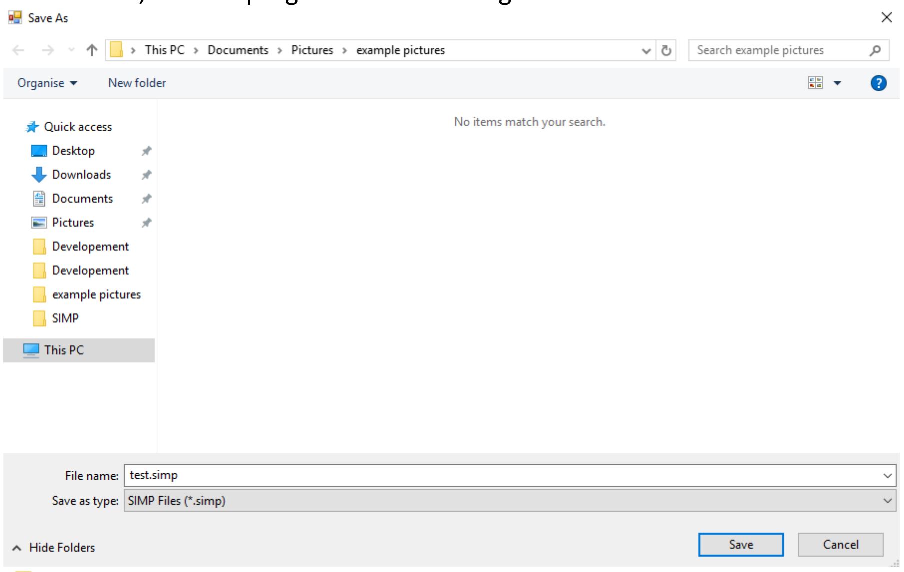
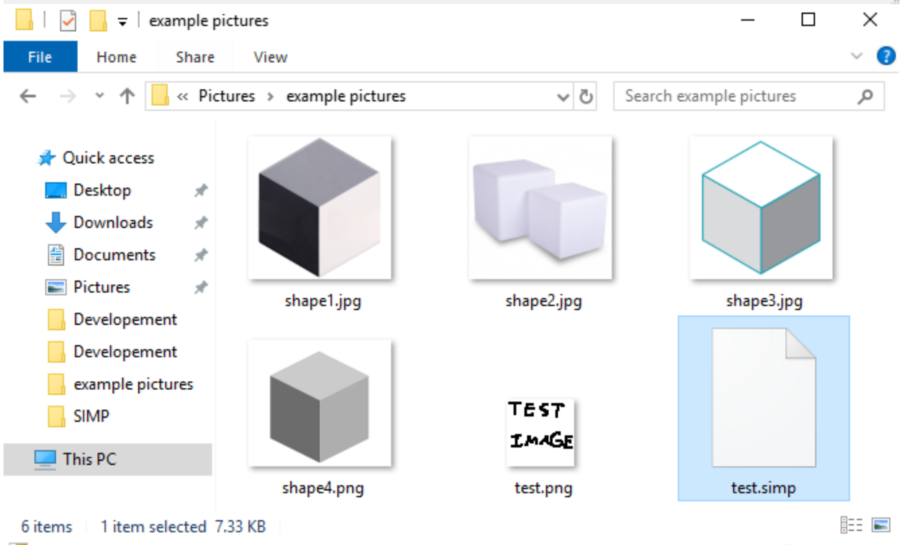
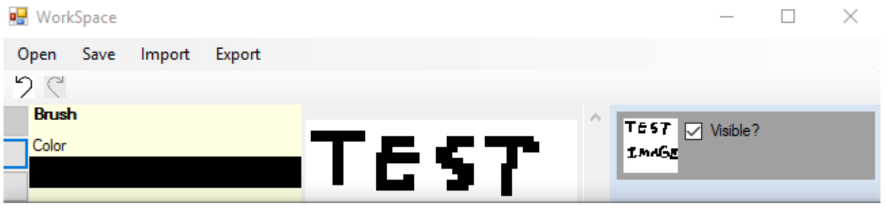
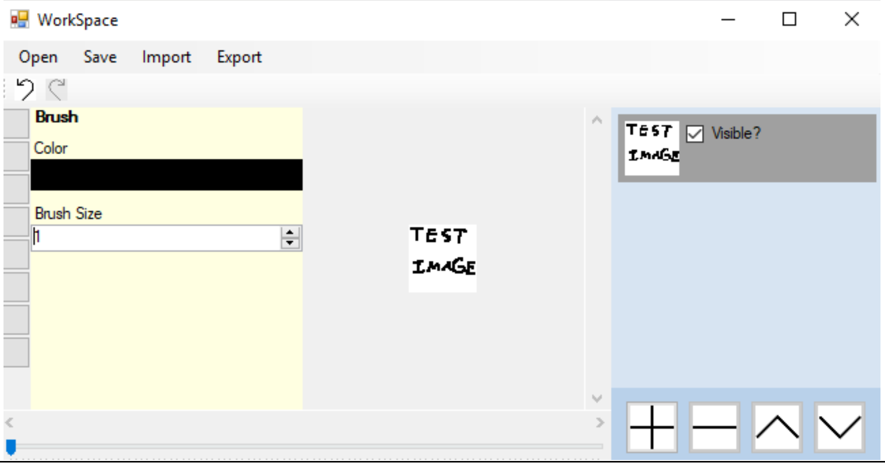
Saved!

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  Open This PC › Documents › Pictures › example pictures Search example pictures Organise New folder Quick access Desktop Downloads Documents Pictures Design Developement Developement SIMP This PC shape1 shape2 shape3 shape4 Open Cancel File name: shape2 Image Files  WorkSpace Open Save Import Export Brush Color Brush Size Visible? Brush Size: 1	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

	 WorkSpace Open Save Import Export Brush Color Brush Size 1 TEST IMAGE Visible? Manage example pictures File Home Share View Picture Tools This PC > Documents > Pictures > example pictures Search example pictures Quick access Desktop Downloads Documents Pictures Developement Developement example pictures SIMP This PC Type: JPG File Size: 195 KB Date modified: 16/12/2019 14:45 shape1 shape2 shape3 shape4 test 5 items 1 item selected 473 bytes	
Supporting PNG and JPEG	Screenshot showing the file browser which accepts both PNG and JPEG images  Open « Documents > Pictures > example pictures Search example pictures Organise New folder Quick access Desktop Downloads Documents Pictures TEST IMAGE shape1.jpg shape2.jpg shape3.jpg shape4.png test.png File name: Image Files (*.bmp;*.jpg;*.png) Open Cancel shape1.jpg shape2.jpg shape3.jpg shape4.png test.png	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  Save As This PC › Documents › Pictures › example pictures No items match your search. File name: test.simp Save as type: SIMP Files (*.simp) Organise New folder Quick access Desktop Downloads Documents Pictures Developement Developement example pictures SIMP This PC Hide Folders Save Cancel  example pictures « Pictures › example pictures shape1.jpg shape2.jpg shape3.jpg shape4.png test.png test.simp 6 items 1 item selected 7.33 KB File Home Share View Search example pictures Quick access Desktop Downloads Documents Pictures Developement Developement example pictures SIMP This PC  WorkSpace Open Save Import Export Brush Color TEST TEST IMAGE Visible?  WorkSpace Open Save Import Export Brush Color Brush Size 1 TEST IMAGE TEST IMAGE Visible?	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