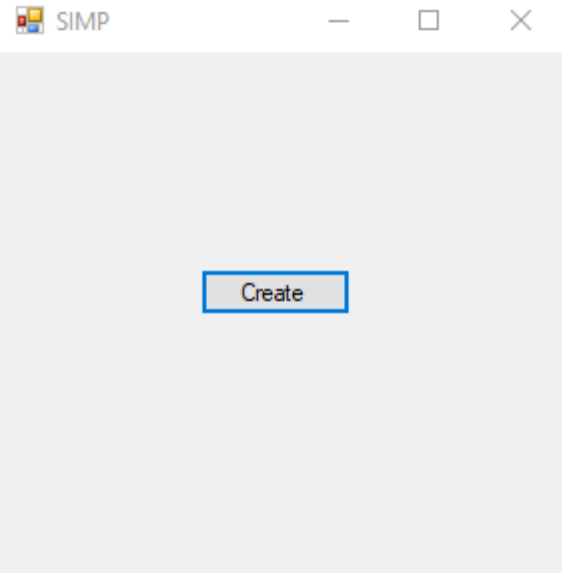
2.2 Developement

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.

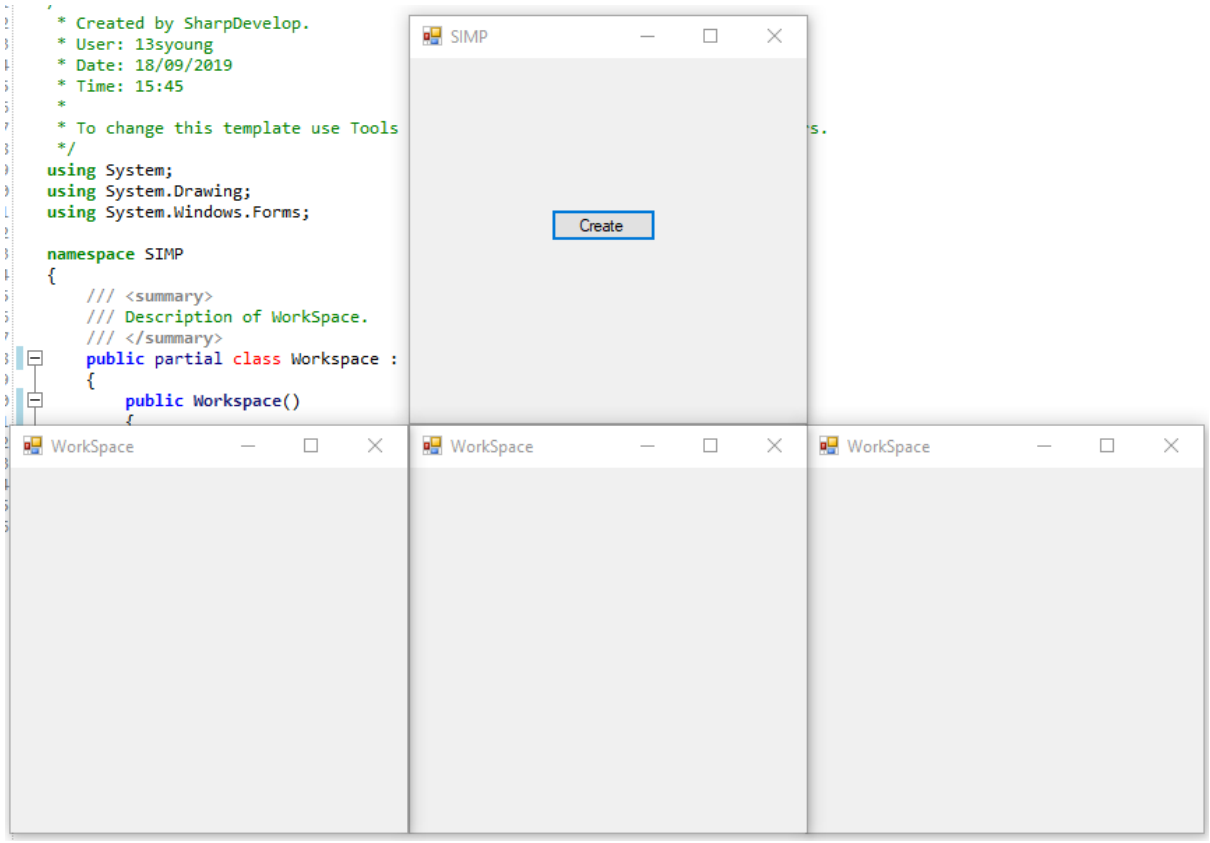
```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();


    }

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

The Workspace itself only contains code for creating a form:

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

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

As demonstrated here, multiple Workspaces can be created:



19/09/2019 – Remaining Class Implementation

Workspace implementation

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

<class>

Workspace

---

image

width

height

form

displayBox

IsScrollBarVisible()

GetScrollBarMax()

GetScrollBarValue()

SetScrollBarValue()

Properties:

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

The constructor has also been defined:

```
public Workspace()
{
    InitializeComponent();

    Constructor(100,100);
}

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

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

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

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

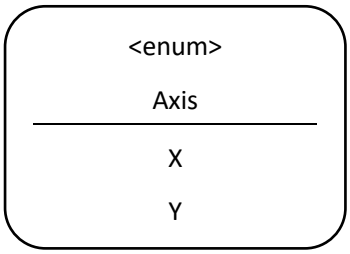
The necessary functions have also been defined:

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

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

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

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

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

Axis Implementation

The Axis enum has been implemented in accordance to 2.1.4.2:




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

Image Implementation

The image implementation has been completed in accordance to 2.1.4.1.

<class>

Image

colours[,]

width

height

zoomSetting

GetPixel()

SetPixel()

GetDisplayImage()

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

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

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

```
public Image(int width, int height)
{
    this.width = width;
    this.height = height;
    pixels = new Color[width,height];

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            pixels[x,y] = Color.White;
        }
    }

    zoomSettings = new ZoomSettings();
```

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

```
public Color GetPixel(int x, int y) {
    return pixels[x,y];
}

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

public Image GetDisplayImage() {
    throw new NotImplementedException();
}
```

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

ZoomSettings implementation

ZoomSettings has been implemented in accordance to 2.1.4.1:

<class>

ZoomSettings

centreLocation

zoomAmount

```
public class ZoomSettings
{
    public int zoom;

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

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

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

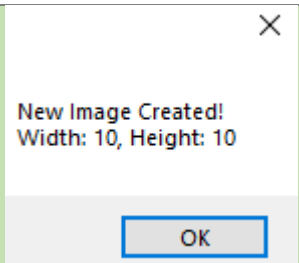
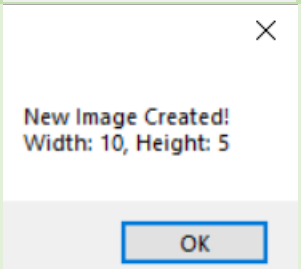
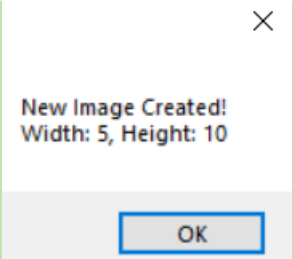
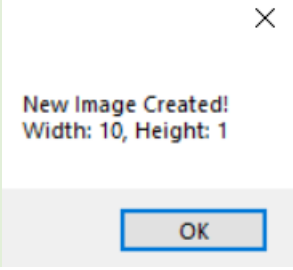
20/09/2019 – Unit Testing

Unit 2.2 Unit Test

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

```
//Create a new SIMP Image
SIMP.Image testImage = new Image(10,10);

//Displays the properties of that image
MessageBox.Show(string.Format(
    "New Image Created! " +
    "\nWidth: {0}, Height: {1}",
    testImage.width,testImage.height));
```

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	New Image Created! Width: 1, Height: 10 OK	The valid image is created
new Image(1,1)	6	A 1fpx, 1fpx image	New Image Created! Width: 1, Height: 1 OK	The valid image is created
new Image(10,0)	7	The parameters are rejected and no image is created	New Image Created! Width: 10, Height: 0 OK	The invalid image is not stopped
new Image(0,10)	8	The parameters are rejected and no image is created	New Image Created! Width: 0, Height: 10 OK	The invalid image is not stopped
new Image(0,0)	9	The parameters are rejected and no image is created	New Image Created! Width: 0, Height: 0 OK	The invalid image is not stopped
new Image(10000,10)	10	The parameters are rejected and no image is created	New Image Created! Width: 10000, Height: 10 OK	The invalid image is not stopped
new Image(10,10000)	11	The parameters are rejected and no image is created	New Image Created! Width: 10, Height: 10000 OK	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:

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

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

Fixing Error #8, #13

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

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

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

Fixing Error #10

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

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

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

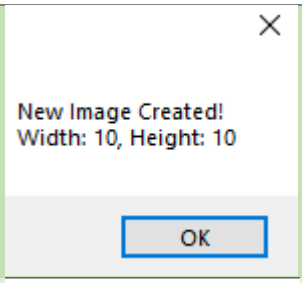
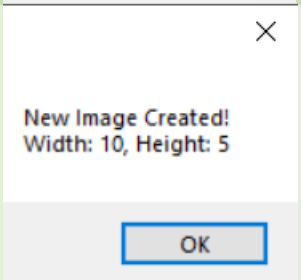
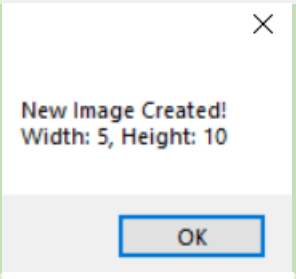
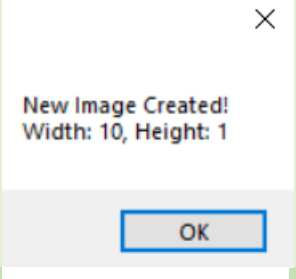
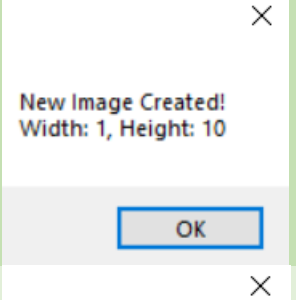
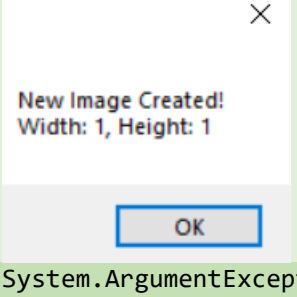
Fixing Error #11

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

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

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

Unit 2.2 Unit Test II

Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
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	`System.ArgumentException: Height is 0 or less`	A descriptive error is thrown

		no image is created		
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:

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

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

21/09/2019 Picture Box setup

Algorithm 2.3 – Resizing Picture Box

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

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

```
public void ResizePictureBox() {
    displayBox.Width = image.width;
    displayBox.Height = image.height;
}
```

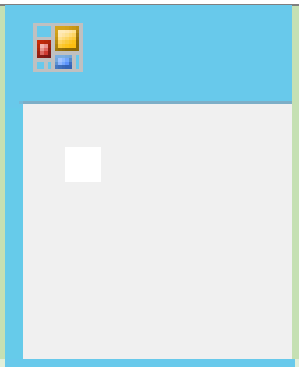
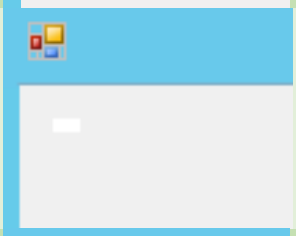
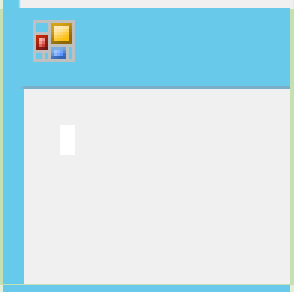
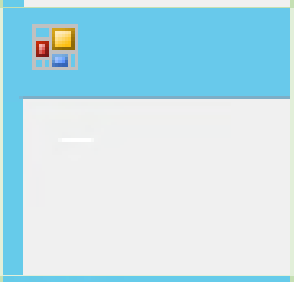
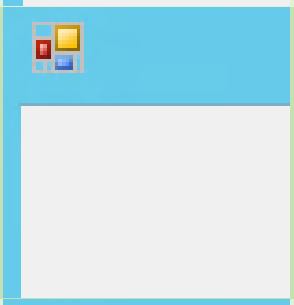
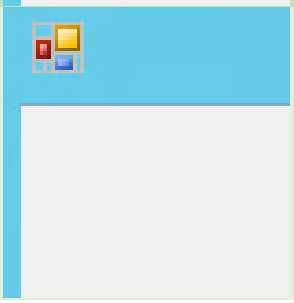
In the pseudocode, ResizePictureBox was included as part of the 'Image' class. However this has been changed to be part of the 'Workspace' class as Workspace contains both the Image and its relevant Picture Box. This enforces better **encapsulation** as Workspace manages the interaction between the Image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

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

```
Constructor(
    SimpConstants.IMAGE_DEFAULT_WIDTH,
    SimpConstants.IMAGE_DEFAULT_HEIGHT
);

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

    public static int IMAGE_DEFAULT_WIDTH = 540;
    public static int IMAGE_DEFAULT_HEIGHT = 360;
}
```

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

Unit 2.3 Unit Test

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

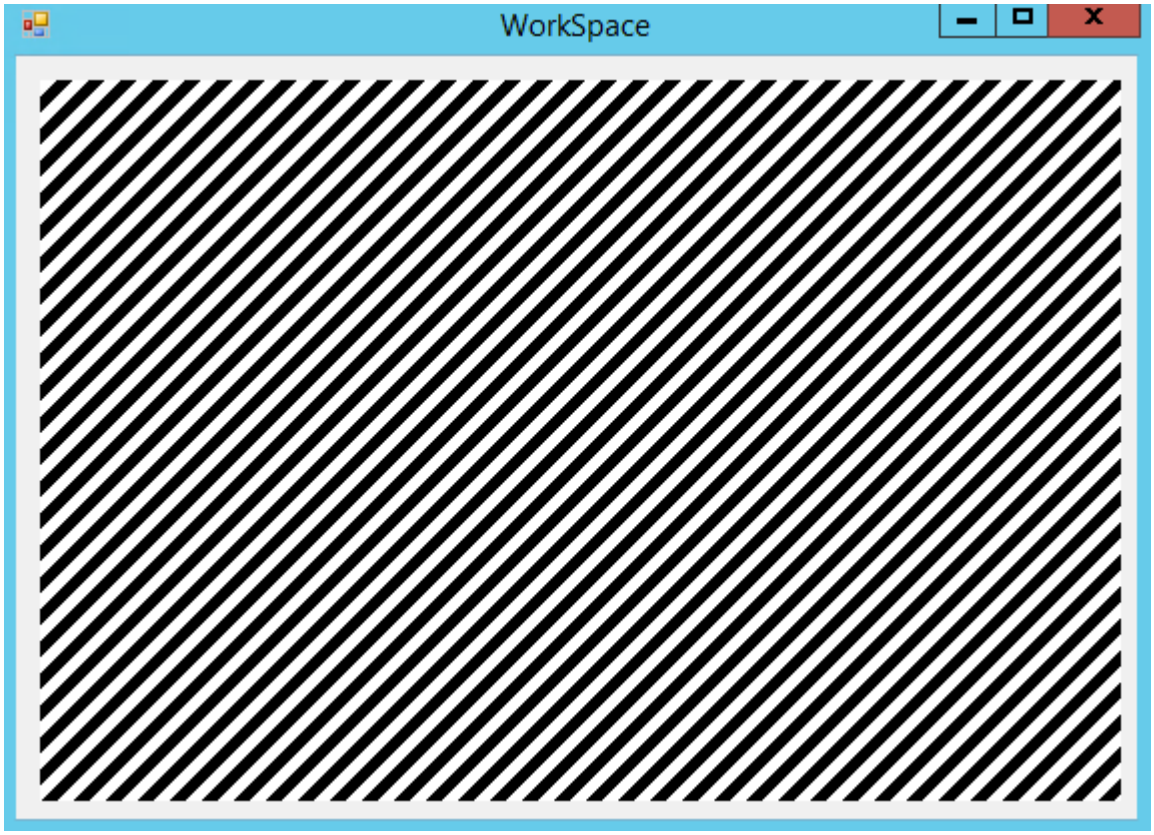
This algorithm has been constructed, according to the following pseudocode:

```
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
}
```

```
public System.Drawing.Image GetDisplayImage() {
    Bitmap newImage = new Bitmap(width,height);
    Color colour;

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            colour = pixels[x,y];
            newImage.SetPixel(x,y,colour);
        }
    }

    return newImage;
}
```

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

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



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

22/09/2019 Setting Pixels

Algorithm 2.5 – 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()
}
```

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

However there has been a change, as the call to Display() has been removed. This makes it so that the image will *not* be redrawn each time a pixel 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:

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

}
```

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

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

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

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

Unit 2.5 Unit Test II

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)
}
```

```
private void RelocatePictureBox() {
    int X = (width - image.width) / 2;
    int Y = (height - image.height) / 2;

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

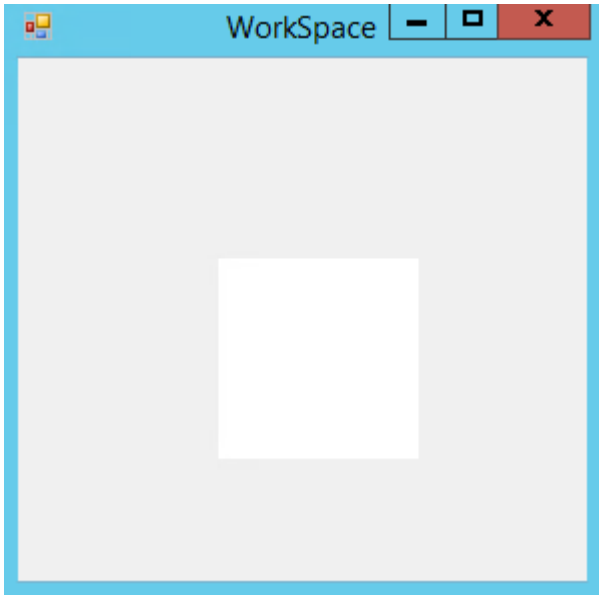
However, some additional code must be added to make the picture box constantly update. To do this a small timer module has been added to the code which calls the following procedure every millisecond:

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

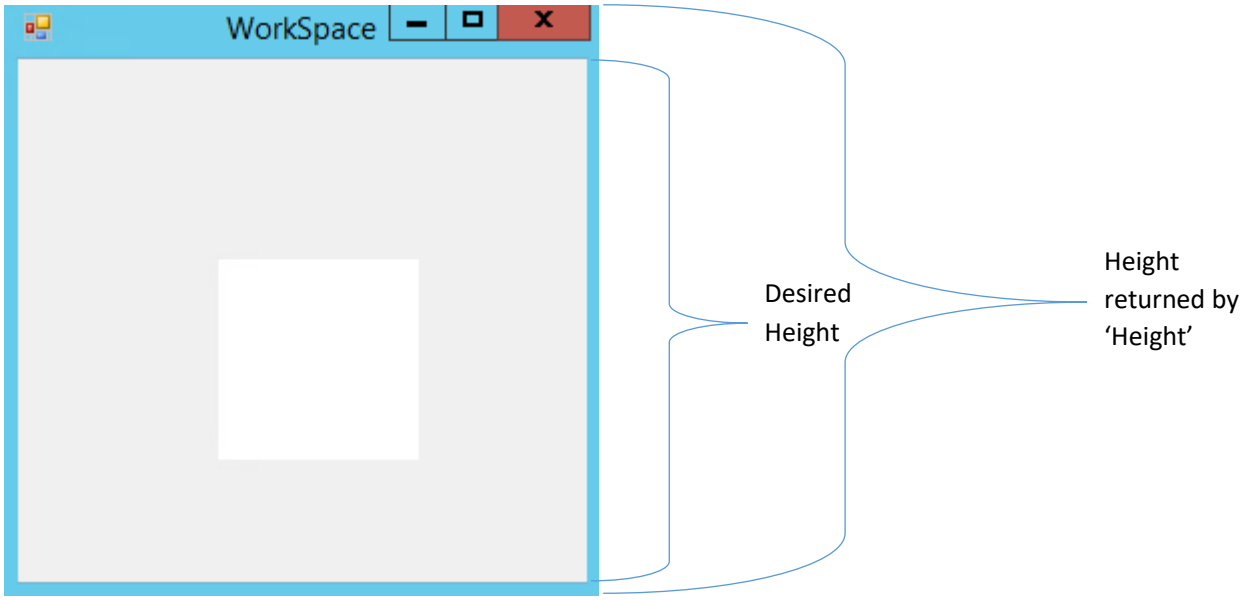
However this is still not enough, as 'width' and 'height' are currently constant and do not update when the form is resized. This can be solved by adding a new event called when the workspace is resized:

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

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



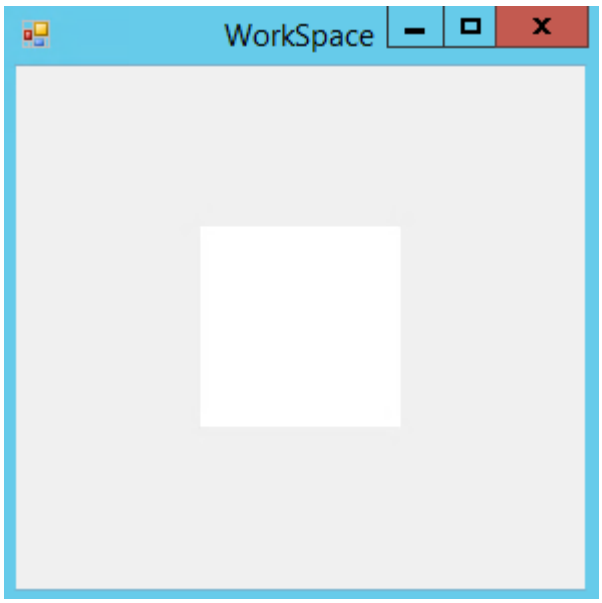
This is caused by the fact that calling the width of the form will return the **entire** width of the form, including its borders. For example:



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

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

Resulting in:



24/09/2019 Minimum Form Size

Algorithm 2.7 – Minimum Form Size

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

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

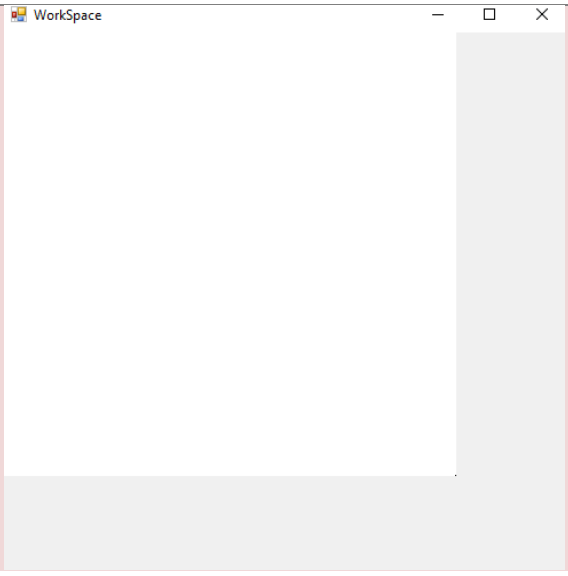
```
MinimumSize = new Size(image.width,image.height);
```

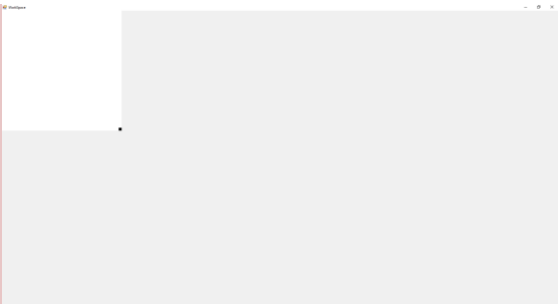
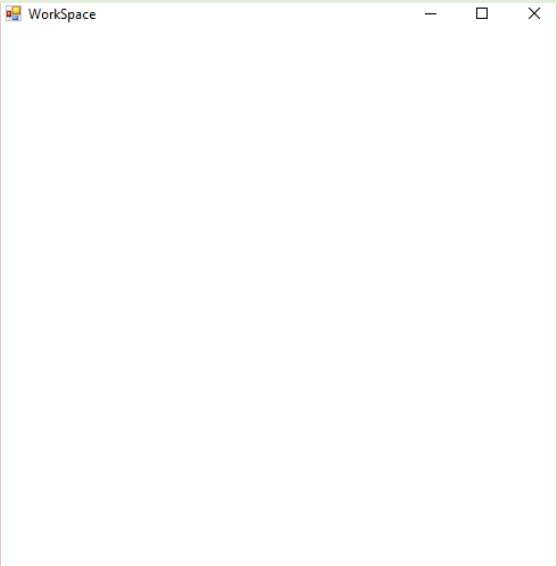
There is a small change in the way that the minimum size is implemented, as it must be added as a size object. Functionally this is the same.

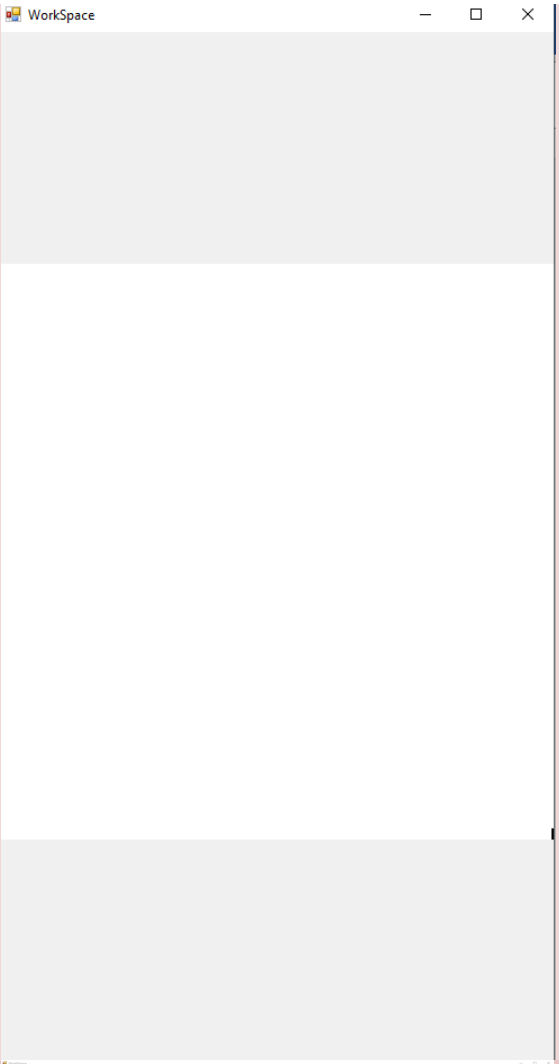
Unit Test 2.7

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

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

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

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:

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

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

    return false;
}
```

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

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

So the tests can now be repeated:

Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
Form is started at normal size	1	Image Displays in the middle of the form		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.

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

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

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

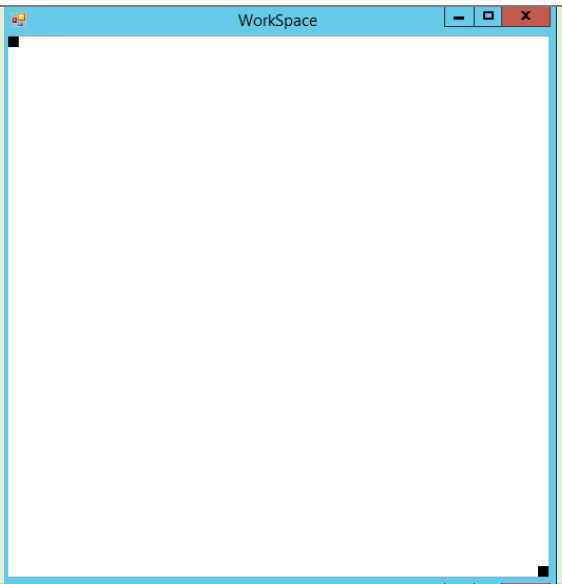
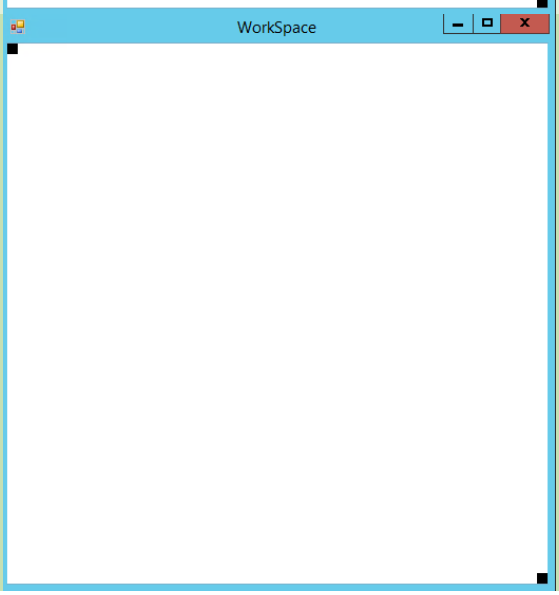
```
public static int WINDOWS_TOP_BAR_HEIGHT;
public static int WINDOWS_BOTTOM_BAR_HEIGHT;
public static int WINDOWS_LEFT_BAR_WIDTH;
public static int WINDOWS_RIGHT_BAR_WIDTH;
```

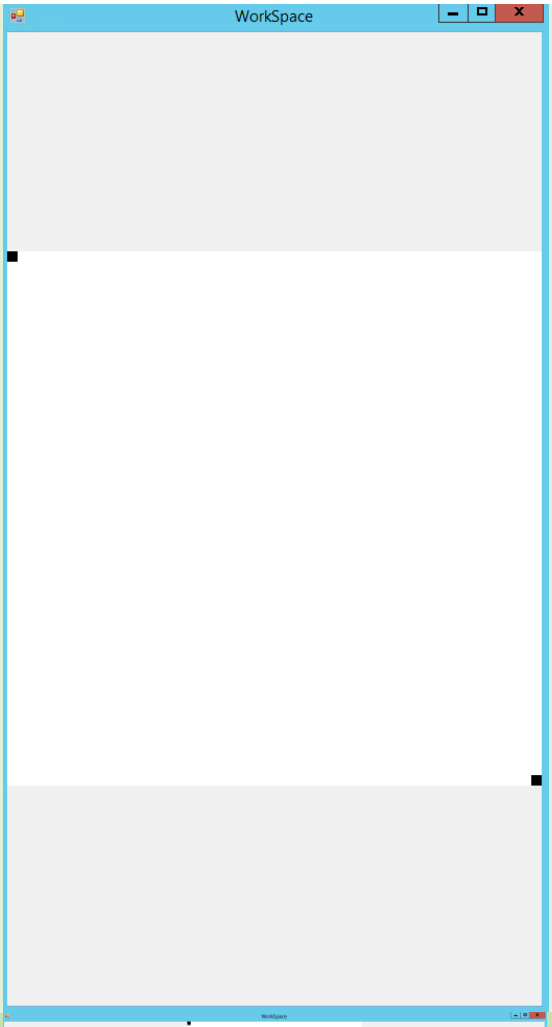
They can be calculated using the following code:

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

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

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

So now they can be included in the size calculations

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

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

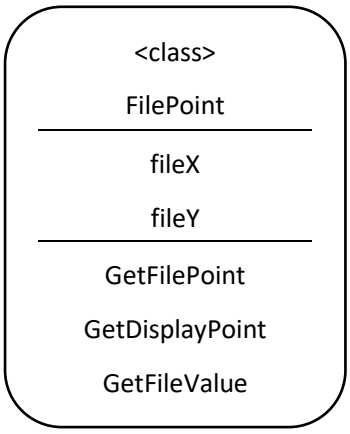
This means that all errors relating to resizing the Form has been resolved.

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

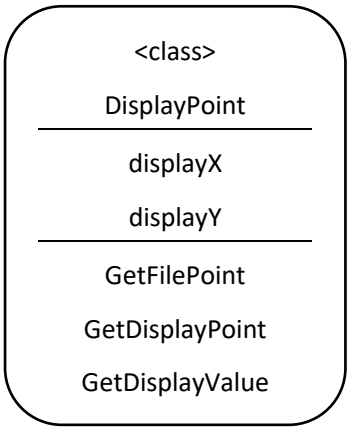



```
public class FilePoint
{
    private int _fileX;
    private int _fileY;

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

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

DisplayPoint

A basic structure has been implemented for FilePoint, in accordance to 2.1.6.3

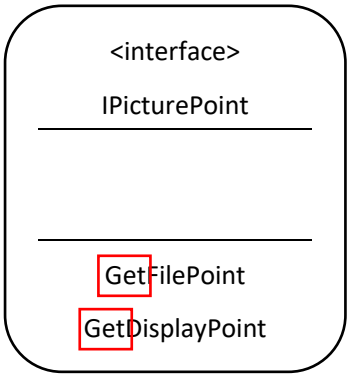



```
public class DisplayPoint
{
    private int _displayX;
    private int _displayY;

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

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

IPicturePoint

IPicturePoint has been implemented, in accordance to 2.1.6.1




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

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

FilePoint functions

The remaining functions for FilePoint have been implemented:

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

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

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

        case Axis.Y:
            return _fileY;
            break;

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

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

DisplayPoint functions

The remaining functions for DisplayPoint have been implemented:

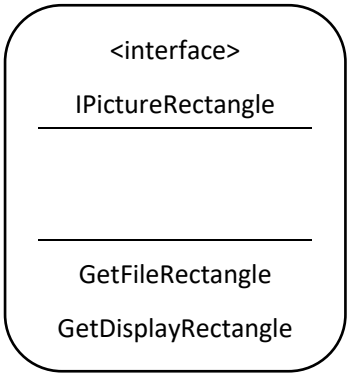
```
public FilePoint ToFilePoint() {
    throw new NotImplementedException();
}

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

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

        case Axis.Y:
            return _displayY;

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

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

26/09/2019 Rectangle Redesign

IPictureRectangle

IPictureRectangle has been implemented, in accordance to 2.1.6.4




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

FileRectangle

The FileRectangle class has been implemented in accordance to 2.1.6.5

<class>

FileRectangle

topLeftCorner

bottomRightCorner

*GetFileRectangle

*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetFileSizeInAxis

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

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

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

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

    public FilePoint GetTopLeftCorner() {
        return _fileTopLeftCorner;
    }

    public FilePoint GetBottomRightCorner() {
        return _fileBottomRightCorner;
    }

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

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

DisplayRectangle

The DisplayRectangle class has been implemented in accordance to 2.1.6.6

<class>

DisplayRectangle

topLeftCorner

bottomRightCorner

*GetFileRectangle

*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetDisplaySizeInAxis

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

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

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

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

    public DisplayPoint GetTopLeftCorner() {
        return _displayTopLeftCorner;
    }

    public DisplayPoint GetBottomRightCorner() {
        return _displayBottomRightCorner;
    }

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

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

ZoomSettings upgrade

The ZoomSettings class has been upgraded in accordance to 2.1.6.7

<class>

ZoomSettings

centreLocation

zoomAmount

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

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

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

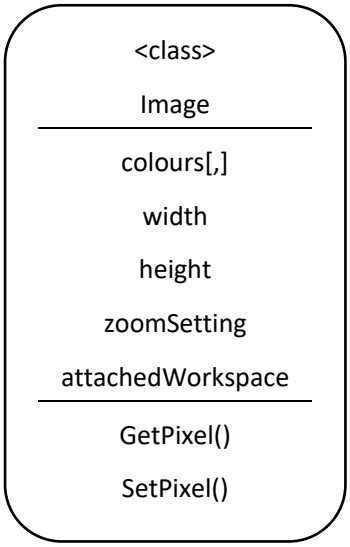
```
constructor(_centreLocation) {
      centreLocation = _centreLocation
      zoomAmount = 1
}
```

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

27/09/2019 Remaining Class Redesigning

Image Redesign

The remaining property for Image has been implemented, the attachedWorkspace




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

So thus the constructor has been edited:

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

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

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

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

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

Workspace Redesign

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

<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:

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

Code for determining locations of displayBox:

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

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

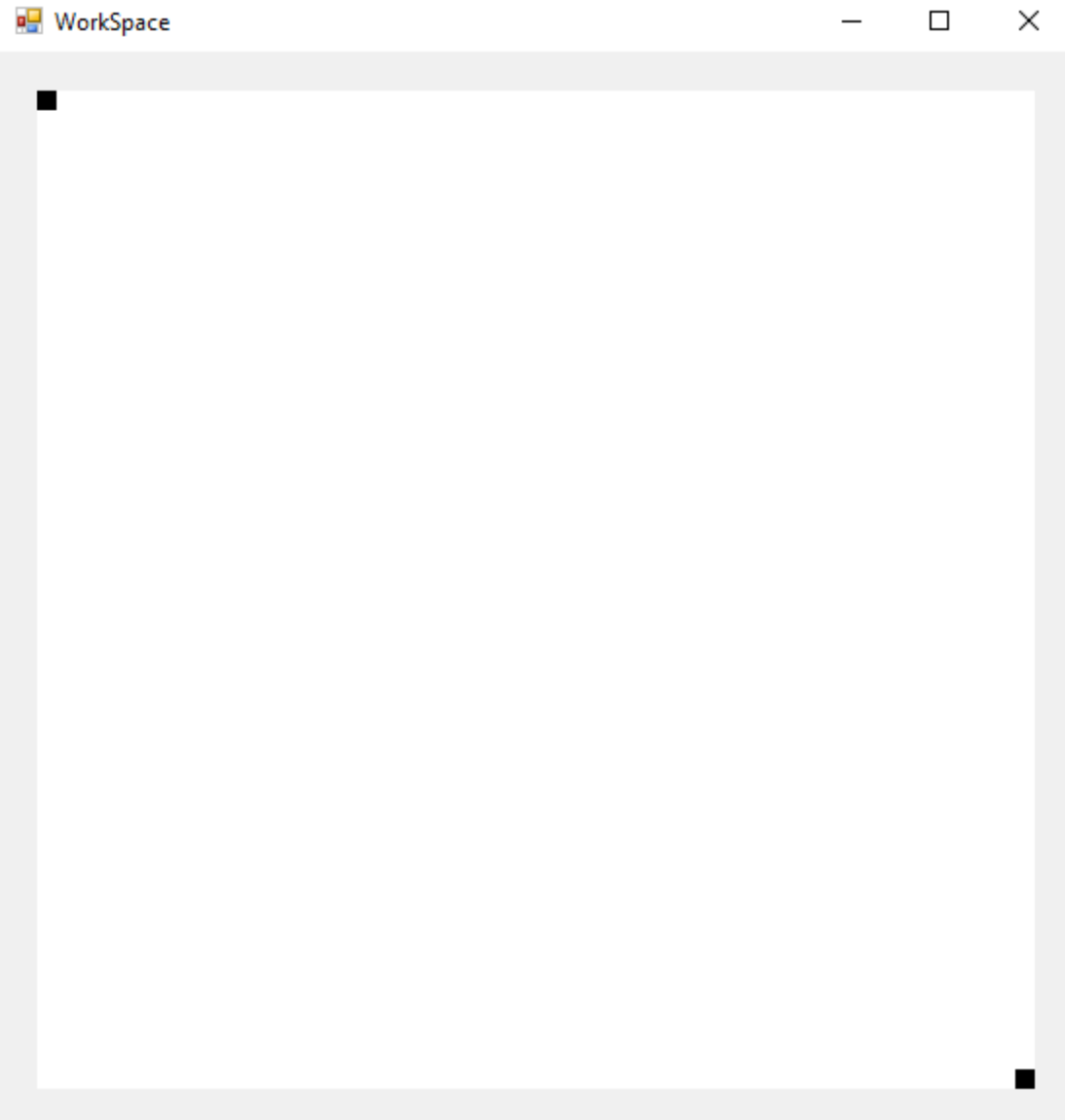
Code for determining minimum size:

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

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

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

The resulting program cannot be made smaller than this size:



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

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

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

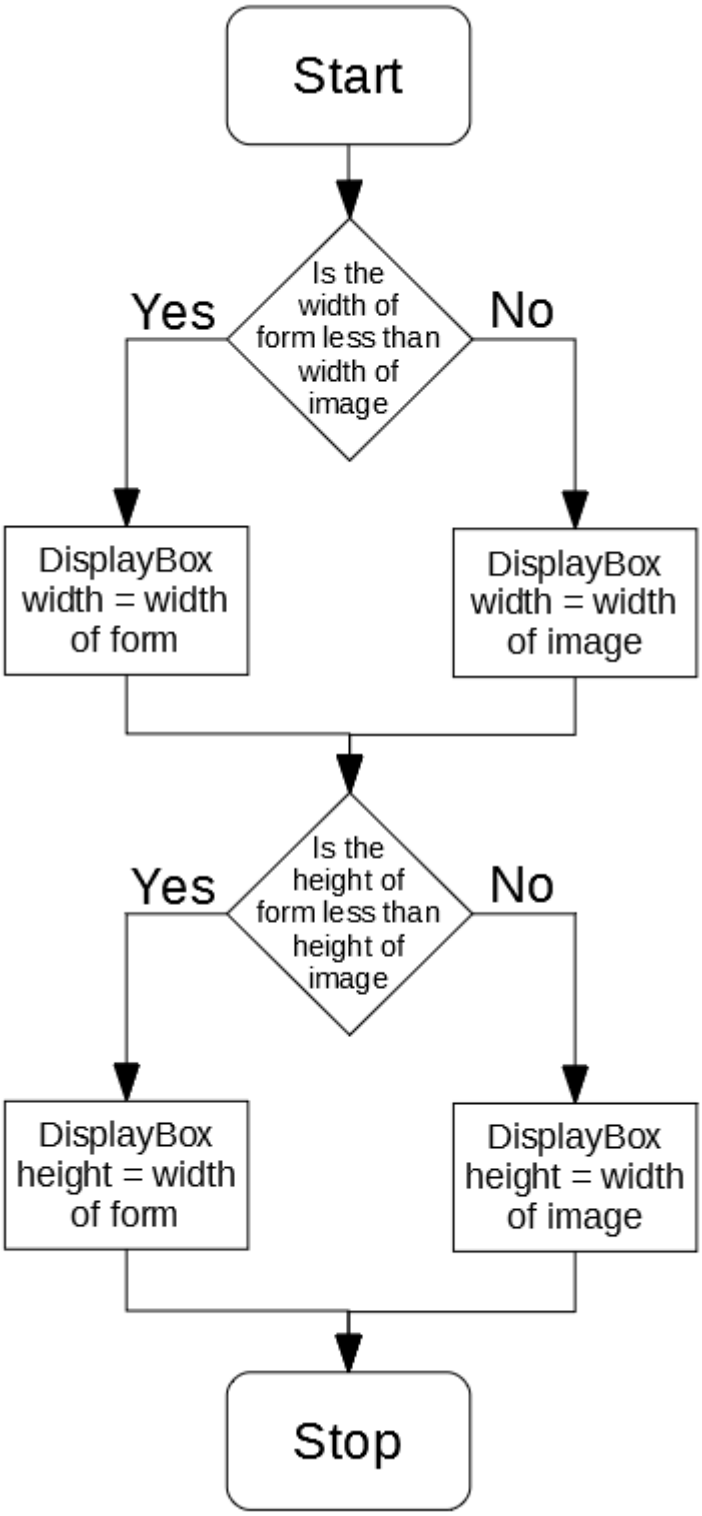
Compared to:

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

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

Relocating displayBox change

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



28/09/2019 Implementing CheckImageSize

EAxisMode

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

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

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

Implementing CheckImageSize

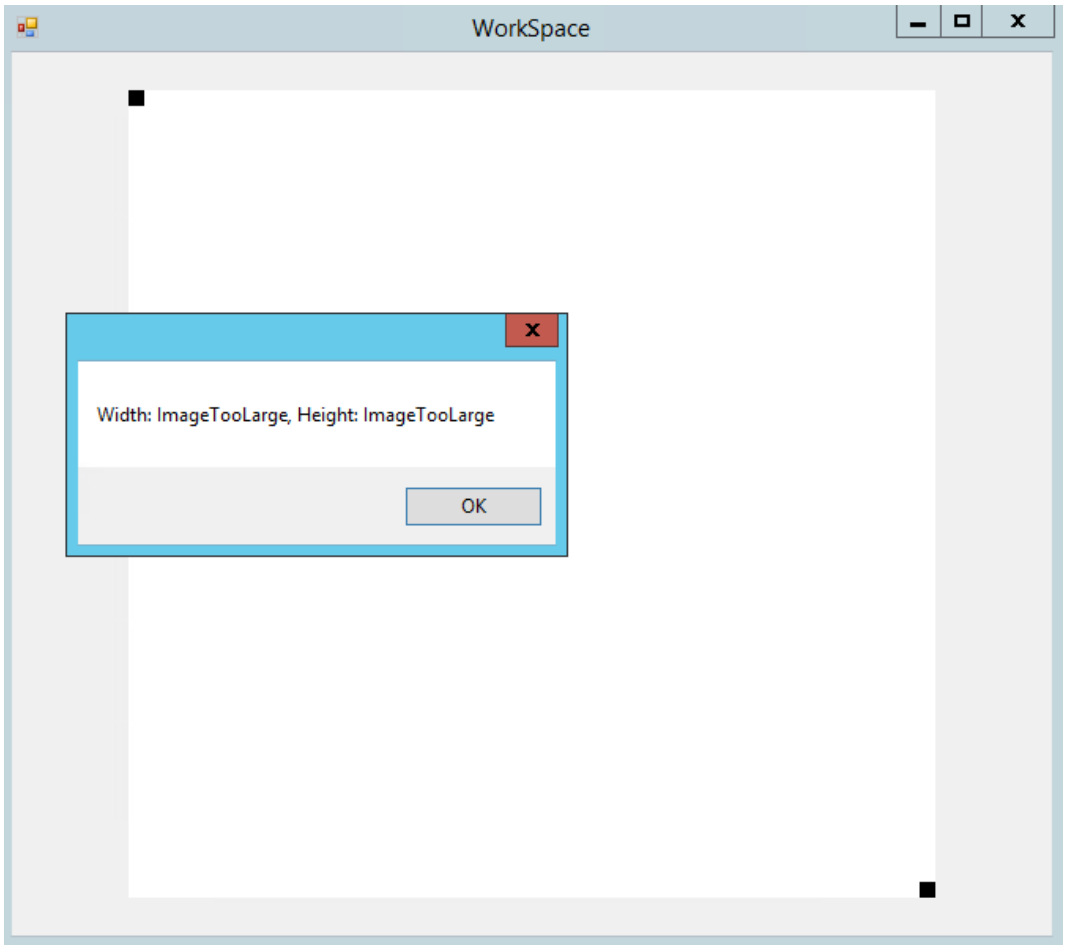
```
private EAxisMode CheckImageSize(EAxis axis) {
    int axisSize;
    int padding;
    switch (axis) {
        case EAxis.X:
            axisSize = DisplayRectangle.Width;
            padding = leftPadding + rightPadding;
            break;
        case EAxis.Y:
            axisSize = DisplayRectangle.Height;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

    DisplayRectangle displayImage = image.ToDisplayRectangle();

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

Alpha Testing CheckImageSize #1

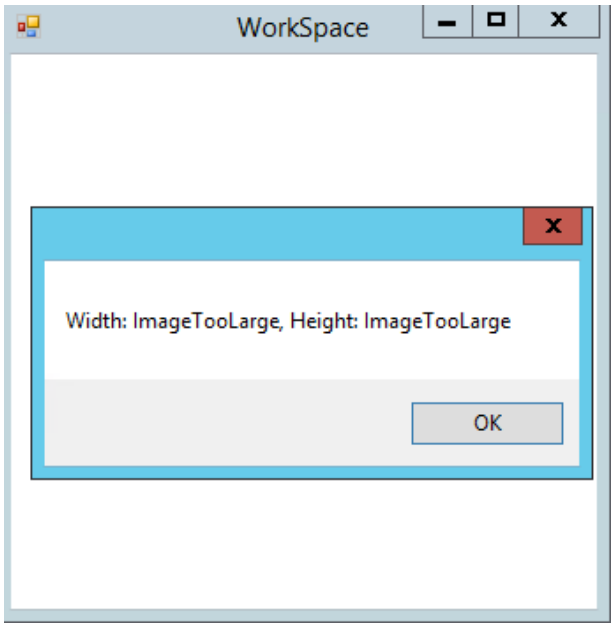
And can be **alpha tested** during development, using the following code:

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

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



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



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

```
DisplayRectangle displayImage = image.ToDisplayRectangle(
int imageAxisSize = displayImage.GetDisplaySize(axis);
        imageAxisSize    0
if (axisSize >= (displayImage.GetDisplaySize(axis) + padd
```

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

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

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

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

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

zoomSettings | SIMP.ZoomSettings

Base class	ValueType
centreLocation	null
zoom	0

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

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

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

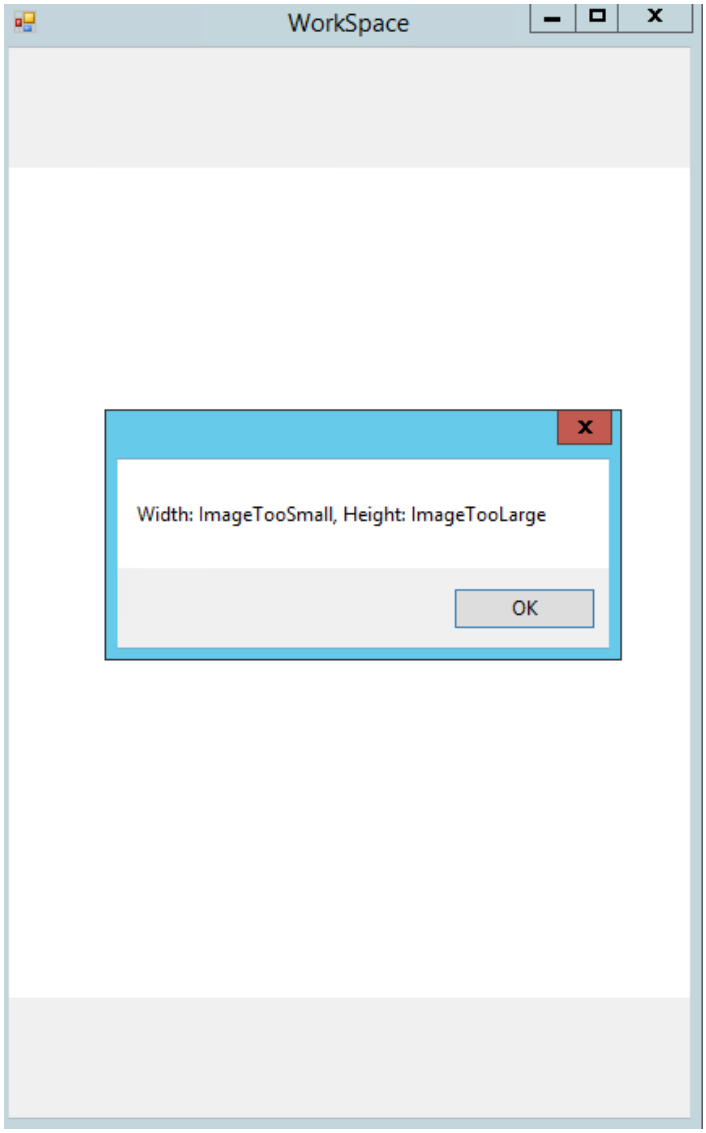
This error can be resolved by making sure that the explicit constructor is called:

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

```
this.zoomSettings = new ZoomSettings(new FilePoint(width/2,height/2));
```

Alpha Testing CheckImageSize #2

From testing CheckImageSize again, there is a second problem:



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

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

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

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

29/09/2019 Upgrading displayBox functionality

Implementing proper resizing and relocating for displayBox

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

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

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

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

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

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

Implementing new Image Requests

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

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

    // Relocates the picture box
    RelocateDisplayBox();

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

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

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

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

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

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

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

    return newImage;
}
```

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

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

1/10/2019 Conversions between pixel types

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

```
FilePointToDisplayPoint(filePoint) {
      displacementX = filePoint.X - centreFilePoint.X
      displacementY = filePoint.Y - centreFilePoint.Y

      displacementX = displacementX * zoom
      displacementY = displacementY * zoom

      newX = centreDisplayPoint.X + displacementX
      newY = centreDisplayPoint.Y + displacementY

      return new Point(newX, newY)
}
```

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

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

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

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

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

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

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

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

    return newImage;
}
```

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

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

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

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

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

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

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

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

2/10/2019 Testing Point conversions

File Point to Display Point testing

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

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

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

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

    outputPoint = outputPoint;

    return newImage;
```

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

Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
DisplayPoint(10,5)	1	(5,5)	Base class: Object displayX: 5 displayY: 5	The location of the centre of the image is returned
DisplayPoint(8,3)	2	(1,1)	Base class: Object displayX: 1 displayY: 1	The top-left is returned
DisplayPoint(12,7)	3	(9,9)	Base class: Object displayX: 9 displayY: 9	The bottom-right is returned
DisplayPoint(7,2)	4	(-1,-1)	Base class: Object displayX: -1 displayY: -1	A location off the edge of the image is returned
DisplayPoint(13,8)	5	(11,11)	Base class: Object displayX: 11 displayY: 11	A location off the edge of the image is returned
DisplayPoint(0,0)	6	(-15,-5)	Base class: Object displayX: -10 displayY: -5	The edge of the image is returned
DisplayPoint(19,9)	7	(23,13)	Base class: Object displayX: 23 displayY: 13	The edge of the image is returned
DisplayPoint(-1,-1)	8	Throws out of bounds error	Base class: Object displayX: -12 displayY: -7	No error is thrown

DisplayPoint(20,10)	9	Throws out of bounds error	Base class: Object displayX: 30 displayY: 15	No error is thrown

Fixing Error #8 & #9

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

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

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)	Base class: Object fileX: 10 fileY: 5	The centre location is returned
FilePoint(1,1)	2	(8,3)	Base class: Object fileX: 8 fileY: 3	The correct point is returned
FilePoint(12,7)	3	(9,9)	Base class: Object fileX: 13 fileY: 6	The correct point is returned
FilePoint(-1,-1)	4	(7,2)	Base class: Object fileX: 7 fileY: 2	The correct point is returned
FilePoint(11,11)	5	(13,8)	Base class: Object fileX: 13 fileY: 8	The correct point is returned
FilePoint(-15,-5)	6	(0,0)	Base class: Object fileX: 0 fileY: 0	The correct point is returned
FilePoint(23,13)	7	(19,9)	Base class: Object fileX: 19 fileY: 9	The correct point is returned
FilePoint(6,6)	8	(10,5)	Base class: Object fileX: 10 fileY: 5	The point is rounded up
FilePoint(4,4)	9	(9,4)	Base class: Object fileX: 10 fileY: 5	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.

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

displacementX	-1

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

displacementX	0

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

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

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

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

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

Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
FilePoint(4,4)	9	(9,4)	Base class: Object fileX: 9 fileY: 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

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

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

Algorithm 2.8C – Finding Green Pixels

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

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

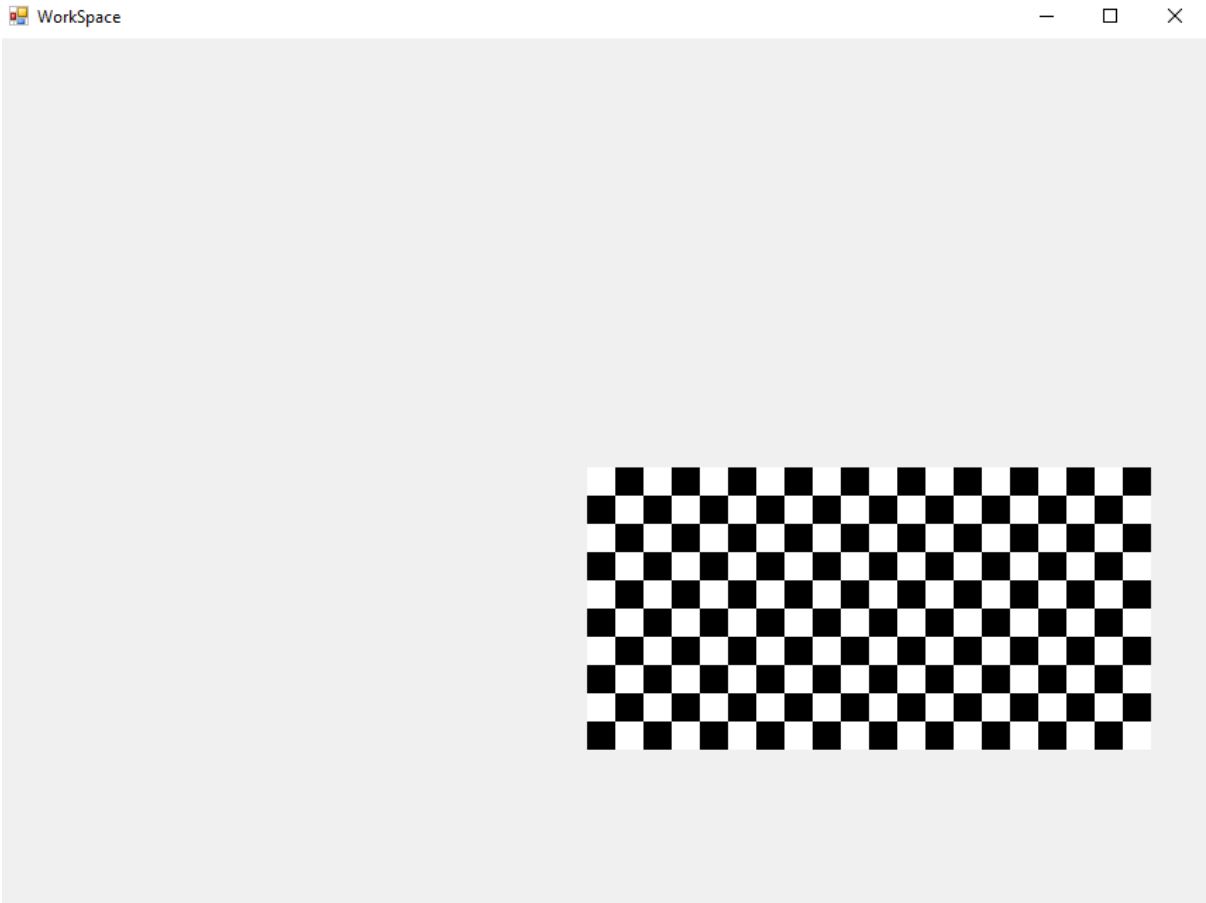
Where the comment will be replaced with the code necessary to draw the zoomed part

Algorithm 2.9 – Draw Green Pixels

The above code has been upgraded to include drawing functionality:

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

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

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

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

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

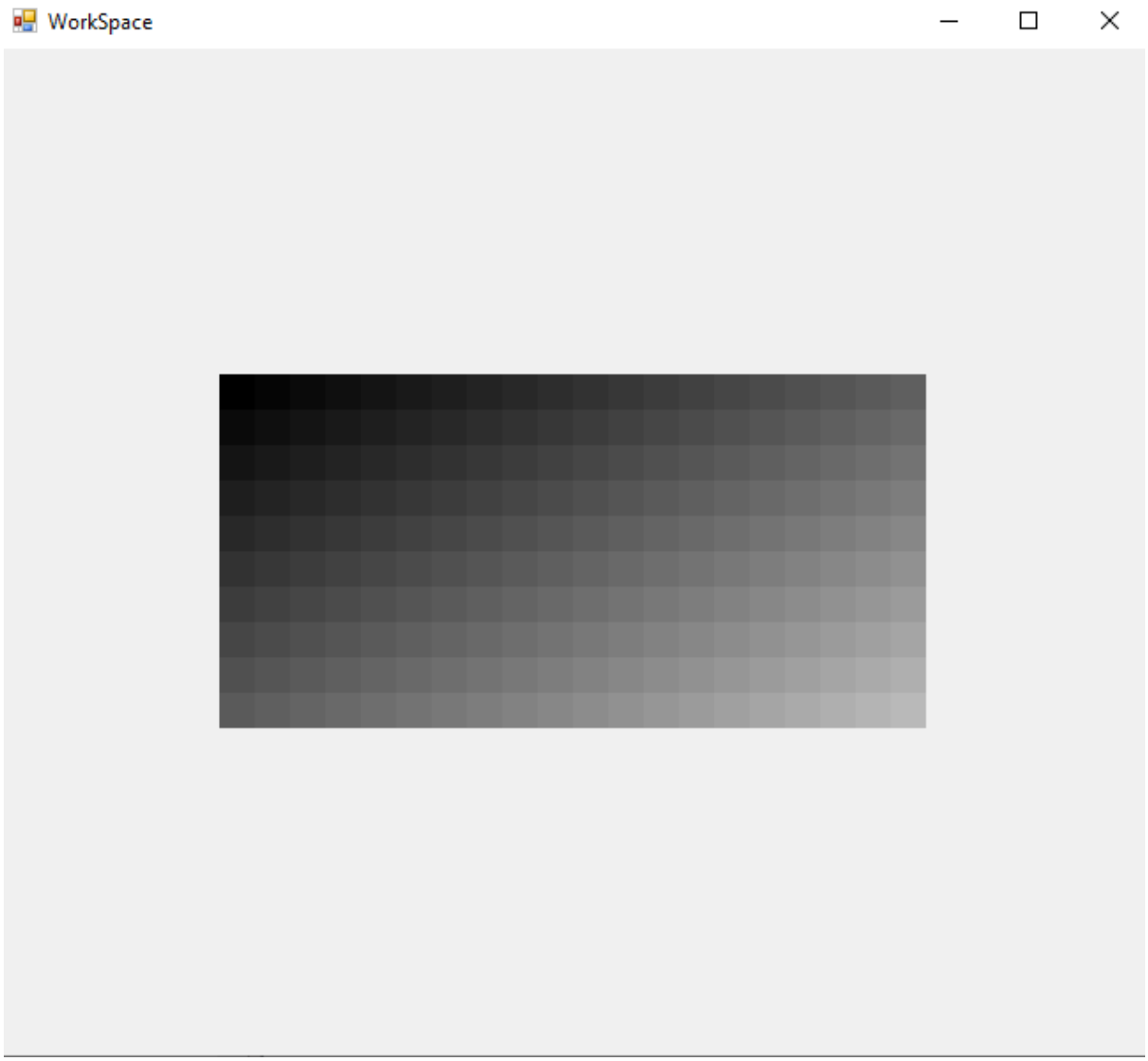


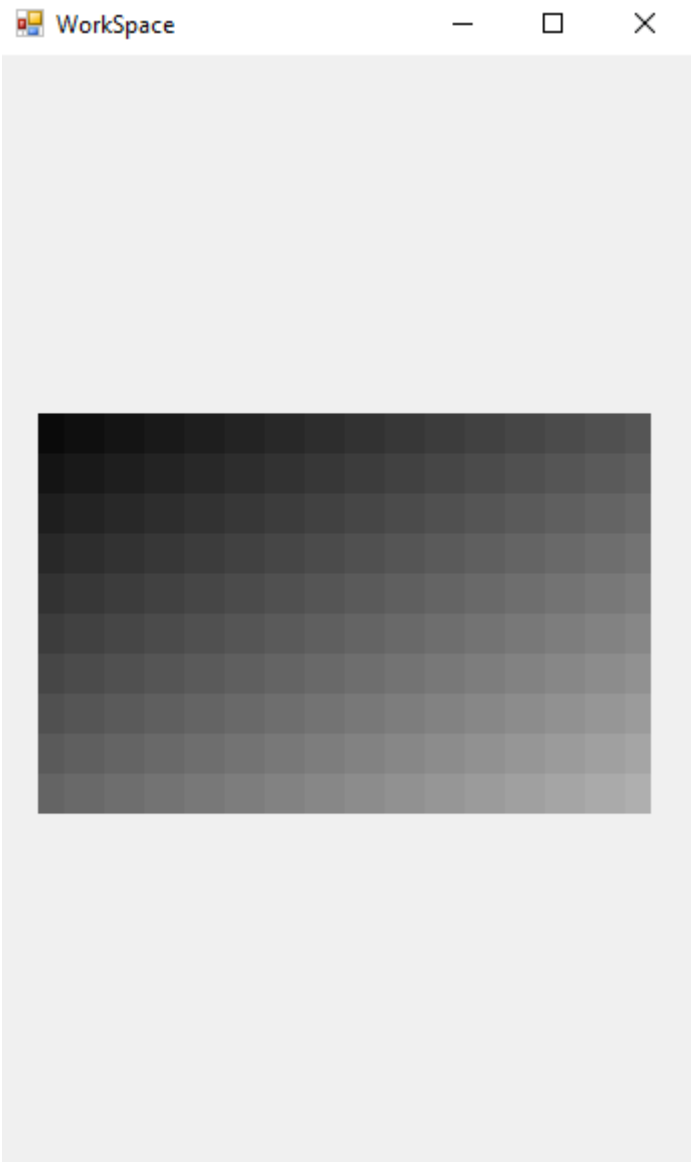
The cause of this was simple, fileWidth was erroneously used instead of displayWidth:

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

Changing this to displayWidth resolves the error.

SIMP can now display images:

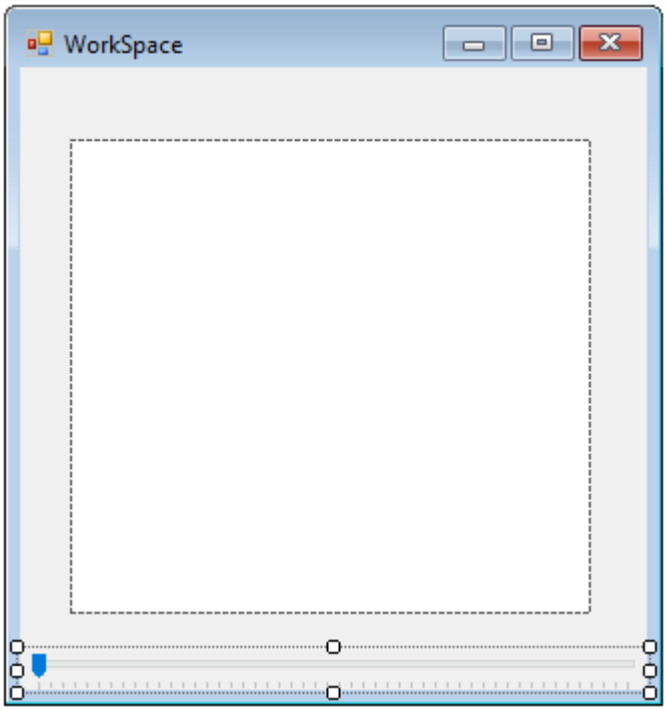




Some example of SIMP displaying images, even when the display form is much smaller than the image

Adding Zoom Bar to GUI

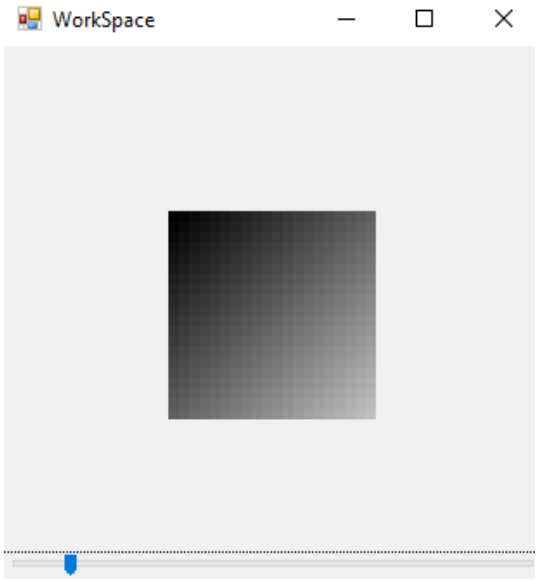
The zoom bar can be added quite simply, filling the border area at the bottom.



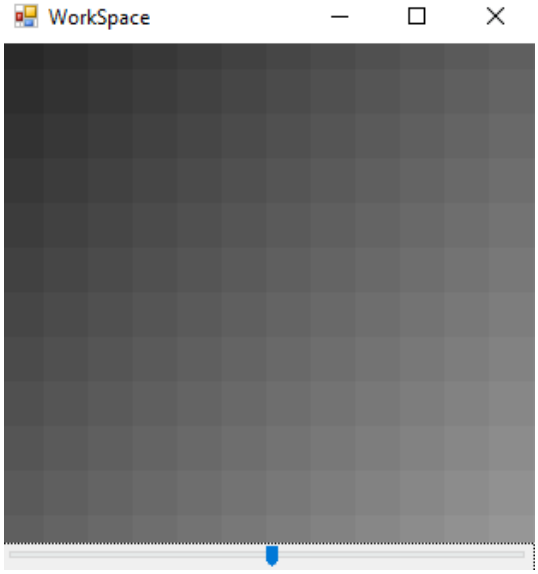
Its value can also be interpreted quite simply. Its minimum and maximum has been set to 1 and 50 respectively. When its value is changed, the image's zoom is updated and the box is redrawn.

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

    UpdateDisplayBox(true);
}
```

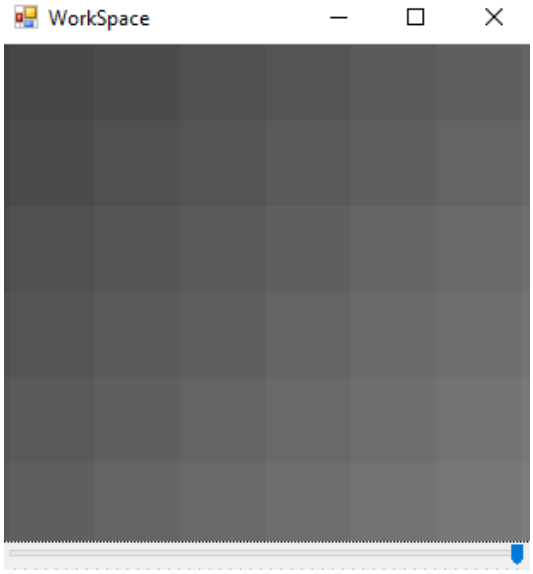
This means the zoom level can be changed using the bar:



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.

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

The completed check for this looks like:

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

Algorithm 2.11A Determining Crosses in Axis

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

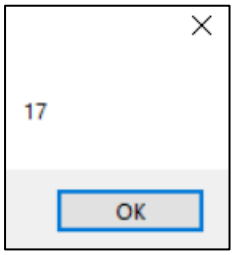
```
DetermineCrossesInAxis(Axis axis) {
      MissingCrossesInAxis = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)*2
      MaxCrossesInAxis = image.getSizeInAxis(axis)
      return MaxCrossesInAxis – MissingCrossesInAxis
}
```

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

When testing the code with the following inputs:

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

The output is:



Which is what was expected in design.

Algorithm 2.11B Setting Bar Size

The algorithm to resize the bars has been implemented, in accordance to Algorithm 2.11B:

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

```
public void SetBarValues(ScrollBar barHorizontal, ScrollBar barVertical, int width, int height) {
    // determines how many missing centre locations there are
    int missingCrossexXAxis = MissingCrossesInAxis(width,fileWidth);
    int missingCrossexYAxis = MissingCrossesInAxis(height,fileHeight);

    // updates horizontal bar
    barHorizontal.LargeChange = missingCrossexXAxis;
    barHorizontal.Maximum = fileWidth;

    // updates vertical bar
    barVertical.LargeChange = missingCrossexYAxis;
    barVertical.Maximum = fileHeight;
}
```

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

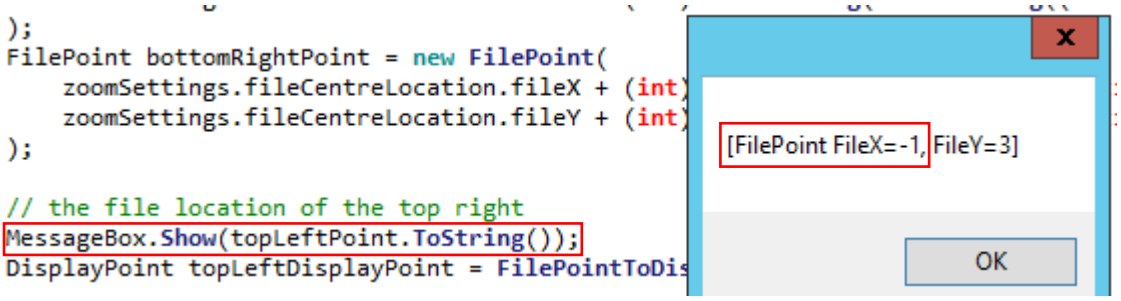
Algorithm 2.11C Determining Centre Location from Bar Value

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

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

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

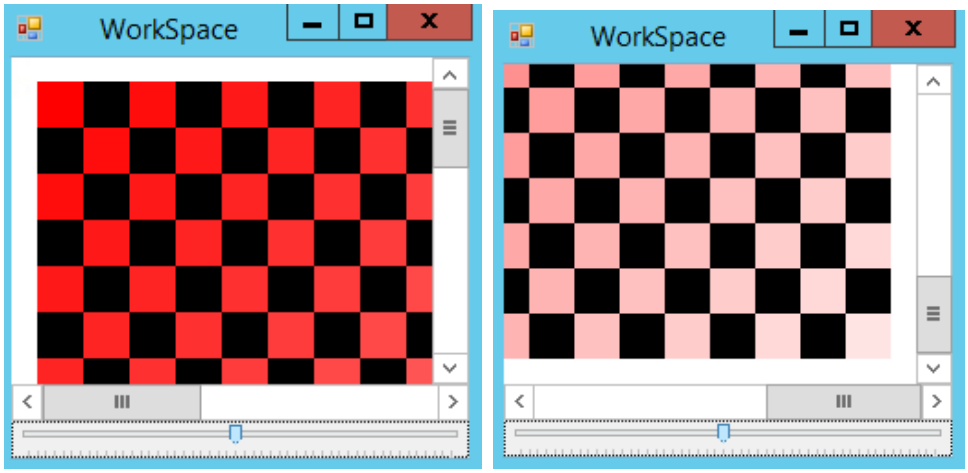
However, when attempting to change the value of a bar, an IndexOutOfBounds error is thrown. The cause of the error is found to be, when rendering, the FileX could be taken out of bounds



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:

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

    if (filePoint.fileY < 0) {
        filePoint.fileY = 0;
    }
    if (filePoint.fileY >= fileHeight) {
        filePoint.fileY = fileHeight-1;
    }
}
```

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



05/10/2019 Remaining Bar code

Algorithm 2.11D Determining Bar Value from Centre Location

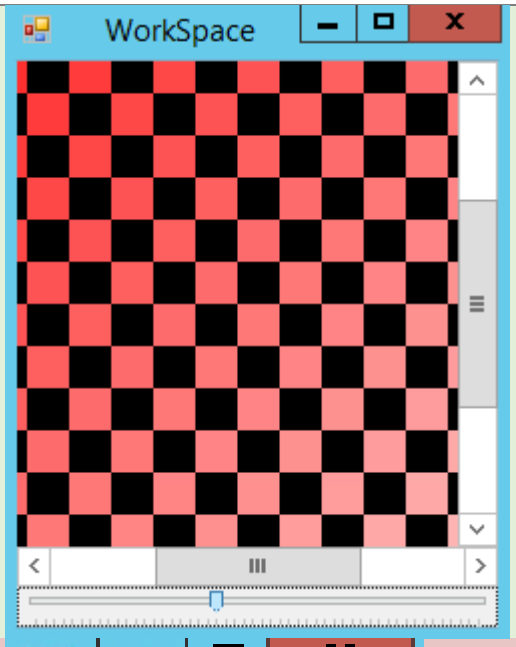
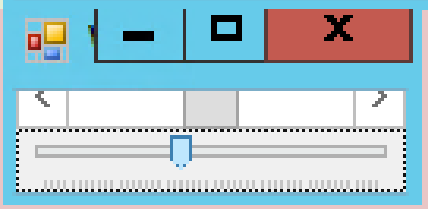
The code for determining the Bar Value from a given Centre Location has been implemented, in accordance to Algorithm 2.11D:

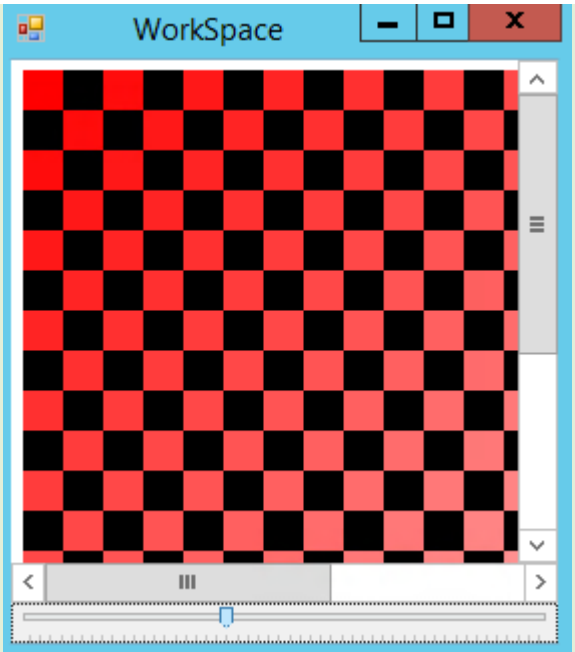
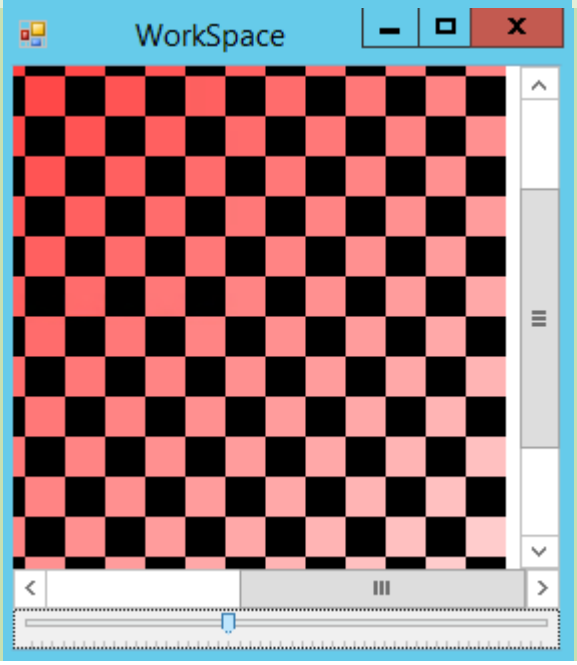
```
SetBarFromCentreLocation(progressBar bar, Axis axis) {
      firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)
      bar.value = centreLocation.getSizeInAxis(axis) - firstCrossPosition
}
```

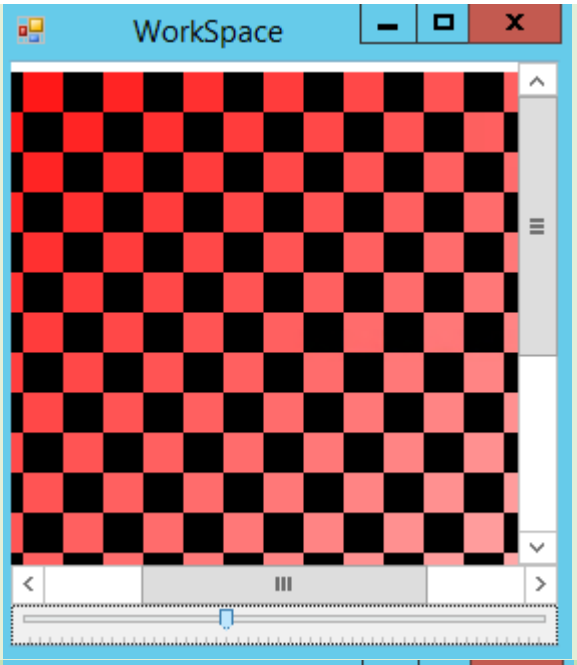
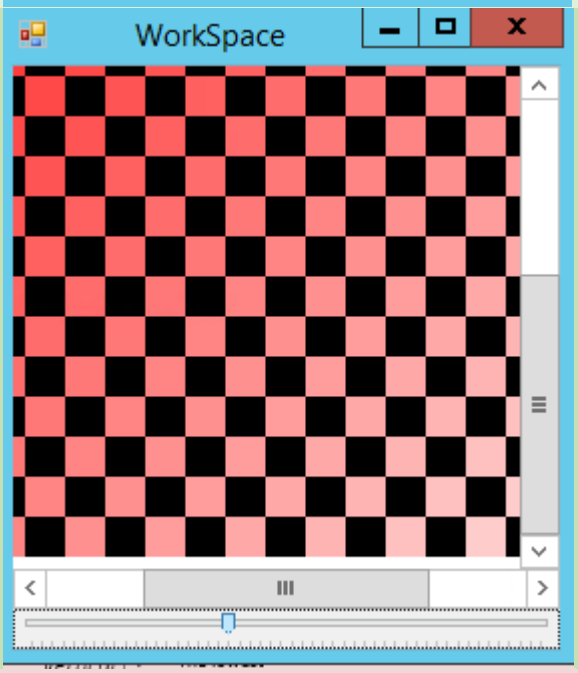
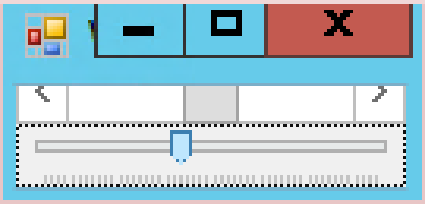
Again I can't add the screenshot, do this later please

Unit Test 2.11 #1

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

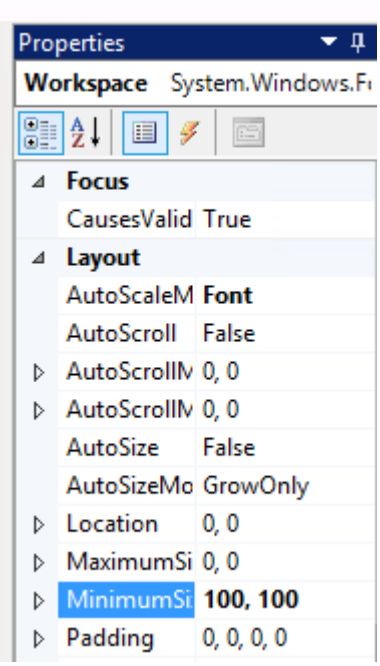
Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
Form is resized to half the size of image	1	Bar is half size of bounds and in its centre position		The bars are displaying the correct information
Form is resized to its smallest position	2	Bar is small but still central.	 `System.ArgumentException: 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:

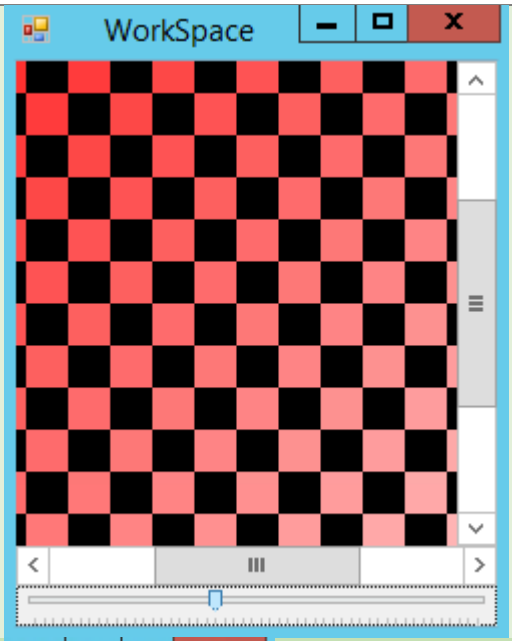
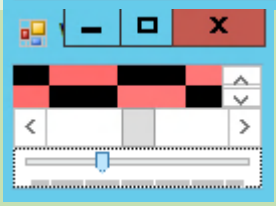
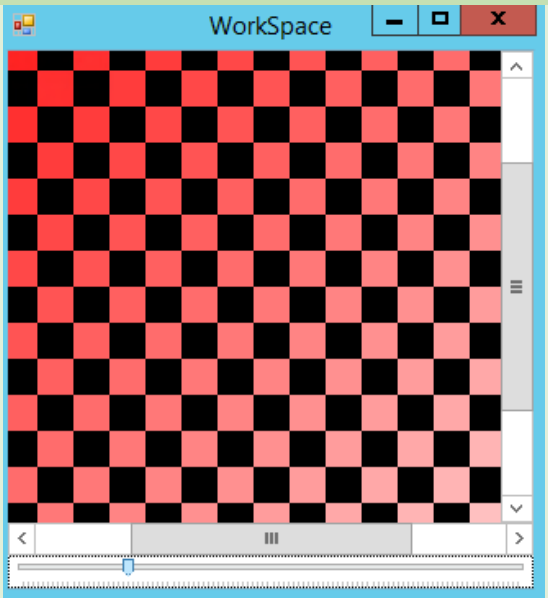


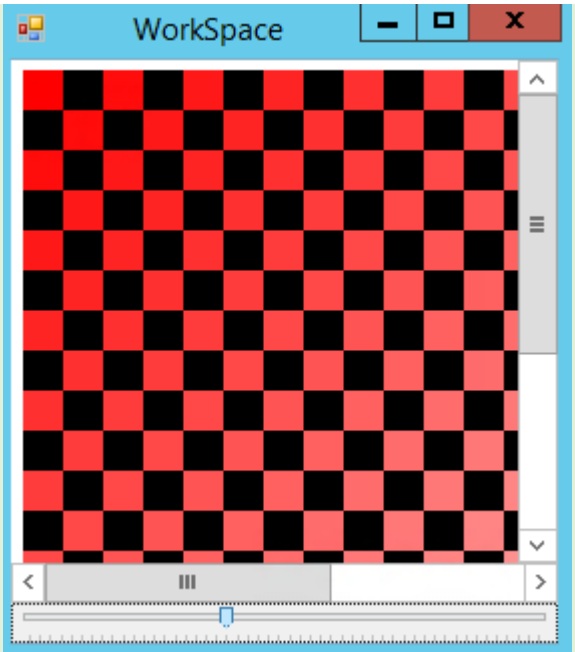
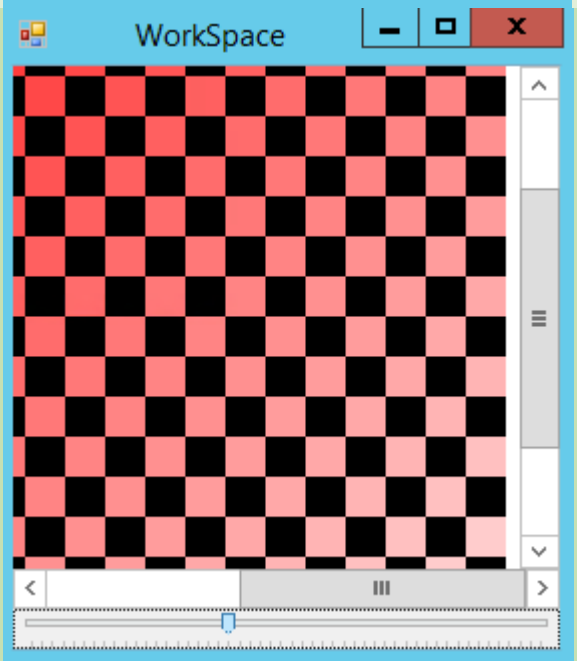
Fixing Error #8

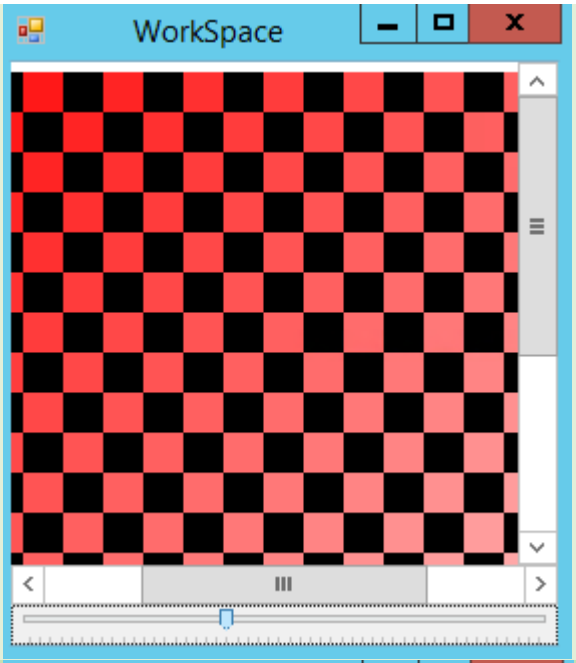
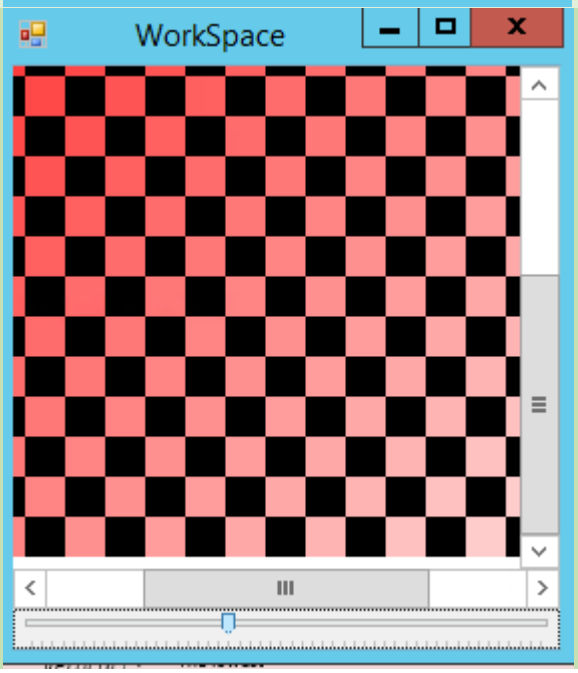
This error has a more problematic cause. It stems from the fact that when zooming in, changing the centre and then zooming out, the centre location can be placed in invalid places. In order to fix this, when displaying a new image the centre location must be checked to ensure it is still valid:

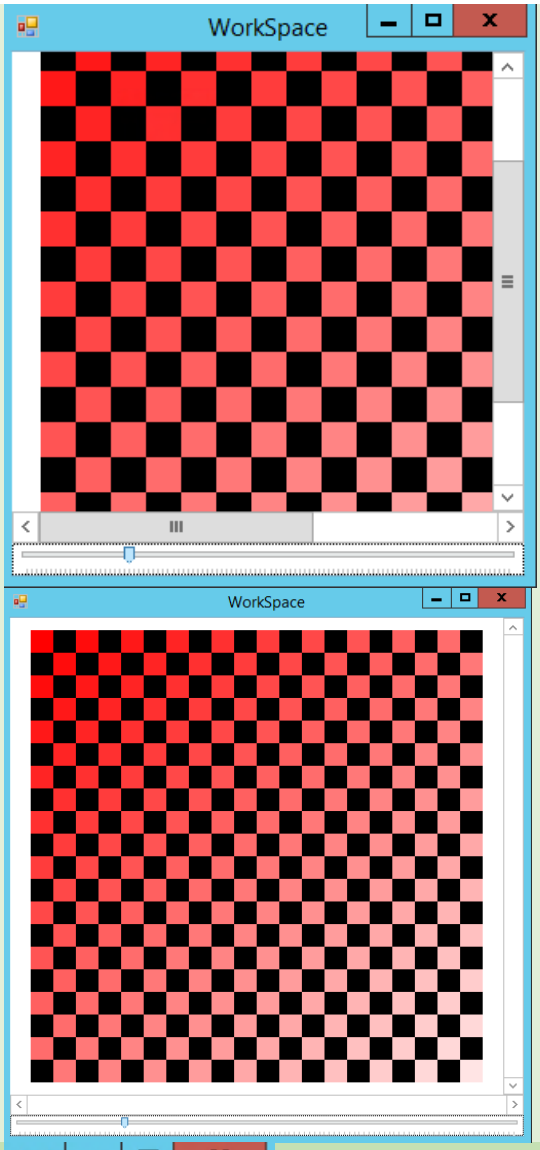
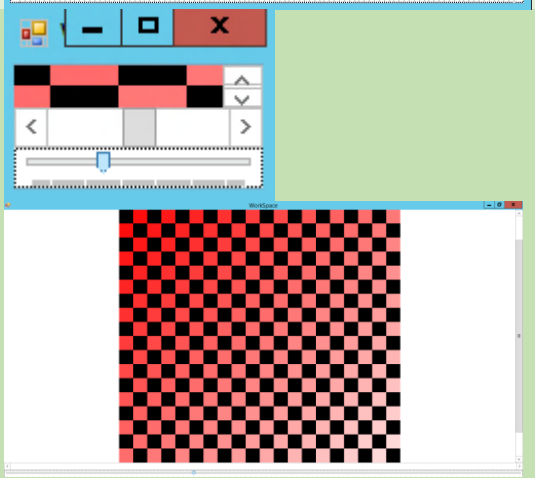
```
private void ClampCentreLocation(int width, int height) {
    int firstCrossInXAxis = GetFirstCrossPosition(width);
    int firstCrossInYAxis = GetFirstCrossPosition(height);

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

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

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

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

Unit Test 2.11 #2

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

```
void PictureBox1Click(object sender, EventArgs e)
{
    MouseEventArgs me = (MouseEventArgs)e;

    MessageBox.Show(String.Format("X: {0} Y: {1}", me.X,me.Y));
}
```

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

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

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

Algorithm 2.14 & Algorithm 2.15

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

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

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

And then adding a call from WorkSpace:

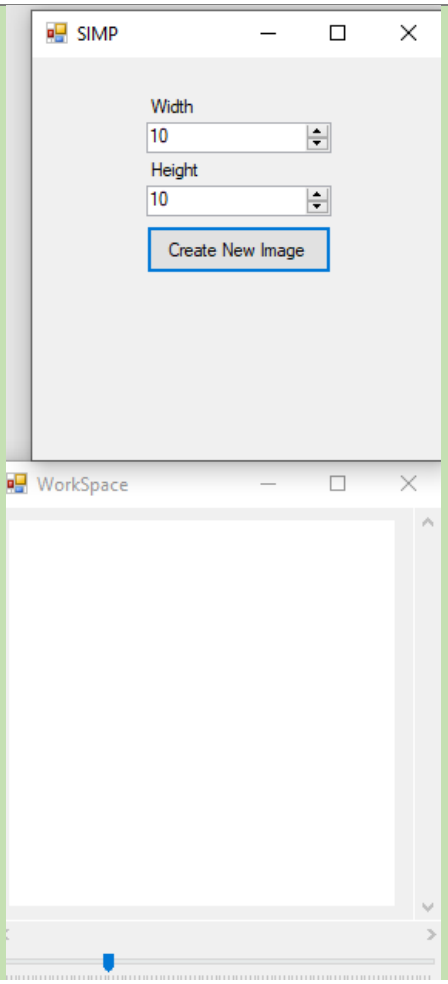
```
void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) {
        DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
        image.SetPixel(clickLocation,Color.Black);
        UpdateDisplayBox(true);
    }
}

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

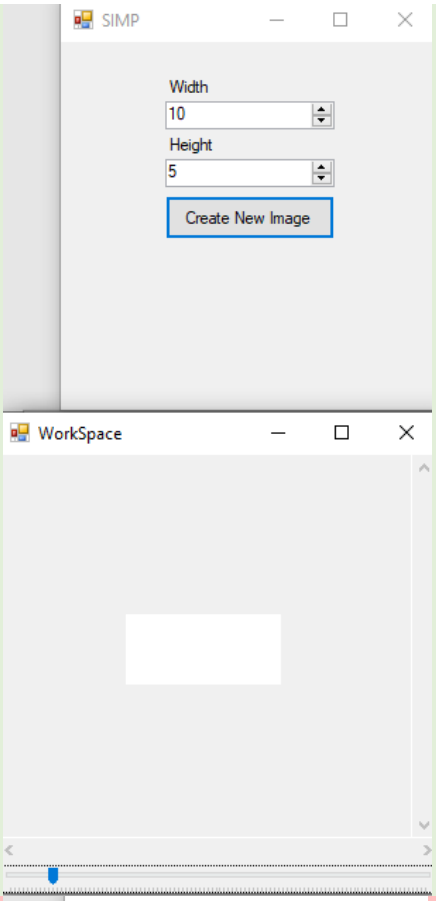
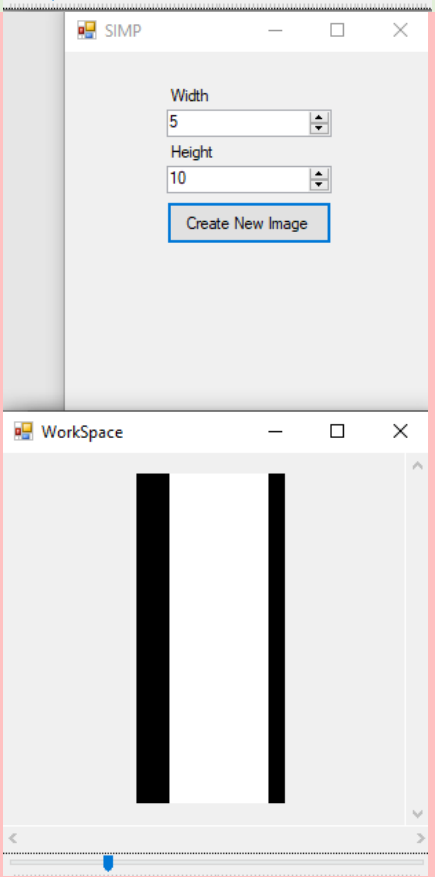
This means that the **image can now be changed at runtime**.

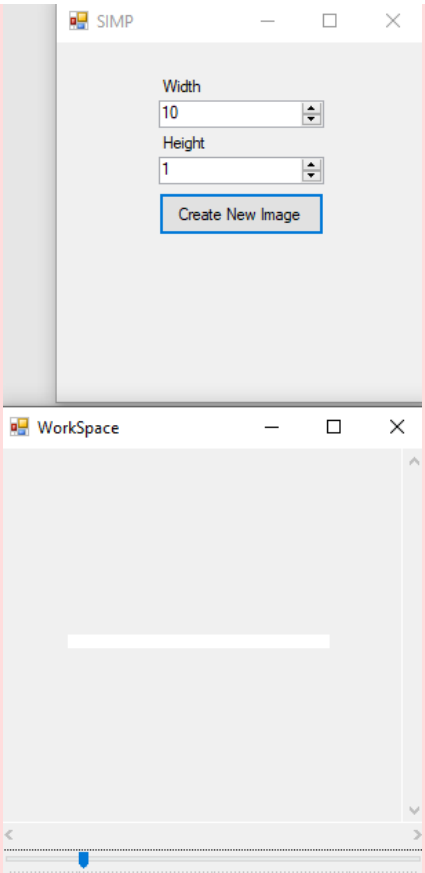
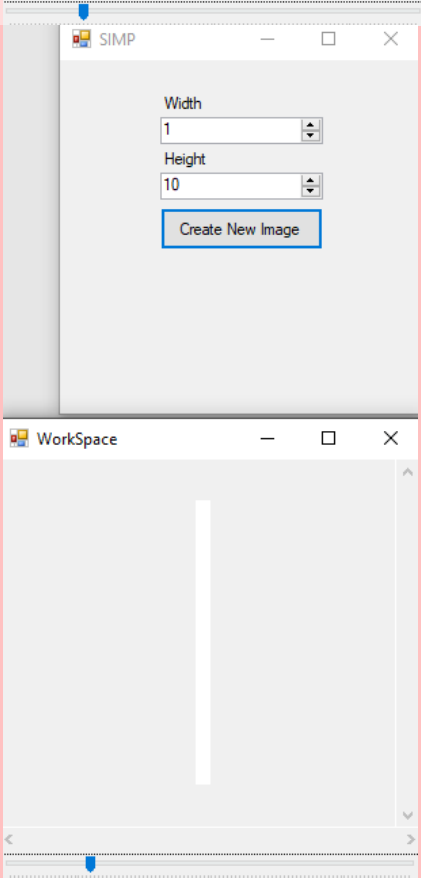
2.3 Testing

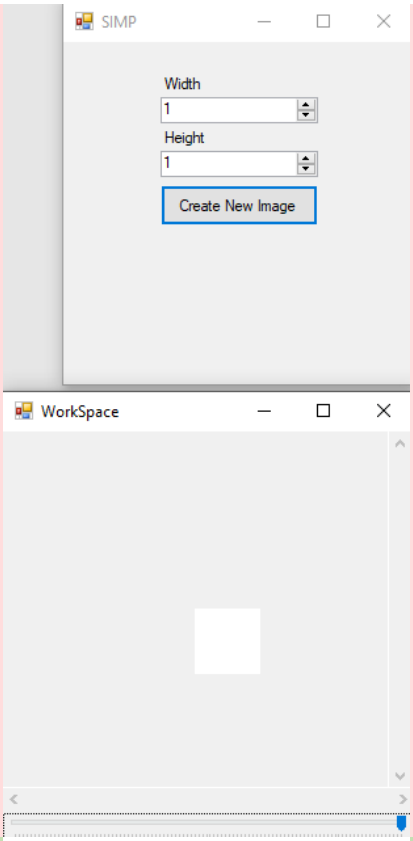
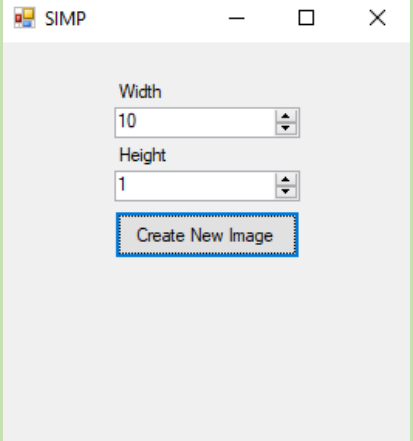
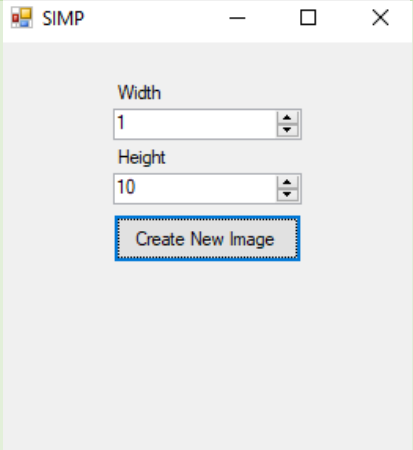
2.3.1 Full Testing

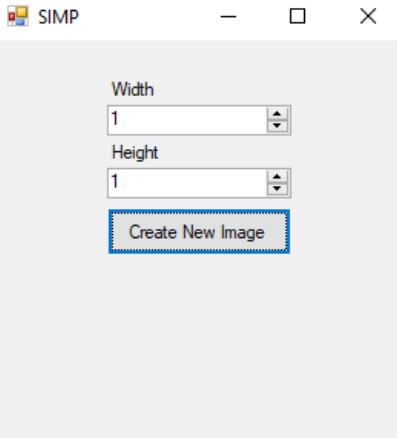
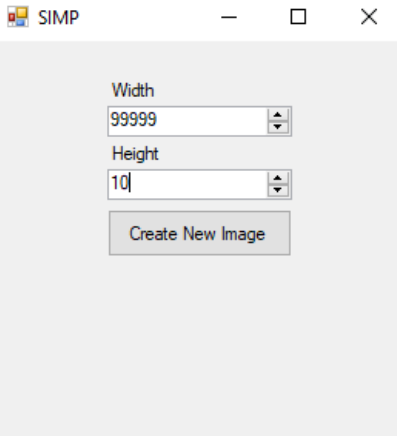
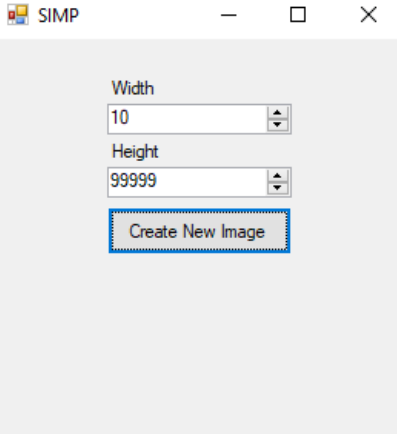
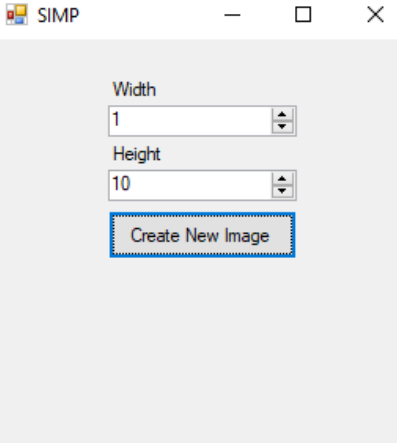
Initial Testing Table

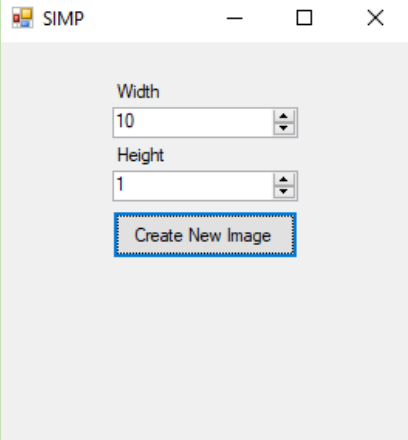
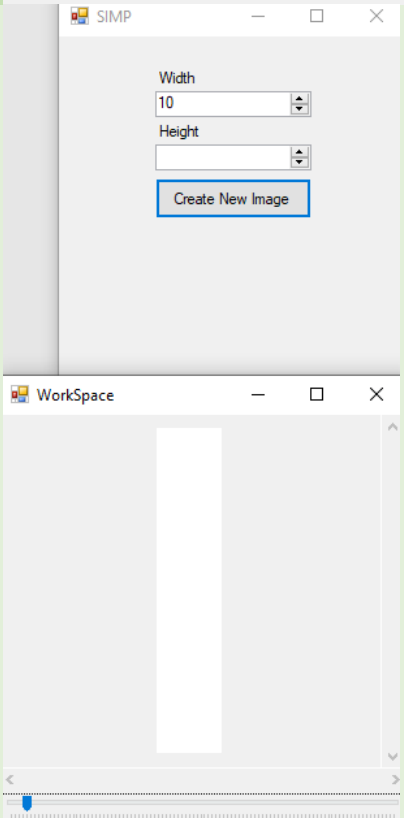
Test	*ID*	*Expected Result*	*Actual Result*	*Comment*
new Image(10,10)	1	An image of size 10fpx, 10fpx		A 10px by 10px image is successfully created.

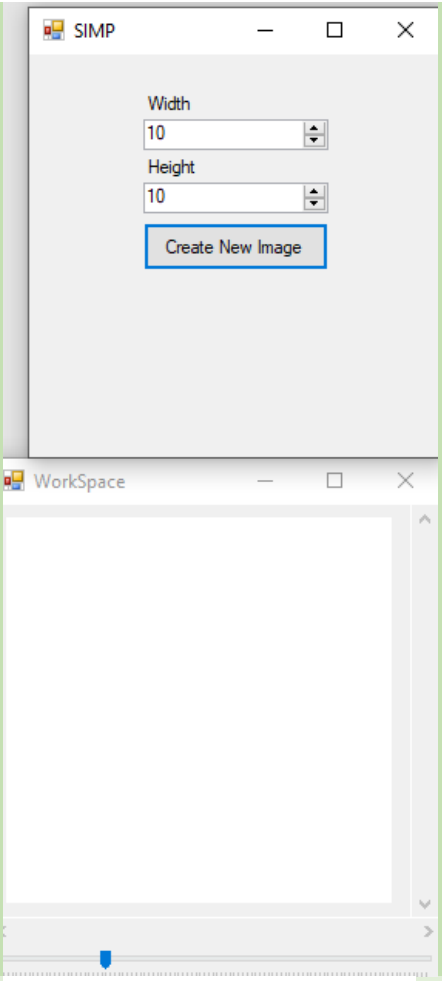
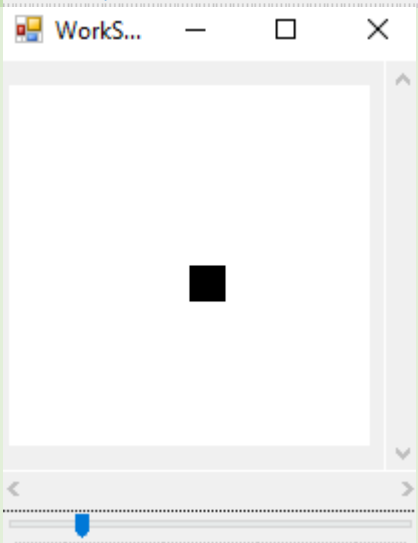
new Image(10,5)	2	A 10fpx, 5fpx image		The slightly thinner image is successfully created
new Image(5,10)	3	A 5fpx, 10fpx image		A tall image is successfully created. However there are problems drawing on the odd edge of this sort of image, as only half of the edge is displayed.

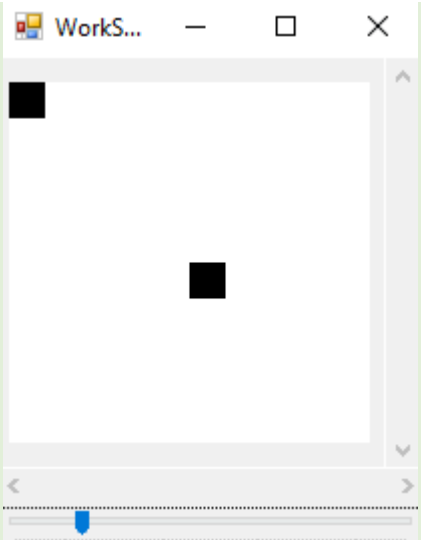
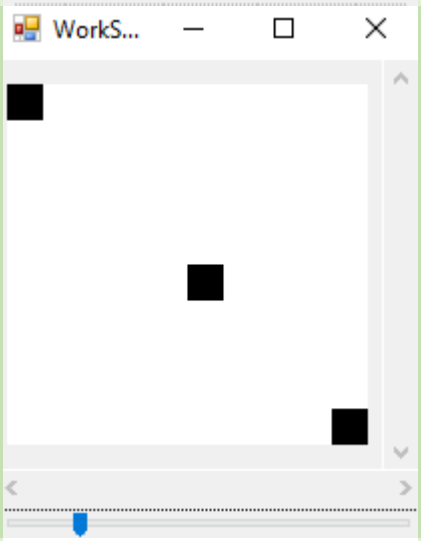
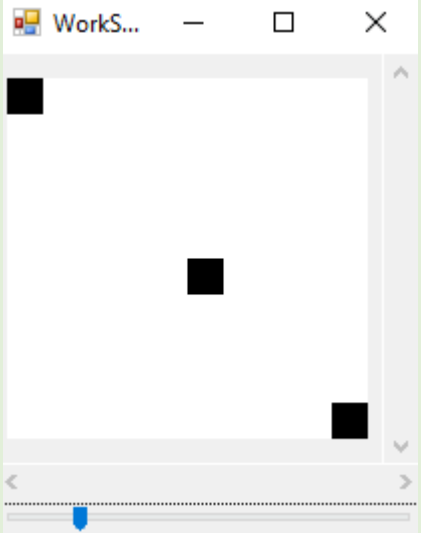
new Image(10,1)	4	A 10fpx, 1fpx image		An image is created, but is much too thin (only displaying half of a pixel).
new Image(1,10)	5	A 1fpx, 10fpx image		An image is created, but is much too thin as only a half pixel is displayed

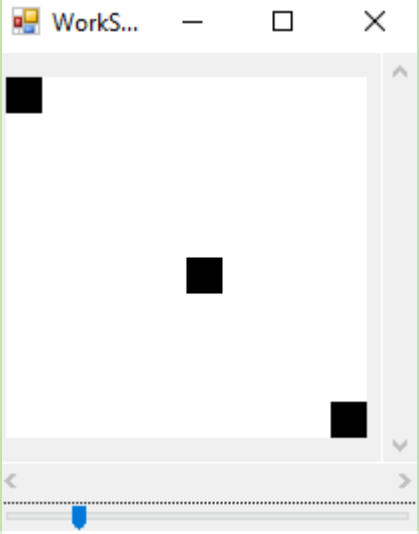
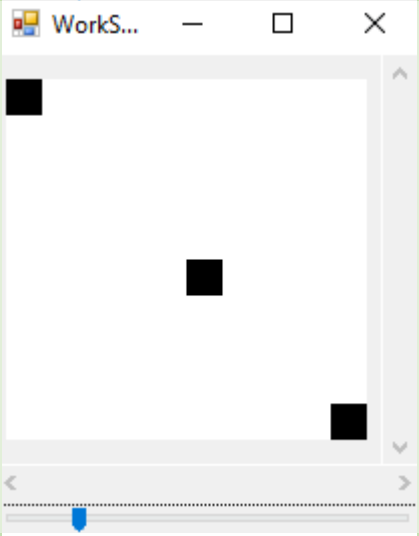
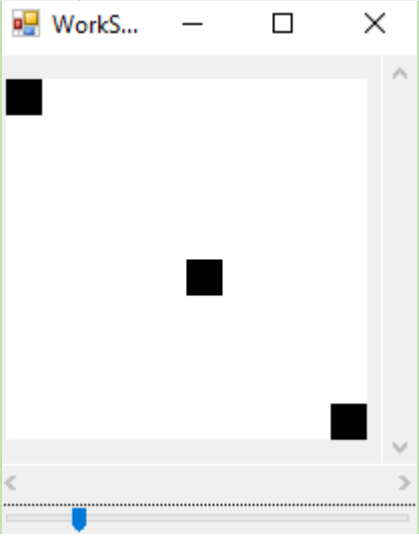
new Image(1,1)	6	A 1fpx, 1fpx image		A very small picture is created, but it is not centrally aligned in the image.
new Image(10,0)	7	The parameters are rejected and no image is created		A height of 0 is not accepted, and autocorrects to 1
new Image(0,10)	8	The parameters are rejected and no image is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

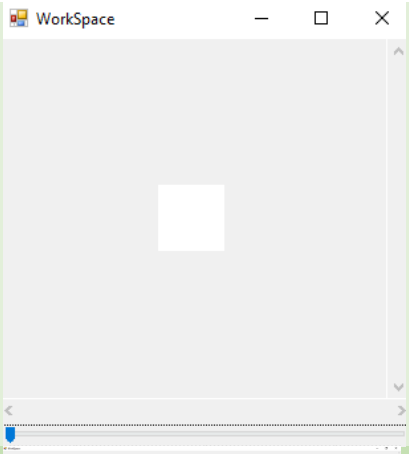
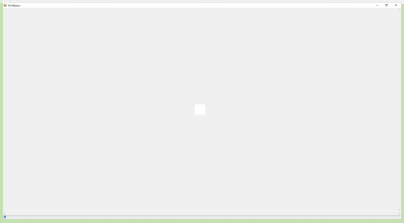
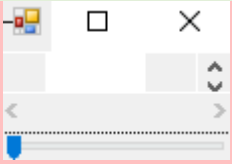
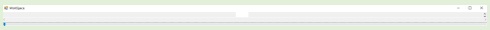
new Image(0,0)	9	The parameters are rejected and no image is created		Both boxes are checked independent of each other so this poses no new issue
new Image(100000,10)	10	The parameters are rejected and no image is created		The width is autocorrected back down to the allowed maximum
new Image(10,100000)	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)

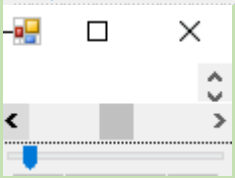
new Image("10","10")	15	The parameters are rejected and no image is created		It is impossible to create an image from text
SetPixel(5,5,Black)	16	A pixel near the middle of the image is set to black		
SetPixel(5,5,Green)	17	A pixel near the middle of the image is set to yellow	This is currently impossible, however previous tests indicate that colour displaying is possible.	

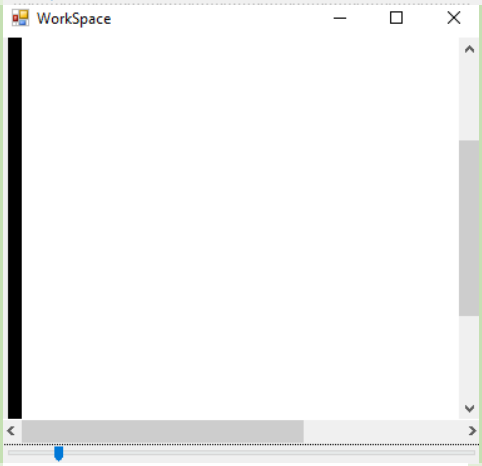
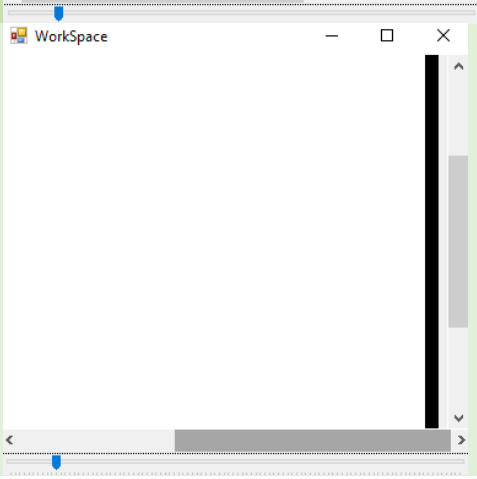
SetPixel(0,0,Black)	18	A pixel in the top-left corner is set to black		A pixel at the top left is set
SetPixel(9,9,Black)	19	A pixel in the bottom-right corner is set to black		
SetPixel(-1,0,Black)	20	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds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

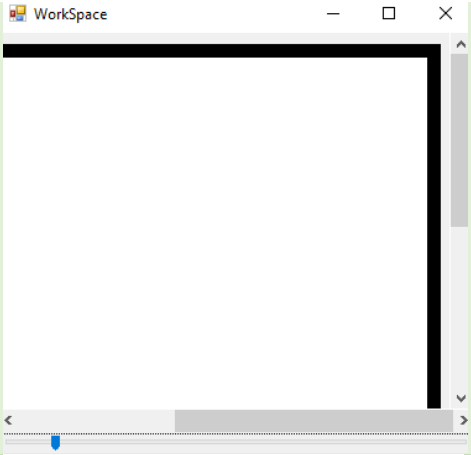
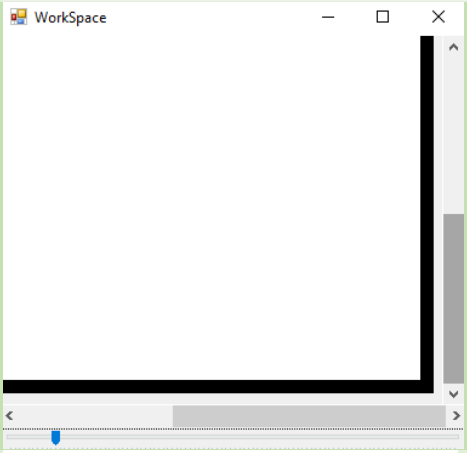
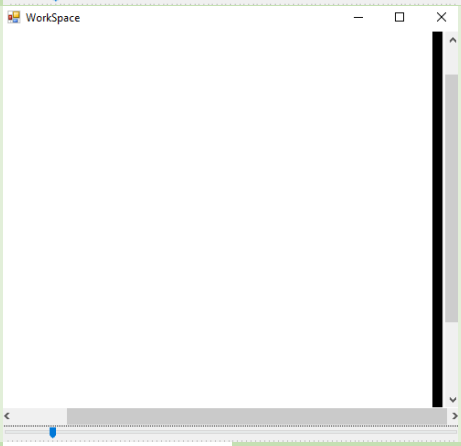
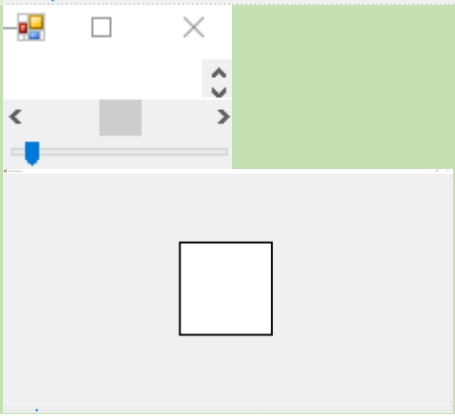
SetPixel(0,-1,Black)	21	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored
SetPixel(10,0,Black)	22	No pixel is set as it is out of bounds		There is no change as attempts to set pixels out of bounds are ignored
SetPixel(0,10,Black)	23	No pixel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

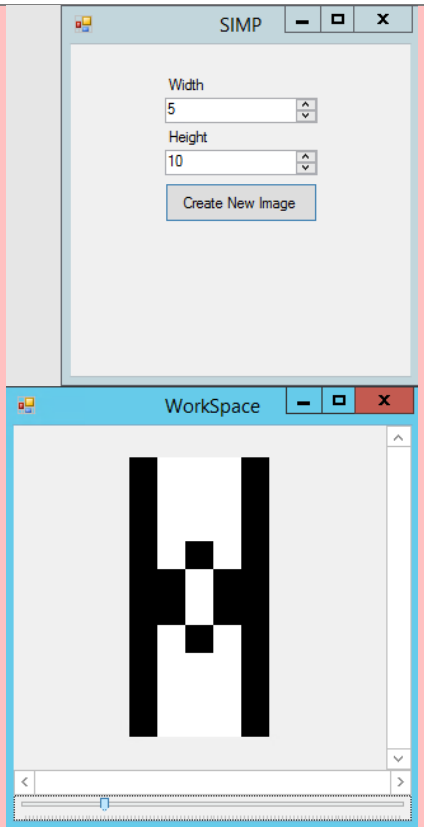
The error arises from the fact that on an odd-dimensioned image, when the display zoom centre is calculated, the width and height are simply halved:

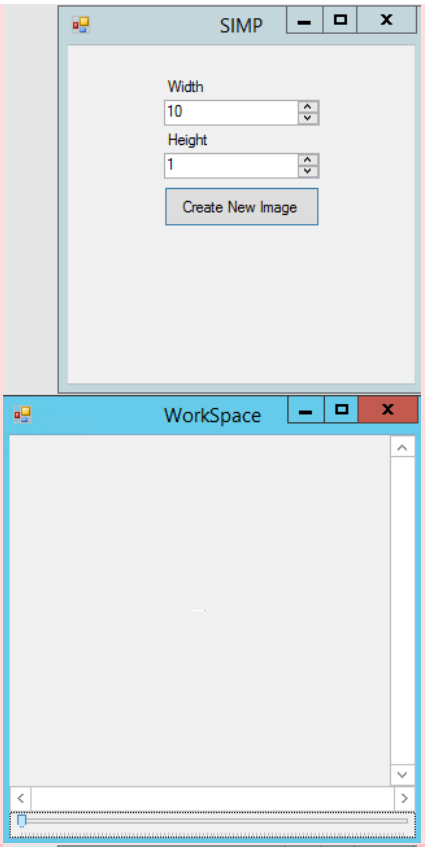
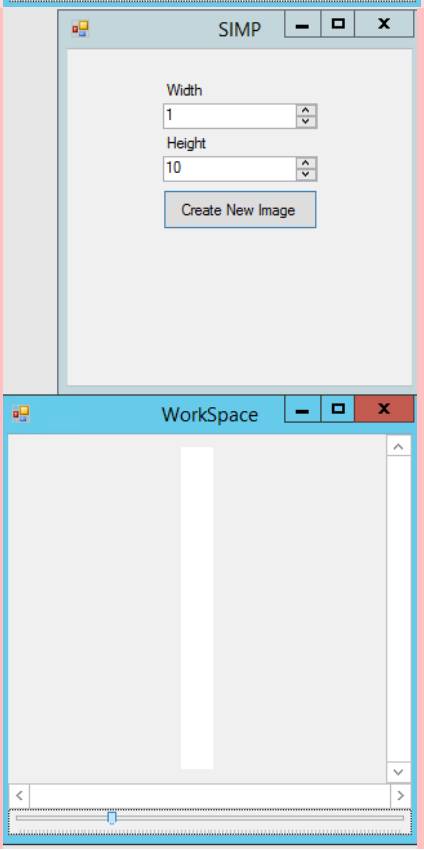
```
zoomSettings.displayCentreLocation = new DisplayPoint(width/2,height/2);
```

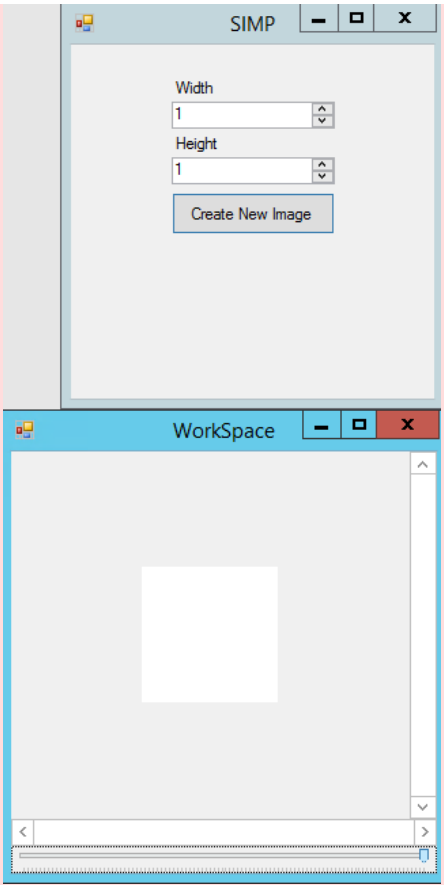
However this would lead to (on odd numbered images) a centre location that didn't align with any file pixels. In order to solve this a division and multiplication system has been implemented to round the centre location to the nearest file pixel:

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

So, performing the tests again:

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

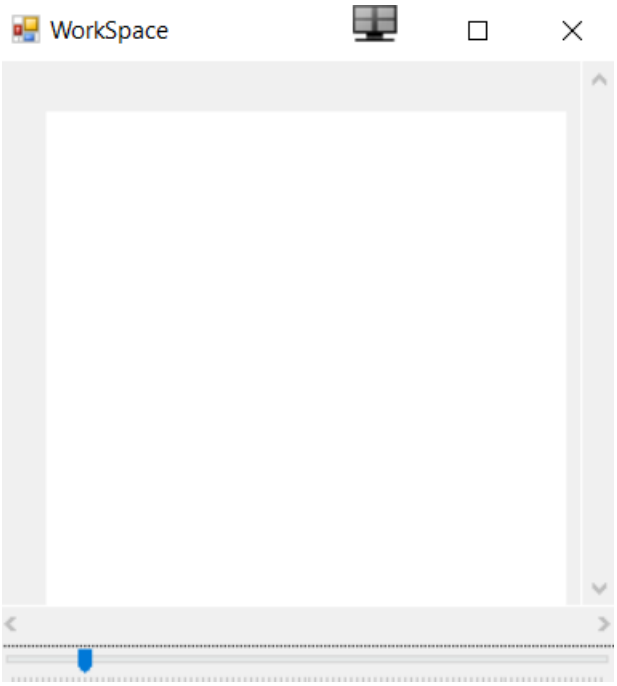
This now means that SIMP has passed **Alpha Testing**. It will now be handed to the Stakeholders for their initial thoughts and **Beta Testing**.

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:

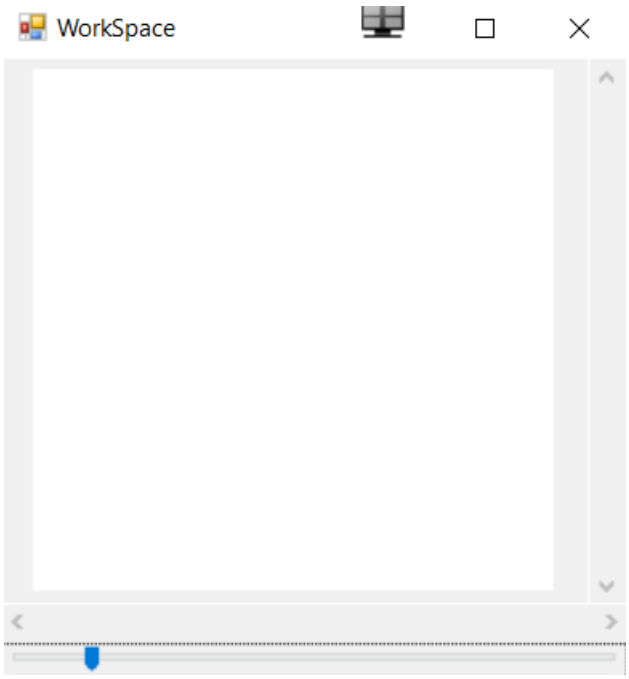
```
private void Constructor(int width, int height) {
    image = new SIMP.Image(width,height,this);

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

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

    // Updates the form
    UpdateDisplayBox(true);

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

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

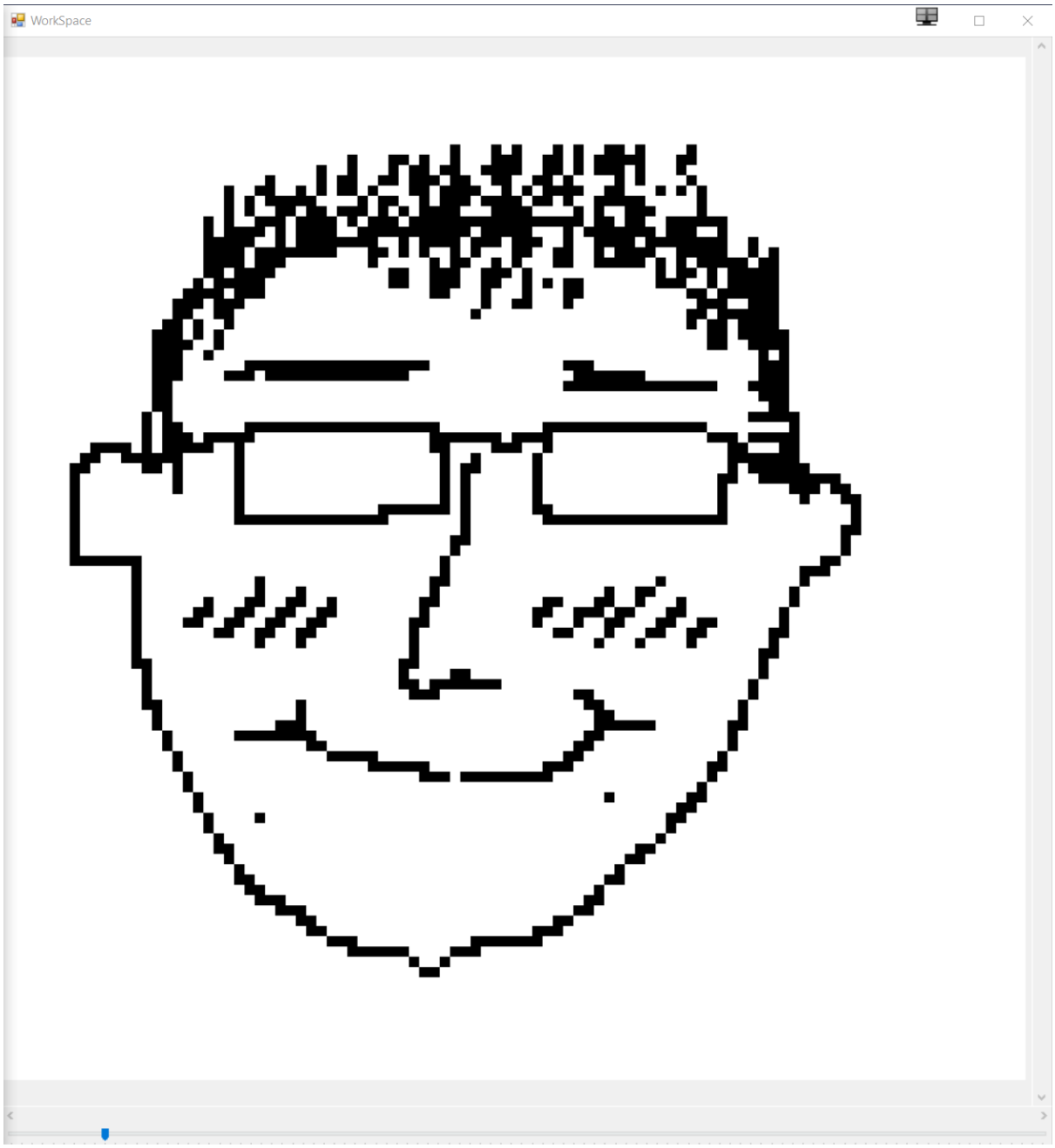
A simple code re-order resolves this:

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

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

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

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

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

So the program now starts correctly.



However, he was able draw an image



Scrolling Upgrade

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

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

However, a first attempt to code this proved unsuccessful:

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

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

This can be solved by using the MouseWheel event:

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

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

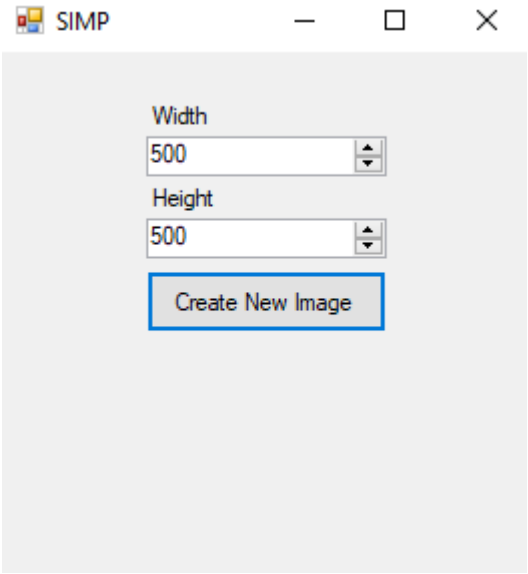
```
void WorkspaceKeyDown(object sender, KeyEventArgs e)
{
    if (e.Shift) {
        shiftPressed = true;
    }
    if (e.Control) {
        controlPressed = true;
    }
}

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

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

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

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

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

Permitting Custom Image Sizes

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



Optimising Rectangle Drawing

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

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

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

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

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

This then means the drawing code can be implemented as:

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

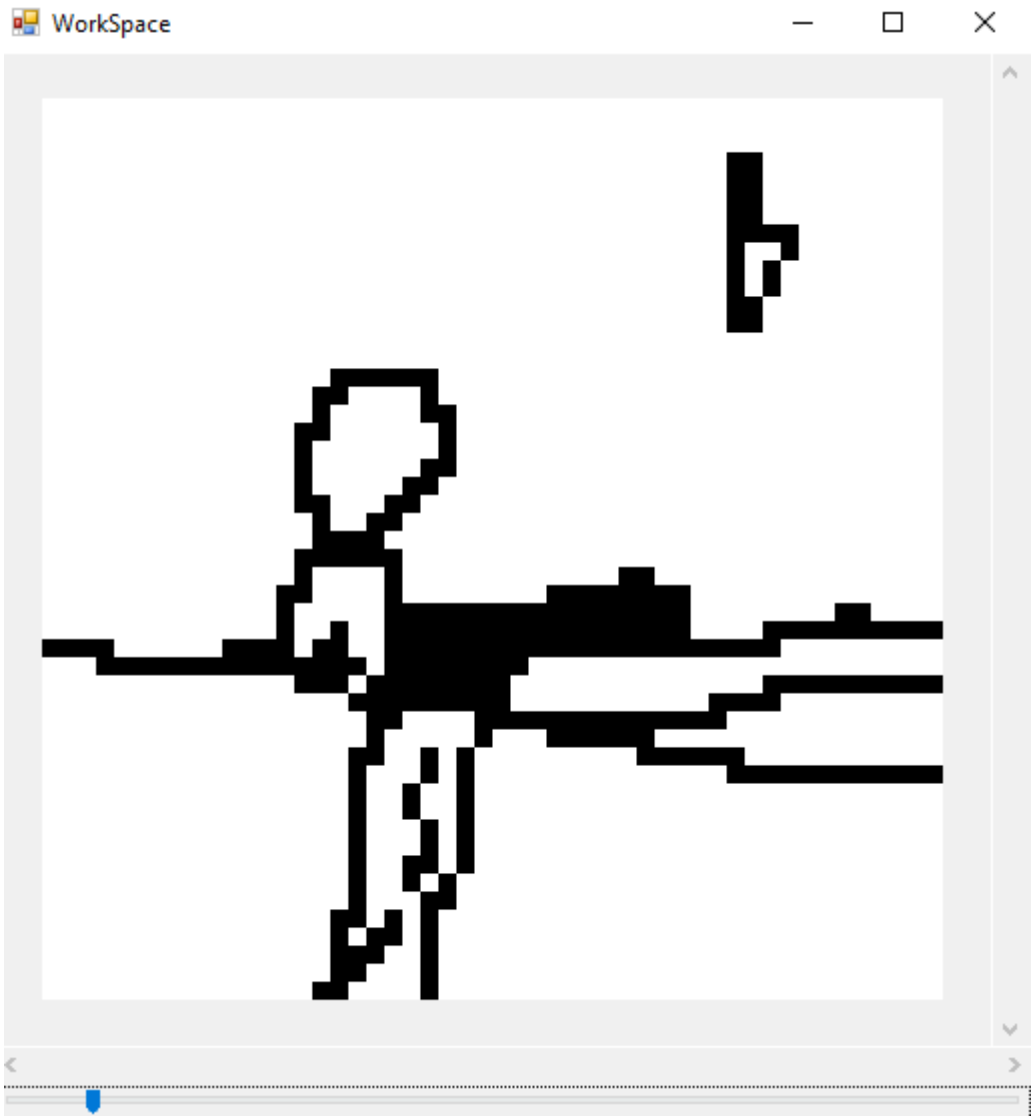
Which drastically reduces the code complexity, and increases performance.

Alex H Client Feedback

User Experience

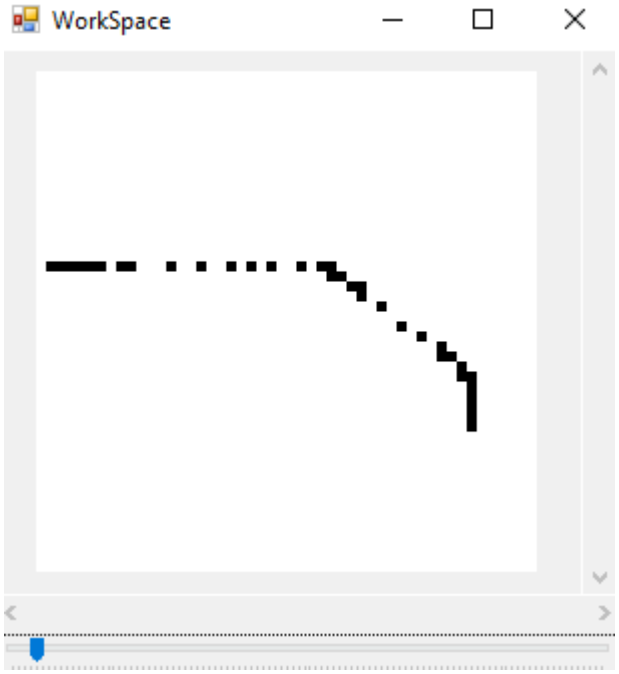
Alex H liked the start to the program so far, the zooming and scrolling capabilities, and the shortcuts. However he did experience a few issues.

Regardless he did draw a picture using the program



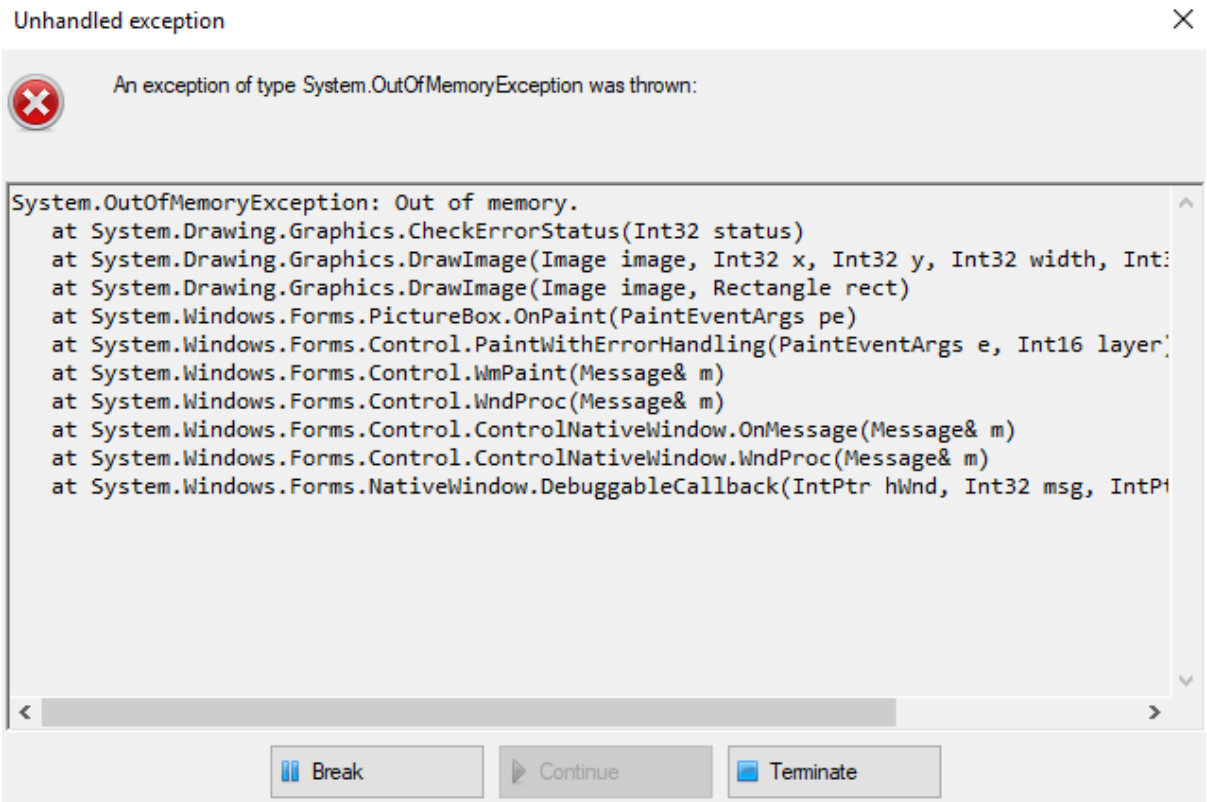
Inconsistent Brush

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



Memory Leak Crash

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

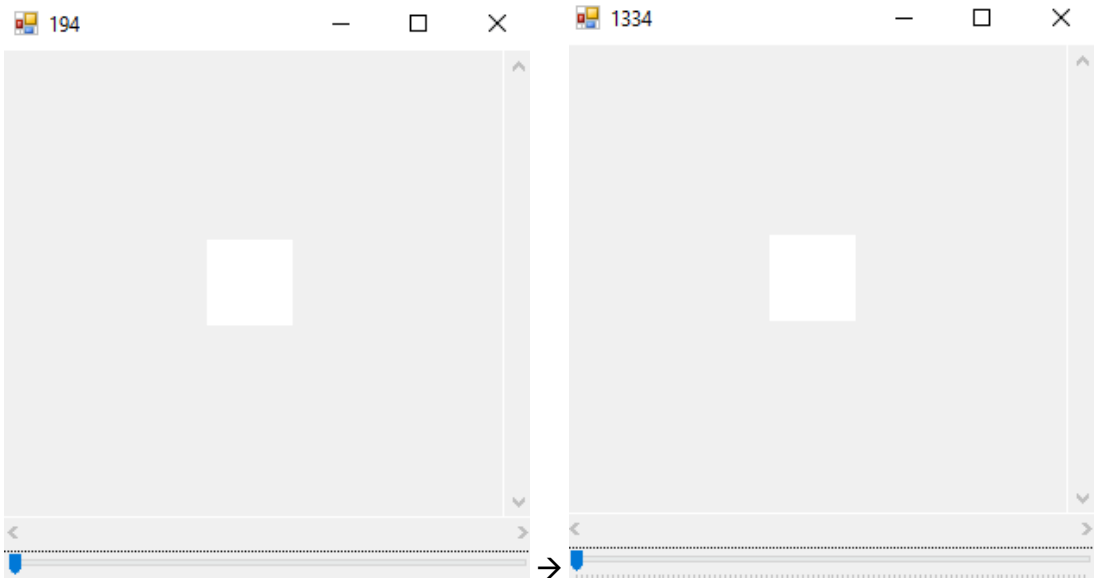


The error occurred because each time a new image is created, the old image is not removed from memory. This means if too many images are created at the same time then they are made faster than the garbage collector can delete them, and the program crashes.

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

```
int test = 0;

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

    // Relocates the picture box
    RelocateDisplayBox();

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

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

    test++;

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

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



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

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

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

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

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

While this helped, when drawing on the image it is still 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.

Apps (6)			
Microsoft Word (32 bit)	0%	50.2 MB	0 MB/s
Microsoft Word (32 bit) (2)	0%	107.0 MB	0 MB/s
SIMP - SharpDevelop	0%	94.9 MB	0 MB/s
SIMP (32 bit) (2)	7.0%	1,551.0 MB	0 MB/s
Task Manager	0.2%	18.2 MB	0 MB/s
Windows Explorer	0.2%	56.5 MB	0 MB/s

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

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

```
updateCount++;

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

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

Apps (6)		
Microsoft Word (32 bit)	0%	50.2 MB
Microsoft Word (32 bit) (2)	0%	88.2 MB
SIMP - SharpDevelop	0.5%	96.1 MB
SIMP (32 bit) (2)	14.3%	25.6 MB
Task Manager	0.3%	18.5 MB
Windows Explorer	0.1%	56.7 MB

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

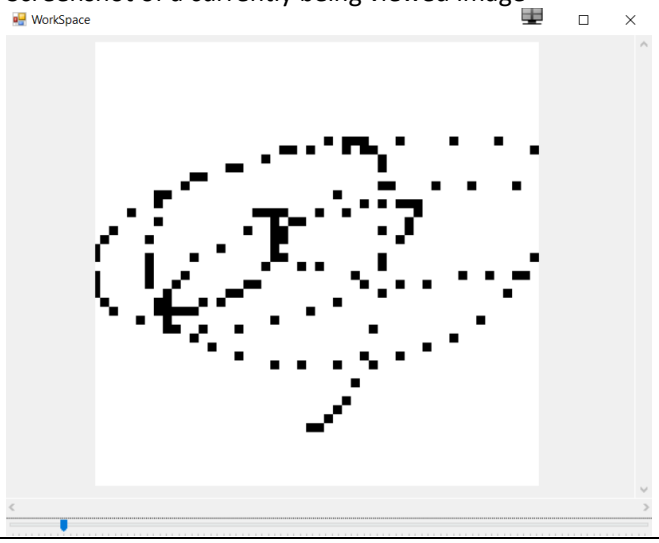
Dheshpreet Feedback

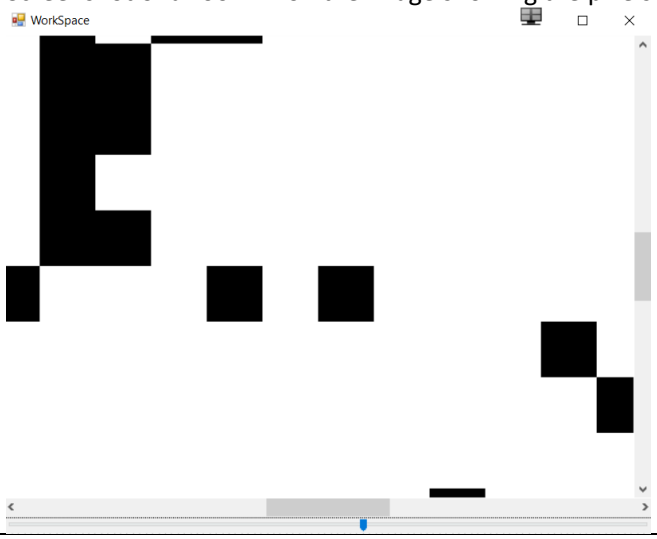
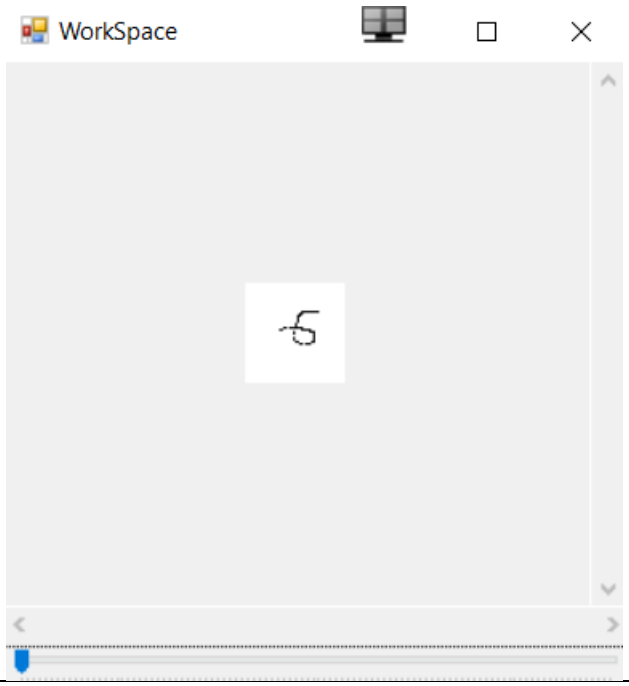
Today when Dheshpreet tested the program, she did generally enjoy the program, and was able to draw a picture.

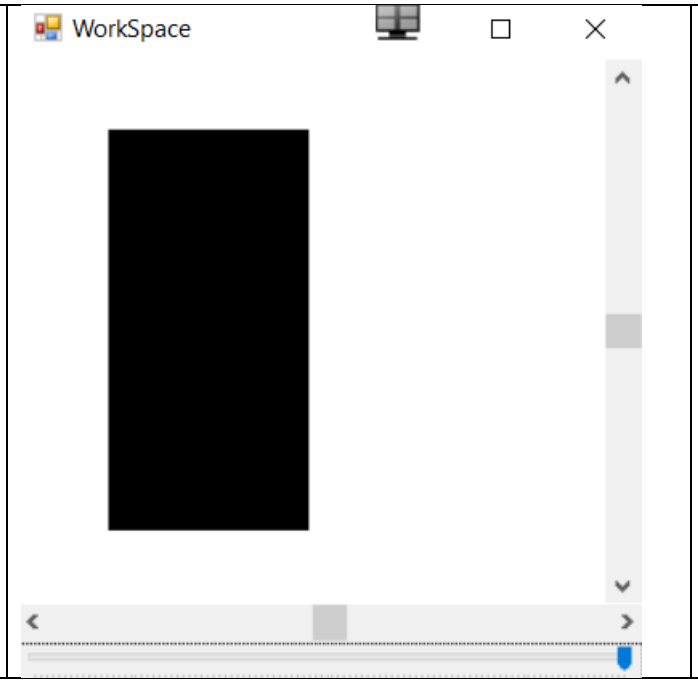
However she did find it difficult to draw without an undo feature. As this is a complicated feature it cannot be added now, so should be implemented in the next development phase.

2.3.3 Success Criteria Evaluation

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

Feature	Proof	Code
Section B – Other editing tools		
Image viewer	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