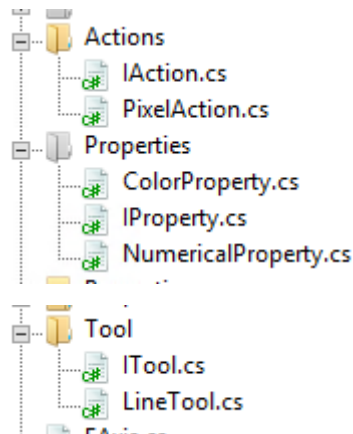


3.2 Development

05/11/2019 Class Design

Today I implemented the necessary classes for the next phase of SIMP. They have been organised into folders. They will be implemented later:

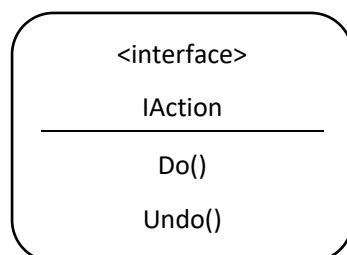


06/11/2019 Class Implementation

The following classes have been implemented:

IAction

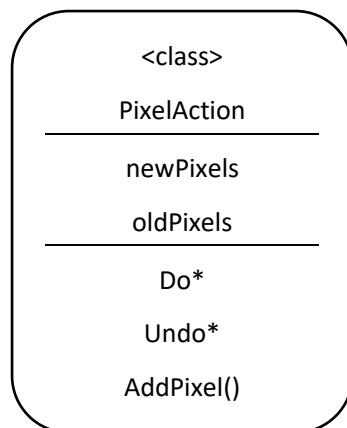
IAction has been implemented with its two functions in accordance to 3.1.3.1:



```
public interface IAction
{
    void Do();
    void Undo();
}
```

PixelAction

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



```
public class PixelAction : IAction
{
    private Dictionary<FilePoint,Color> oldPixels;
    private Dictionary<FilePoint,Color> newPixels;

    public PixelAction()
    {
        oldPixels = new Dictionary<FilePoint, Color>();
        newPixels = new Dictionary<FilePoint, Color>();
    }

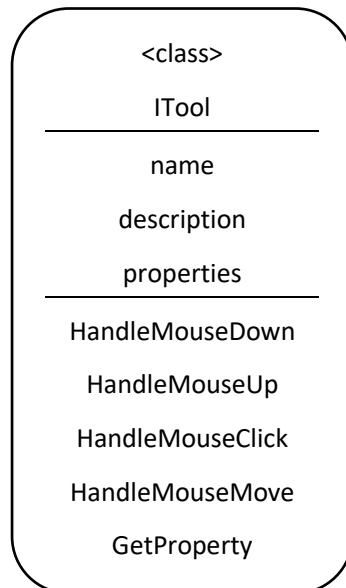
    public void AddPixel(FilePoint pixelLocation, Color oldColour, Color newColour) {
        // saves the old and new colours in the dictionary
        oldPixels[pixelLocation] = oldColour;
        newPixels[pixelLocation] = newColour;
    }

    public void Do(Workspace workspace) {
        //TODO: PixelAction Do()
        throw new NotImplementedException();
    }

    public void Undo(Workspace workspace) {
        //TODO: PixelAction Undo()
        throw new NotImplementedException();
    }
}
```

ITool

ITool has received a major change, it has been changed from an Interface to an Abstract Class. This is because the GetProperty method is common code for each class, there should be no need for each class to implement its own GetProperty method. Thus an Abstract Class is a better fit:



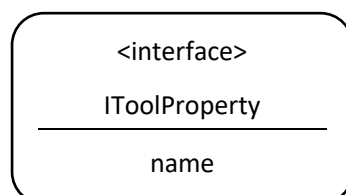
```
public abstract class ITool
{
    public string name;
    public string description;
    public List<SIMP.Properties.IProperty> properties

    public abstract void HandleMouseDown(FilePoint clickLocation, MouseButton button);
    public abstract void HandleMouseUp(FilePoint clickLocation, MouseButton button);
    public abstract void HandleMouseClicked(FilePoint clickLocation, MouseButton button);
    public abstract void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation);

    public SIMP.Properties.IProperty GetProperty(string propertyName) {
        //TODO: ITool GetProperty()
        throw new NotImplementedException();
    }
}
```

IProperty (IToolProperty)

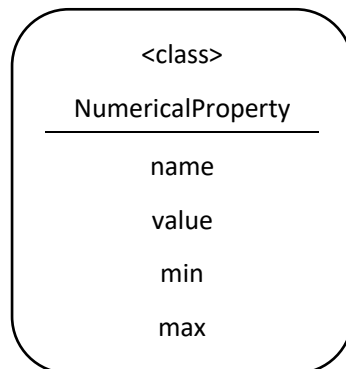
The simple IToolProperty interface has been implemented, though has been renamed to IProperty (as there is no obligation for it to be attached to a tool), and has also been changed to a static class, in accordance to 3.1.3.4:



```
public abstract class IProperty
{
    public string name;
}
```

NumericalProperty

NumericalProperty has been implemented in accordance to 3.1.3.5:



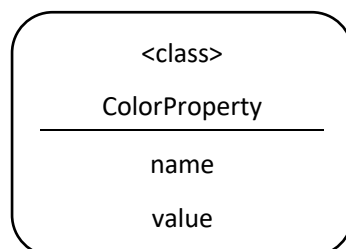
```
public class NumericalProperty : IProperty
{
    public int value;
    public int min;
    public int max;

    public NumericalProperty(string name, string value, string min, string max)
    {
        this.name = name;
        this.value = value;
        this.min = min;
        this.max = max;
    }
}
```

(Name is inherited from IProperty)

ColorProperty

NumericalProperty has been implemented in accordance to 3.1.3.6:



```
public class ColorProperty : IProperty
{
    public Color value;

    public ColorProperty(string name, Color value)
    {
        this.name = name;
        this.value = value;
    }
}
```

LineTool

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

```
public class LineTool : ITool
{
    public LineTool(string name, string description, Color color)
    {
        this.name = name;
        this.description = description;

        this.properties.Add(new ColorProperty("Color",color));
        this.properties.Add(new NumericalProperty("Brush Size",1,1,100));
    }

    public override void HandleMouseDown(FilePoint clickLocation, MouseButton button) {
        //TODO: LineTool HandleMouseDown()
        throw new NotImplementedException();
    }

    public override void HandleMouseUp(FilePoint clickLocation, MouseButton button){
        //TODO: LineTool HandleMouseUp()
        throw new NotImplementedException();
    }

    public override void HandleMouseClicked(FilePoint clickLocation, MouseButton button) {
        //TODO: LineTool HandleMouseClicked()
        throw new NotImplementedException();
    }

    public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
        //TODO: LineTool HandleMouseMove()
        throw new NotImplementedException();
    }

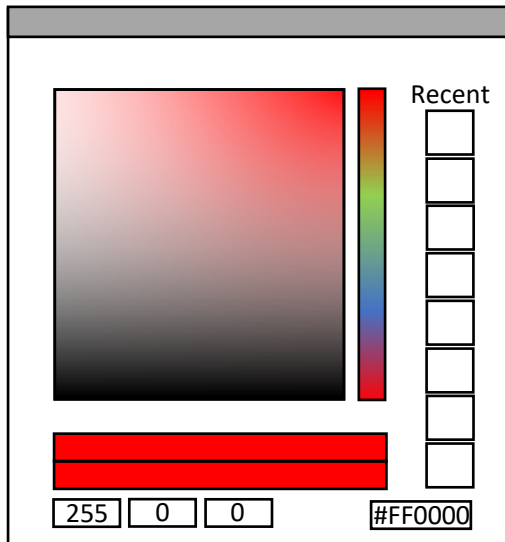
    public void DrawLine() {
        //TODO: LineTool DrawLine()
        throw new NotImplementedException();
    }
}
```

Layer

09/11/2019 RGB Picker design

Form Design

Today the basic outline of the RGB picker has been implemented, in accordance to the updated design in Algorithm 3.1:



There has been a single change made, the hex code input box has been moved to below the color input boxes, as it more closely part of that section. However none of the code for the form has been completed.

HSL to RGB

The algorithm for converting from HSL notation to RGB notation has been implemented, in accordance to Algorithm 3.2B:

```
HSLtoRGB(H, S, L) {  
    C = (1 - Abs((2 * L) - 1)) * S  
    X = C * (1 - Abs(((H / 60) % 2) - 1))  
    M = L - C/2  
    IF H < 60 THEN  
        r = C, g = X, b = 0  
    ELSE IF H >= 60 && H < 120 THEN  
        r = X, g = C, b = 0  
    ELSE IF H >= 120 && H < 180 THEN  
        r = 0, g = C, b = X  
    ELSE IF H >= 180 && H < 240 THEN  
        r = 0, g = X, b = C  
    ELSE IF H >= 240 && H < 300 THEN  
        r = X, g = 0, b = C  
    ELSE IF H >= 300 && H < 360 THEN  
        r = C, g = 0, b = X  
    END IF  
    R = (r + m) * 255  
    G = (g + m) * 255  
    B = (b + m) * 255  
    RETURN new Colour(R,G,B)  
}
```

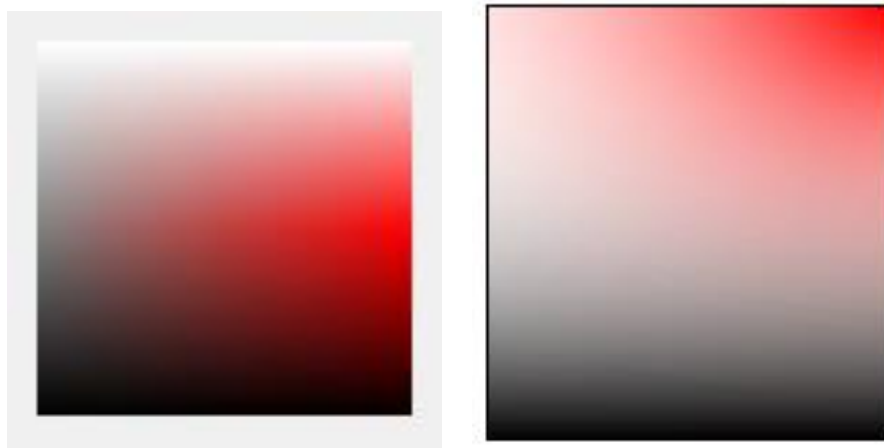
```
public static Color HSLtoRGB(double h, double s, double l) {  
    double c = (1 - Math.Abs((2*l)-1)) * s;  
    double x = c * (1-Math.Abs(((h/60)%2)-1));  
    double m = 1-(c/2);  
    double r,g,b;  
    if (h < 60) {  
        r = c; g = x; b = 0;  
    }  
    else if (h >= 60 && h < 120) {  
        r = x; g = c; b = 0;  
    }  
    else if (h >= 120 && h < 180) {  
        r = 0; g = c; b = x;  
    }  
    else if (h >= 180 && h < 240) {  
        r = 0; g = x; b = c;  
    }  
    else if (h >= 240 && h < 300) {  
        r = x; g = 0; b = c;  
    }  
    else {  
        r = c; g = 0; b = x;  
    }  
    int R = (int)((r+m) * 255);  
    int G = (int)((g+m) * 255);  
    int B = (int)((b+m) * 255);  
    return Color.FromArgb(R,G,B);  
}
```

Generating Colour Square

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

```
private void UpdateColourSquare() {  
    float hue = _startColor.GetHue();  
    Bitmap newImage = new Bitmap(100,100);  
  
    // generate each pixel  
    for (float x = 0; x < 100; x++) {  
        for (float y = 0; y < 100; y++) {  
            // uses 1-() in order to flip in y axis  
            Color currentColour = SIMPRGB.HSLtoRGB(hue,x/100,1-(y/100));  
            newImage.SetPixel((int)x,(int)y,currentColour);  
        }  
    }  
  
    picColourSquare.Image = newImage;  
}
```

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



This is due to the fact that the analysed program, GIMP, uses HSV colour notation rather than HSL. The calculation for HSV differ slightly, resulting in a different square. HSV cannot be used in SIMP without extra work due to the fact that the internal C# libraries for Color use HSL notation, as the Hue of a colour can be gotten with a pre-written function. This means the program will run faster.

Other Graphics

Code has been written for the other graphics objects, making the GUI currently look like so:



10/11/2019 RGB Picker functionality

Numerical Inputs

Some simple code for updating the current colour when a numeric input has been implemented. Each one checks the Boolean updating before proceeding, to make sure that the boxes aren't being updated by code:

```
void NumRValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb((byte)numR.Value, _currentColor.G, _currentColor.B);

    Update();
}

void NumGValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb(_currentColor.R, (byte)numG.Value, _currentColor.B);

    Update();
}

void NumBValueChanged(object sender, EventArgs e)
{
    if (_updating) {
        return;
    }

    _currentColor = Color.FromArgb(_currentColor.R, _currentColor.G, (byte)numB.Value);

    Update();
}

private new void Update()-{
    _updating = true;

    UpdateColourSquare();
    UpdateColourStrip();
    UpdateColourDisplays();
    UpdateColourInputs();

    _updating = false;
}
```

This prevents the current colour from being changed during update.

Colour Strip Input

Gathering input from the Colour Strip is also reasonably simple. As the strip is 100px tall simply multiplying the Y by 100 gets the wanted hue:

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

    float hue = ((float)me.Location.Y)*3.6f;
    _currentColor = _currentColor.SetHue(hue);

    Update();
}
```

Colour Square Input

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

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

    float hue = _currentColor.GetHue();
    float saturation = ((float)me.Location.X)/100;
    float lightness = 1-((float)me.Location.Y)/100;

    _currentColor = SIMPRGB.HSLtoRGB(hue,saturation,lightness);

    Update();
}
```

RGB Display

When the first code for the RGB display was implemented, a problem arose. This was because when the code converted a number to hexadecimal, it did not pad the number, meaning that a hex code could be generated that was less than 6 in width:

```
private void UpdateColourInputs() {
    numR.Value = _currentColor.R;
    numG.Value = _currentColor.G;
    numB.Value = _currentColor.B;

    txtRGB.Text = string.Format("#{0}{1}{2}", _currentColor.R.ToString("X"), _currentColor.G.ToString("X"), _currentColor.B.ToString("X"));
}
```



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

```
private void UpdateColourInputs() {  
    numR.Value = _currentColor.R;  
    numG.Value = _currentColor.G;  
    numB.Value = _currentColor.B;  
  
    string R = _currentColor.R.ToString("X").PadLeft(2, '0');  
    string G = _currentColor.G.ToString("X").PadLeft(2, '0');  
    string B = _currentColor.B.ToString("X").PadLeft(2, '0');  
  
    txtRGB.Text = string.Format("#{0}{1}{2}", R, G, B);  
}
```

The parts are now padded

Problem with returning

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

Solution - Delegates

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

```
public delegate void ColorCallback(Color newColour);
```

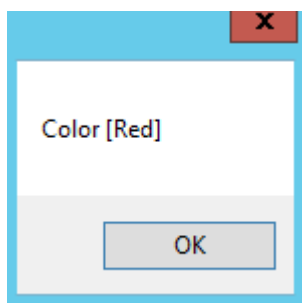
Thus, when a new Save button is clicked, this delegate is called, sending the new colour back to the caller and allowing it to do whatever is needed with the new colour:

```
void BtnSaveClick(object sender, EventArgs e)  
{  
    // calls the callback to be handled by caller  
    _updateCallback(_currentColor);  
  
    Close();  
}
```

Meaning a call can be made like this:

```
RGBPicker newPicker = new RGBPicker(Color.Red, delegate(Color newColour) {  
    MessageBox.Show(newColour.ToString());  
});
```

The colour is indeed correctly returned after pressing 'Save':



13/11/2019 Defining Tools

Storing Tools

The simple brush has been added to the workspace, though only in basic class form. This was done by defining a list, and adding the relevant info for the linetool:

```
// Defines tools
tools.Add(new LineTool("Brush", "Simple Brush", Color.Black));
```

Displaying Tools

However, it then becomes necessary to display the possible tools on the side of the program. Some simple code can be written to accomplish this:

```
// Adds buttons
int buttonY = 0;
foreach (ITool tool in tools) {
    Button newButton = new Button();
    newButton.Size = new Size(24, 24);
    newButton.Location = new Point(0, buttonY);
    newButton.Name = tool.name;
    newButton.Click += new EventHandler(ToolButtonClick);
    buttonY += 24;

    panTools.Controls.Add(newButton);
}
```

Then when the button is pressed:

```
void ToolButtonClick(object sender, EventArgs e) {
    Button buttonSender = (Button)sender;

    // disables every button except for pressed one
    foreach (Button button in panTools.Controls) {
        button.Enabled = true;
    }
    buttonSender.Enabled = false;

    // selects the tool
    foreach (ITool tool in tools) {
        if (tool.name.Equals(buttonSender.Name)) {
            currentTool = tool;
        }
    }
}
```

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

Hooking Tools

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

Before:

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

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

After:

```
void DisplayBoxMouseDown(object sender, MouseEventArgs e)
{
    DisplayPoint clickLocation = new DisplayPoint(e.Location.X,e.Location.Y);
    currentTool.HandleMouseDown(image.DisplayPointToFilePoint(clickLocation),e.Button);
}

DisplayPoint oldLocation = new DisplayPoint(0,0);
void DisplayBoxMouseMove(object sender, MouseEventArgs e)
{
    //TODO: Finish the clicking stuff
    DisplayPoint newLocation = new DisplayPoint(e.Location.X,e.Location.Y);
    currentTool.HandleMouseMove(image.DisplayPointToFilePoint(oldLocation),image.DisplayPointToFilePoint(newLocation));

    oldLocation = newLocation;
}
```

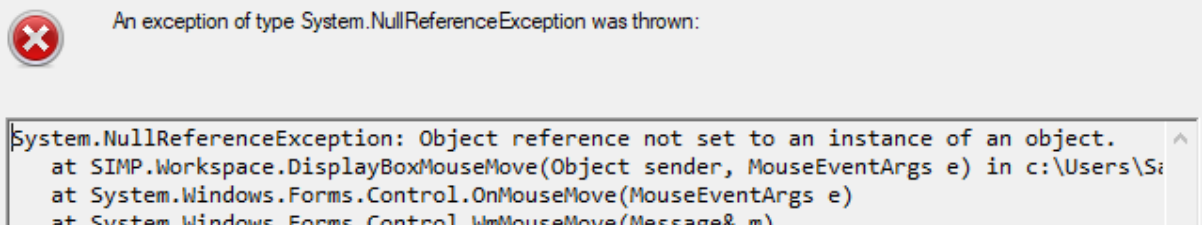
Which uses a new constructor for `FilePoint`

```
public FilePoint(Point p) : this(p.X,p.Y) {
    // no extra implementation
}
```

Fixing a NullReferenceError

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

Unhandled exception



However, to make it so that there is a clean way of showing that there is a blank tool, a `BlankTool` class can be created and statically stored in the encompassing `ITool` class:

```
internal class BlankTool : ITool
{
    internal BlankTool()
    {
        //nothing
    }

    // no implementation
    public override void HandleMouseDown(FilePoint clickLocation, MouseButton button){}
    public override void HandleMouseUp(FilePoint clickLocation, MouseButton button){}
    public override void HandleMouseClicked(FilePoint clickLocation, MouseButton button){}
    public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation){}
}
```

In `ITool`:

```
//blanktool
public static ITool BlankTool;

static ITool() {
    BlankTool = new BlankTool();
}
```

And so the Blank Tool can be set like so:

```
currentTool = ITool.BlankTool;
```

Meaning that the crash is avoided.

16/11/2019 Implementing LineTool

A simple implementation for the tool can be coded, using a Boolean for whether the brush is down or not:

```
public override void HandleMouseDown(FilePoint clickLocation, MouseButton button) {
    //TODO: LineTool HandleMouseDown()
    _mouseDown = true;
}

public override void HandleMouseUp(FilePoint clickLocation, MouseButton button){
    //TODO: LineTool HandleMouseUp()
    _mouseDown = false;
}

public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
    //TODO: LineTool HandleMouseMove()
    Color myColor = Color.Black;
    if (_mouseDown) {
        myWorkspace.image.SetPixel(newLocation, myColor);
        myWorkspace.UpdateDisplayBox(true);
    }
}
```

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

Implementing Colour Parameter

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

```
public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {
    //TODO: LineTool HandleMouseMove()
    if (_mouseDown) {
        Color myColor = ((ColorProperty)GetProperty("Color")).value;
        myWorkspace.image.SetPixel(newLocation, myColor);
        myWorkspace.UpdateDisplayBox(true);
    }
}
```

Casts an IProperty to a ColorProperty to get its value

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

```
public SIMP.Properties.IProperty GetProperty(string propertyName) {
    //TODO: ITool GetProperty()
    foreach (IProperty property in properties) {
        if (property.name.Equals(propertyName)) {
            return property;
        }
    }
    throw new KeyNotFoundException("Couldn't find property " + propertyName);
}
```

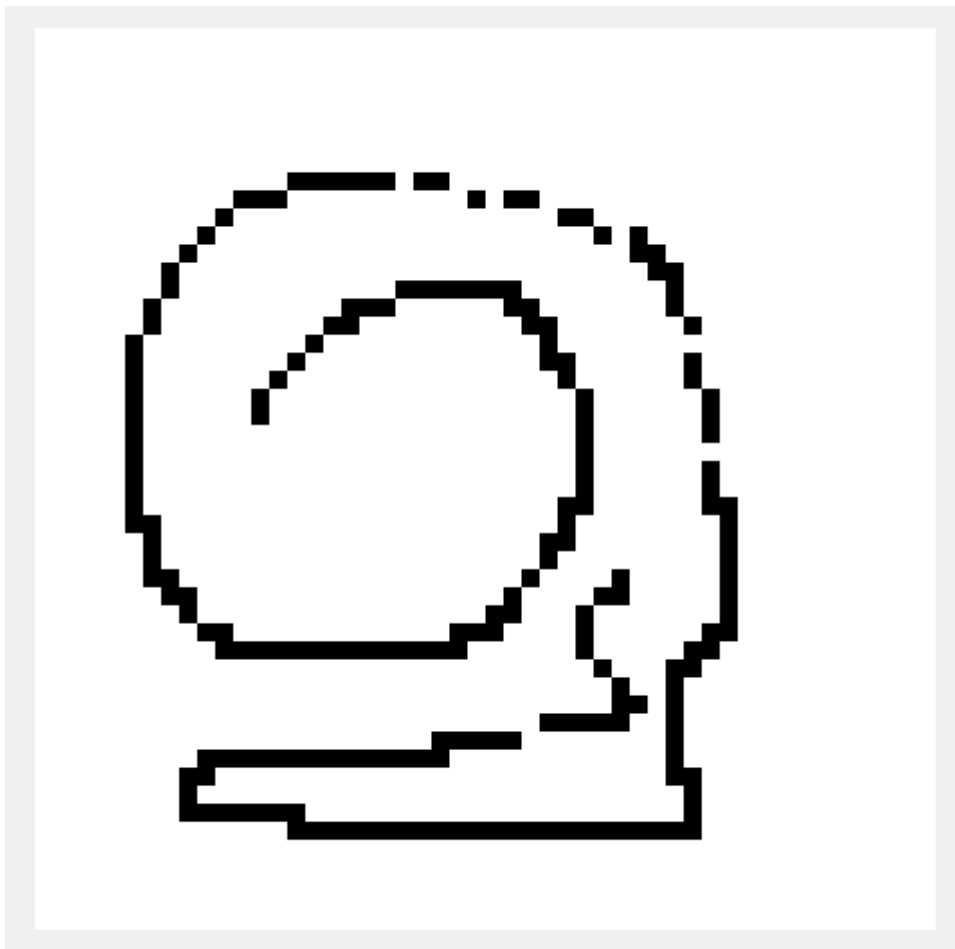

Connecting Pixels – Bresenham's

First, a check for whether two pixels is adjacent is implemented. This is because two adjacent pixels don't require Bresenham's Algorithm, it can just be simply drawn:

```
private bool IsAdjacent(FilePoint point1, FilePoint point2) {  
    if (Math.Abs(point1.fileX - point2.fileX) > 1) {  
        return false;  
    }  
    if (Math.Abs(point1.fileY - point2.fileY) > 1) {  
        return false;  
    }  
    return true;  
}
```

Then, Bresenham's algorithm can be implemented in accordance to Algorithm 3.6A

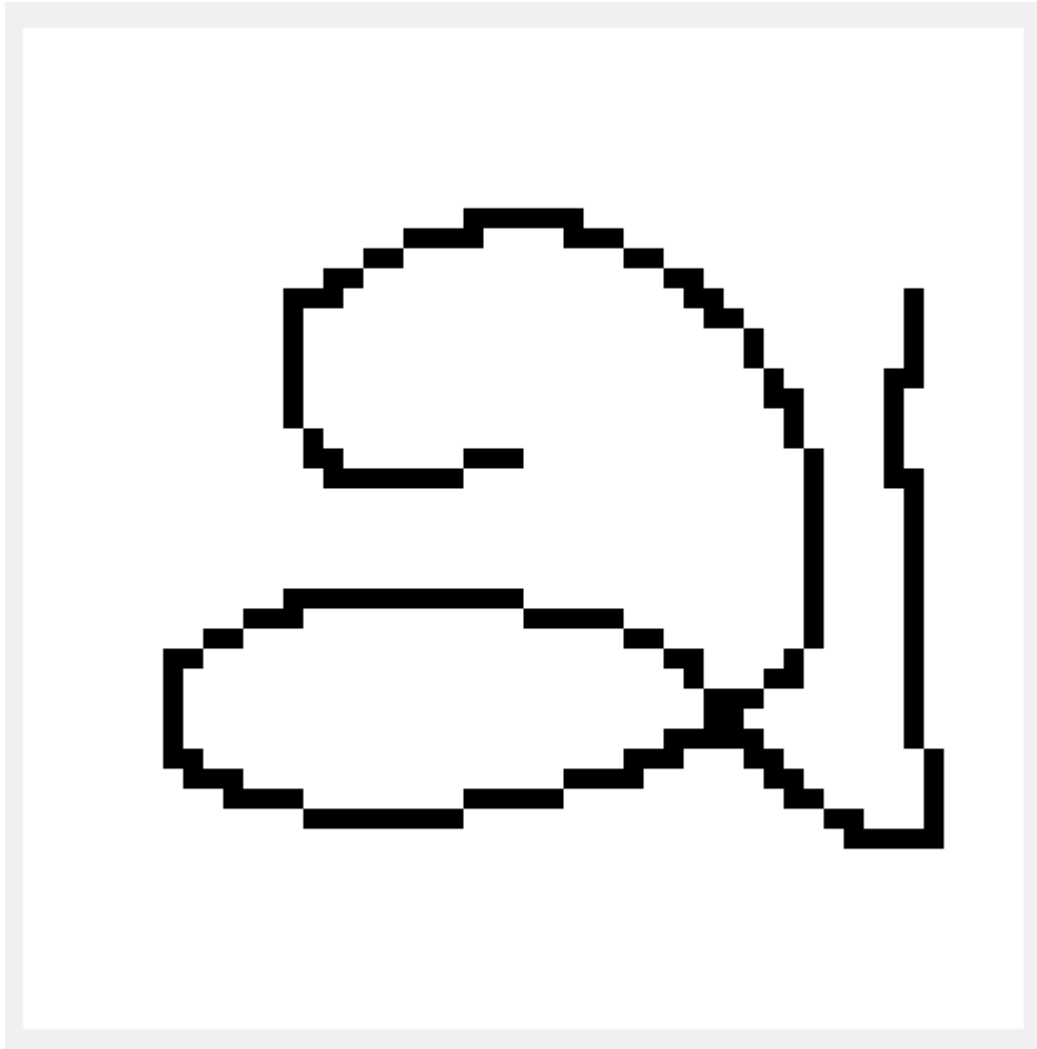
However, it was discovered that there was a problem with the implemented algorithm, it sometimes left a gap:



The problem turned out to be that the last pixel is not set by Bresenham's Algorithm, so it needs to be set manually, with an extra line of code:

```
myWorkspace.image.SetPixel(newLocation,myColor);  
DrawLine(oldLocation,newLocation);
```

This makes sure all lines are now connected:



17/11/2019 Generating Brushes

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

```
private List<FilePoint> currentBrush;
```

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

```
FOR x = brushSize*-1 TO brushSize
  FOR y = brushSize*-1 TO brushSize
    IF IsPointInCircle(x+0.5, y+0.5) THEN
      currentBrush.Add(x,y)
    END IF
  NEXT
NEXT
```

```
private void GenerateBrush() {
    this.currentBrush = new List<FilePoint>();

    int radius = 10;

    for (float x = radius * -1; x < radius; x++) {
        for (float y = radius * -1; y < radius; y++) {
            if (IsPointInCircle(x + 0.5f, y + 0.5f)) {
                currentBrush.Add(new FilePoint((int)x, (int)y));
            }
        }
    }
}
```

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

```
IsPointInCircle(x,y) {  
    IF ((x*x)+(y*y) <= (brushSize*brushSize)) THEN  
        RETURN true  
    ELSE  
        RETURN false  
    END IF  
}
```

```
private bool IsPointInCircle(float x, float y, float radius) {  
    // returns whether this condition is true or false  
    return ((x*x)+(y*y)) <= (radius * radius);  
}
```

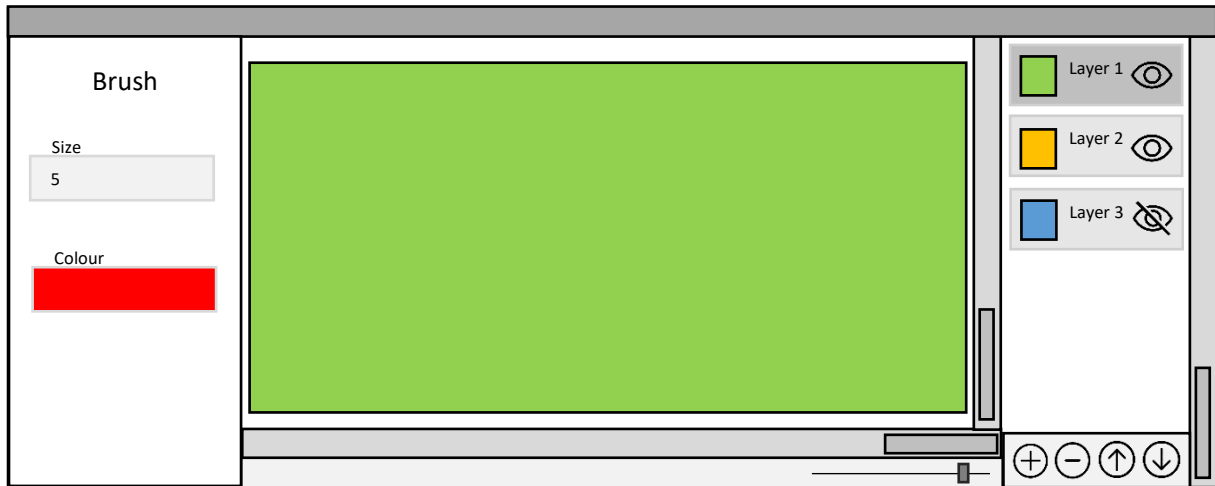
Finally, with a small piece of code to set the relevant pixels, thick brushes can now be used:



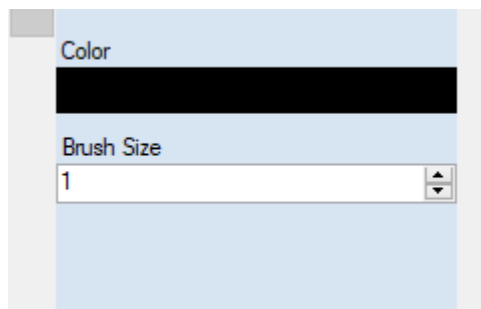
18/11/2019 Implementing Properties

All the properties thus far have been drawing from constants. It is now necessary that the user can edit these properties, so a new menu must be designed.

This menu should simply be a list of properties, placed onto the left hand side of the screen. As a mockup:



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



```

public void ShowTool() {
    leftPadding = SimpConstants.WORKSPACE_LEFT_PADDING + 200;
    panToolProperties.Visible = true;
    panToolProperties.Controls.Clear();

    Label lblToolName = new Label();
    lblToolName.Location = new Point(6, 0);
    lblToolName.Size = new Size(191, 23);
    lblToolName.Text = currentTool.name;
    panToolProperties.Controls.Add(lblToolName);

    int currentY = lblToolName.Height;
    int width = panToolProperties.Width;
    foreach (IProperty property in currentTool.properties) {
        Label newLabel = new Label();
        newLabel.Size = new Size(width, SimpConstants.PROPERTY_LABEL_HEIGHT);
        newLabel.Location = new Point(0, currentY);
        newLabel.Text = property.name;
        panToolProperties.Controls.Add(newLabel);
        currentY += SimpConstants.PROPERTY_LABEL_HEIGHT;

        currentY += SimpConstants.PROPERTY_GAP_HEIGHT;

        if (property is ColorProperty) {
            ColorProperty colorProperty = (ColorProperty)property;
            PictureBox newPicture = new PictureBox();
            newPicture.Size = new Size(width, SimpConstants.PROPERTY_FIELD_HEIGHT);
            newPicture.Location = new Point(0, currentY);
            newPicture.BackColor = colorProperty.value;
            newPicture.Click += colorProperty.onInteract;
            panToolProperties.Controls.Add(newPicture);
        } else if (property is NumericalProperty) {
            NumericUpDown newNum = new NumericUpDown();
            NumericalProperty numericalProperty = (NumericalProperty)property;
            newNum.Size = new Size(width, SimpConstants.PROPERTY_FIELD_HEIGHT);
            newNum.Location = new Point(0, currentY);
            newNum.Value = numericalProperty.min;
            newNum.Minimum = numericalProperty.min;
            newNum.Maximum = numericalProperty.max;
            newNum.Value = numericalProperty.value;
            newNum.ValueChanged += numericalProperty.onInteract;
            panToolProperties.Controls.Add(newNum);
        }
        currentY += SimpConstants.PROPERTY_FIELD_HEIGHT;

        currentY += SimpConstants.PROPERTY_SPACER_HEIGHT;
    }

    CalculateDimensions();
    UpdateDisplayBox(true);
}

```

After this, the value updating could be handled by a delegate model, where each Property has its own OnInteract delegate that is called whenever their respective field has been interacted with:

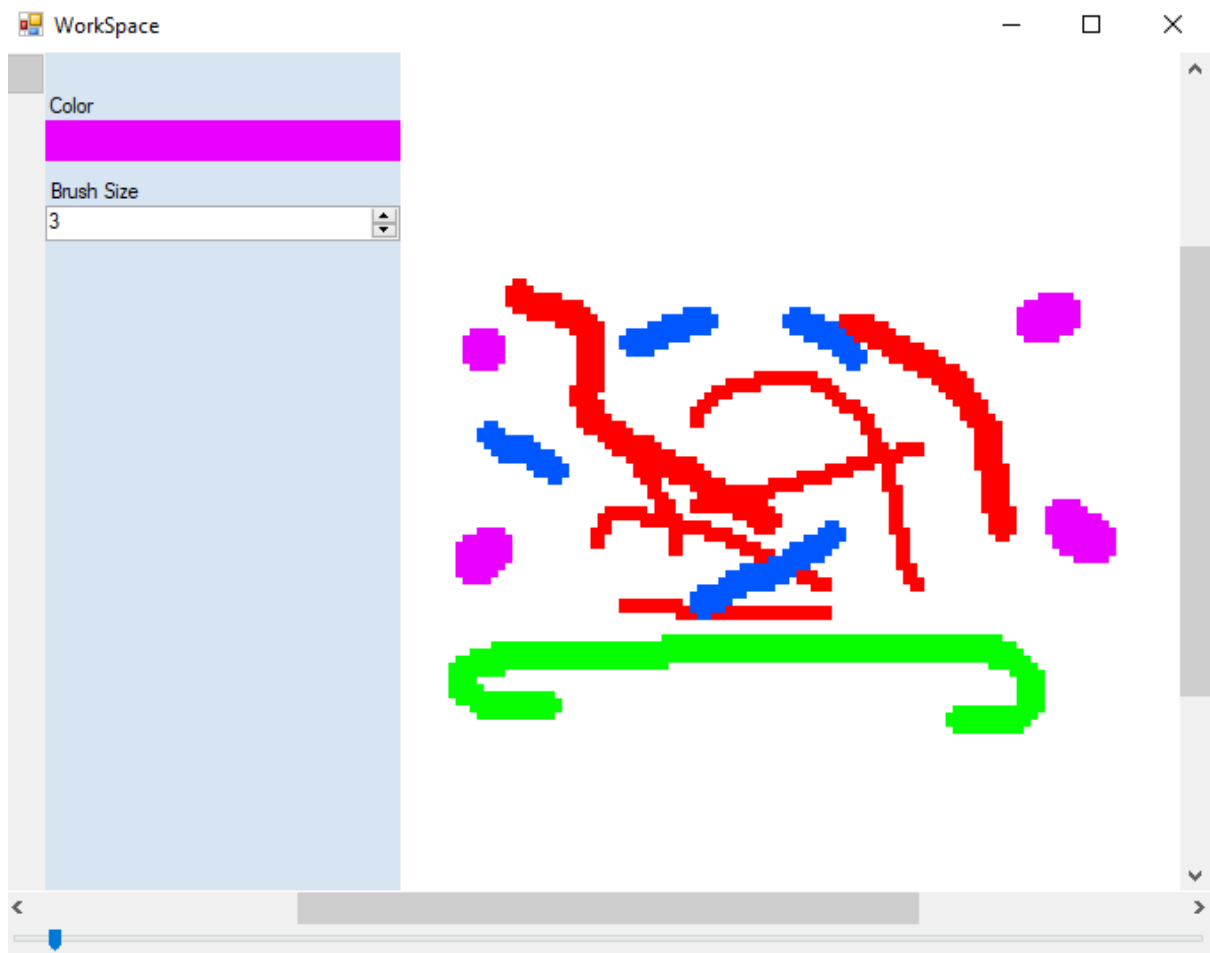
```

public abstract class IProperty
{
    public string name;
    public EventHandler onInteract;
    public Workspace myWorkspace;
}

```

onInteract is implemented by each property.

This now means that colour images can be created with SIMP, and multiple thicknesses of line:



19/11/2019 Stakeholder feedback

Today I presented the current version of SIMP for my stakeholders to use. I received the following feedback:

Alex H

- Creating a brush of size 100 and then pasting on the image causes a large freeze
- Clicking the top of the hex selector fixes the hue at red until colour is changed
- Inputting hex code
- Better representation of how thick the brush will be
- Tutorial
- No copy paste
- No layers
- Brush starts unselected
- No copy and paste
- Creating new image should be inside workspace
- Must start drawing on canvas
- No recent colours
- No rubber
- No single pixel brush

Alex G

- Colour picker improvements:
 - Holding button down should update in real-time
 - Picker area should have the brightest hue at top
 - Releasing button off grid doesn't update colour to max
- Side panel can't be hidden
- Zoom should display numerically how much zoom
- Holding mouse off canvas and then moving on doesn't work
- Resizing window to smallest when having properties open causes a crash

Dheshpreet

- Needs a single pixel brush
- No eyedropper tool
- No layers
- No undo / redo
- Slow response times

Categorizing Feedback

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

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

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

Performance issues with large brushes

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

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

Under normal condition, only the valid section is used. However when big brushes are used the majority of the points are invalid, meaning many error are thrown. Computationally error handling is very expensive, and so many expensive throws will slow the program massively.

To fix this, the brush checked code can be edited to check whether the pixel is in-bounds before checking it:

```
foreach (FilePoint brushPoint in currentBrush) {
    stampPoint = new FilePoint(x + brushPoint.fileX, y + brushPoint.fileY);
    if (stampPoint.fileX < 0 ||
        stampPoint.fileX >= myWorkspace.image.fileWidth ||
        stampPoint.fileY < 0 ||
        stampPoint.fileY >= myWorkspace.image.fileHeight) {
        continue;
    }
    myWorkspace.image.SetPixel(stampPoint,myColor);
}
```

Performance issues displaying image

Currently when making brush manipulations, it is very slow to update. This is due to the fact that when a brush makes a change, the entire image is updated. This is very inefficient, so can be changed to only update changed pixels (if needed).

To do this, a list of changed pixels was introduced, which is updated whenever a change is made to the pixels:

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

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

```
private void DrawChangedImage(Graphics GFX) {  
    DisplayPoint displayChangedLoc;  
    foreach (FilePoint changedPixel in changedPixels) {  
        // finds this pixels location  
        displayChangedLoc = FilePointToDisplayPoint(changedPixel);  
        GFX.FillRectangle(pixels[changedPixel.fileX, changedPixel.fileY],  
            displayChangedLoc.displayX, displayChangedLoc.displayY,  
            zoomSettings.zoom, zoomSettings.zoom);  
    }  
    changedPixels.Clear();  
}
```

This significantly improves performance on larger images.

Using HSV rather than HSL

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

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

```
double c = s * l;  
double x = c * (1-Math.Abs(((h/60)%2)-1));  
double m = l - c;  
double r = (1 - Math.Abs((2*l)-1)) * s;  
double g = c * (1-Math.Abs(((h/60)%2)-1));  
double b = l-(c/2);
```

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

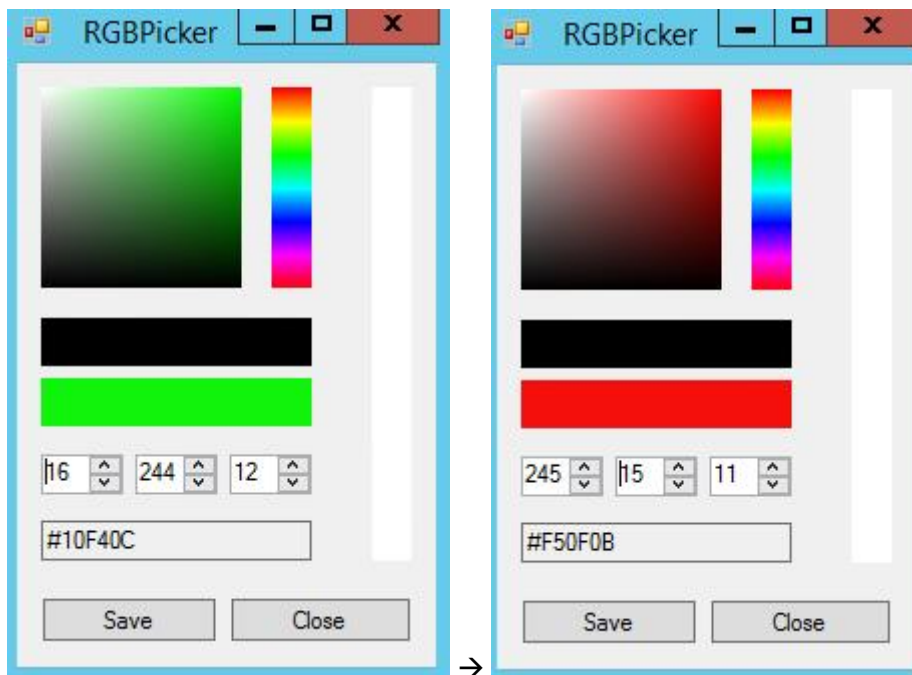
This means an algorithm for the conversion will need to be implemented:

```
public static void RGBtoHSV(double r, double g, double b) {  
    double R = r/255;  
    double G = g/255;  
    double B = b/255;  
  
    double Cmax = Max(R,G,B);  
    double Cmin = Min(R,G,B);  
  
    double Δ = Cmax - Cmin;  
  
    double h,s,v;  
    if (Δ == 0) {  
        h = 0;  
    } else if (Cmax == R) {  
        h = 60 * ((G-B)/Δ)%6;  
    } else if (Cmax == G) {  
        h = 60 * ((B-R)/Δ)+2;  
    } else if (Cmax == B) {  
        h = 60 * ((B-R)/Δ)+4;  
    }  
  
    if (Cmax == 0) {  
        s = 0;  
    } else {  
        s = Δ/Cmax;  
    }  
  
    v = Cmax;  
}
```

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

```
struct HSVColor {  
    public double H;  
    public double S;  
    public double V;  
  
    public HSVColor(double H, double S, double V) {  
        this.H = H; this.S = S; this.V = V;  
    }  
}
```

However when testing the RGB picker, a problem seems to emerge with the conversion, as it seems to incorrectly convert from RGB to HSV:



The problem was isolated to some missing brackets when calculating the hue, so could be corrected:

```

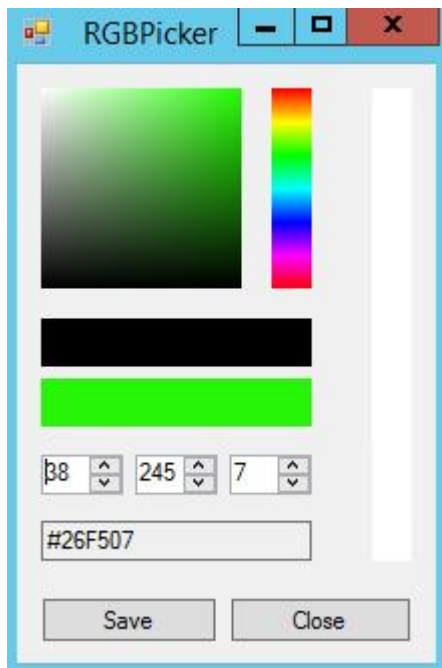
if ( $\Delta$  == 0) {
    h = 0;
} else if (Cmax == R) {
    h = 60 * ((G-B)/ $\Delta$ )%6;
} else if (Cmax == G) {
    h = 60 * ((B-R)/ $\Delta$ )+2;
} else if (Cmax == B) {
    h = 60 * ((B-R)/ $\Delta$ )+4;
}

if ( $\Delta$  == 0) {
    h = 0;
} else if (Cmax == R) {
    h = 60 * (((G-B)/ $\Delta$ )%6);
} else if (Cmax == G) {
    h = 60 * (((B-R)/ $\Delta$ )+2);
} else if (Cmax == B) {
    h = 60 * (((B-R)/ $\Delta$ )+4);
}

```

→

However the numeric input still appears to have some small errors, as in some cases they are unresponsive. For example in this scenario increasing the 'G' does nothing:



This is due to the fact that upon changing colour, all 3 values are recalculated. As the calculation is not reversible this leads to small inaccuracies that can lead to rounding errors.

The solution for this is to introduce a Boolean on the update code on whether the RGB nums are recalculated. This stops them from being recalculated when they are being edited:

```
private void UpdateColourInputs(bool updateNums) {
    Color displayColor = _currentColor.ToColor();

    if (updateNums) {
        numR.Value = displayColor.R;
        numG.Value = displayColor.G;
        numB.Value = displayColor.B;
    }

    string R = ((int)numR.Value).ToString("X").PadLeft(2, '0');
    string G = ((int)numG.Value).ToString("X").PadLeft(2, '0');
    string B = ((int)numB.Value).ToString("X").PadLeft(2, '0');

    txtRGB.Text = string.Format("#{0}{1}{2}", R, G, B);
}
```

20/11/2019 Further improvements

Single Pixel Brush

A single pixel brush has been highly requested, due to the fact that all brushes are stored by radius, and the minimum radius is 1, the smallest possible diameter is 2.

In order to make it, a new class was implemented, SinglePixelLineTool. This inherits from LineTool but continues the constructor to replace the properties with ones to make a line of single pixel width.

However, the only way to make a single width line without redesigning the system is to have a radius of 0.5 (thus diameter 1).

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

```
public class DecimalProperty : IProperty
{
    public decimal value;
    public decimal min;
    public decimal max;

    public DecimalProperty(string name, decimal value, decimal min, decimal max, Workspace myWorkspace)
    {
        this.name = name;
        this.value = value;
        this.min = min;
        this.max = max;
        this.myWorkspace = myWorkspace;

        this.onInteract = delegate(Object sender, EventArgs e) {
            this.value = (decimal)((NumericUpDown)sender).Value;
            myWorkspace.ShowTool();
        };
    }
}
```

Then this can be added to the Single Pixel Brush:

```
this.properties.Clear(); // removes all previous properties
this.properties.Add(new ColorProperty("Color", color, myWorkspace));
this.properties.Add(new DecimalProperty("Brush Size", 0.5M, 0.5M, 0.5M, myWorkspace));
```

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

```
this.currentBrush = new List<FilePoint>();

float radius = (float)((NumericalProperty)GetProperty("Brush Size")).value;
```

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

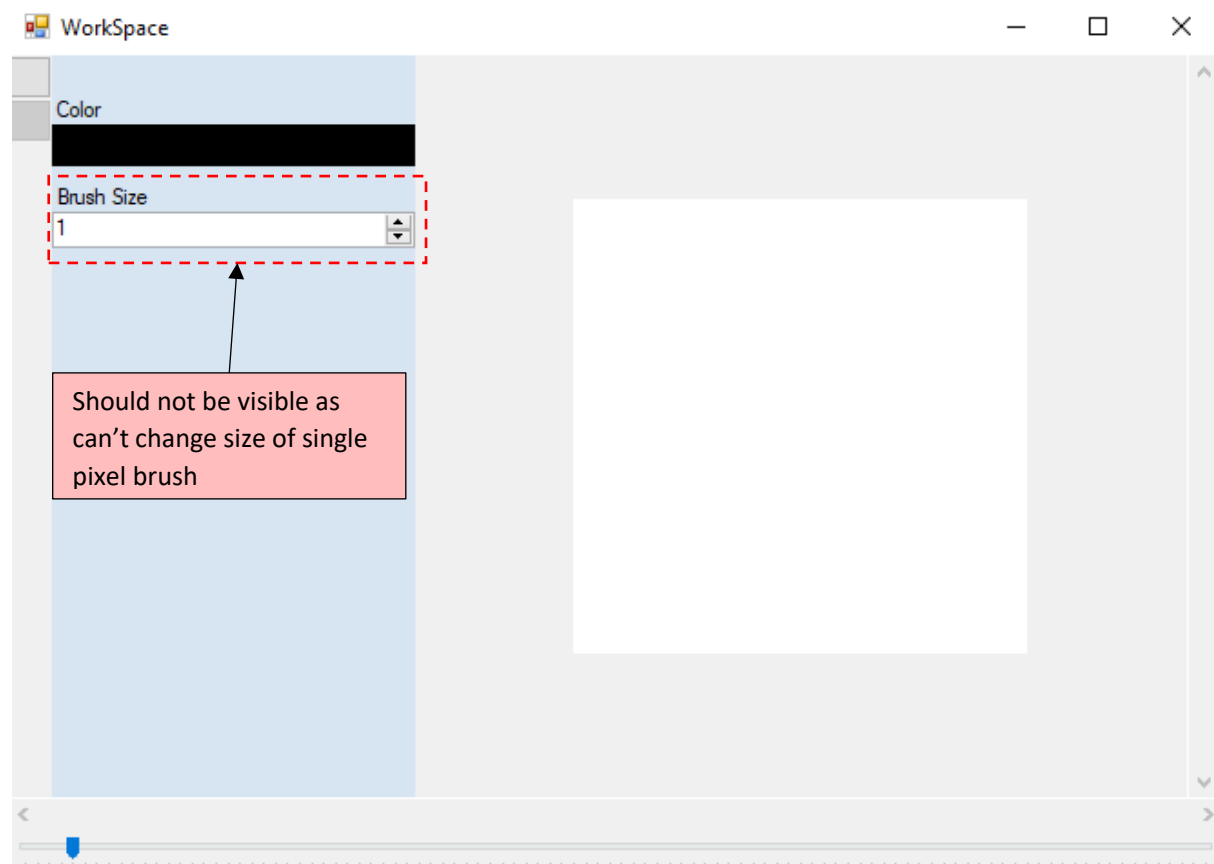
```
public abstract class IProperty
{
    public string name;
    public object value;
    public EventHandler onInteract;
    public Workspace myWorkspace;
}
```

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

```
float radius = Convert.ToSingle(GetProperty("Brush Size").value);
```

Property properties

However, there is still a problem, as the brush size is still being stored as a property, thus will show up on the properties pane:

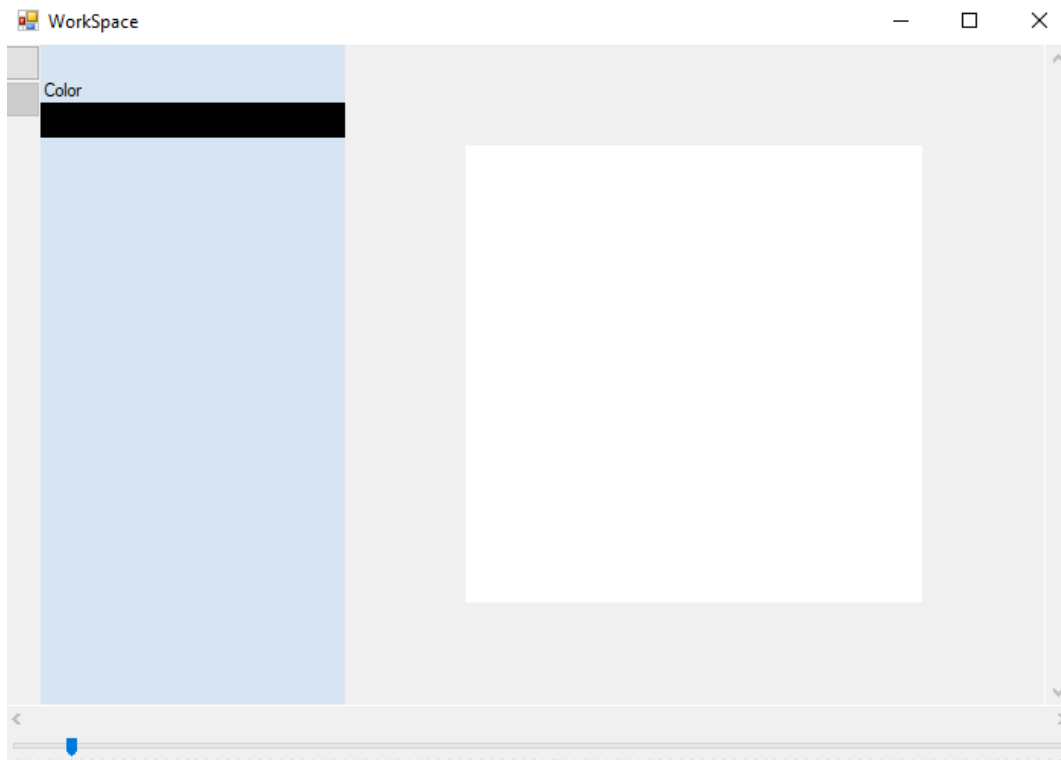


In order to do this, an enum has been introduced, to help organise three possible sorts of property, normal (visible), readonly (cannot be edited) or hidden (cannot be seen or interacted with)

```
public enum PropertyType {
    Normal,
    ReadOnly,
    Hidden
}
```

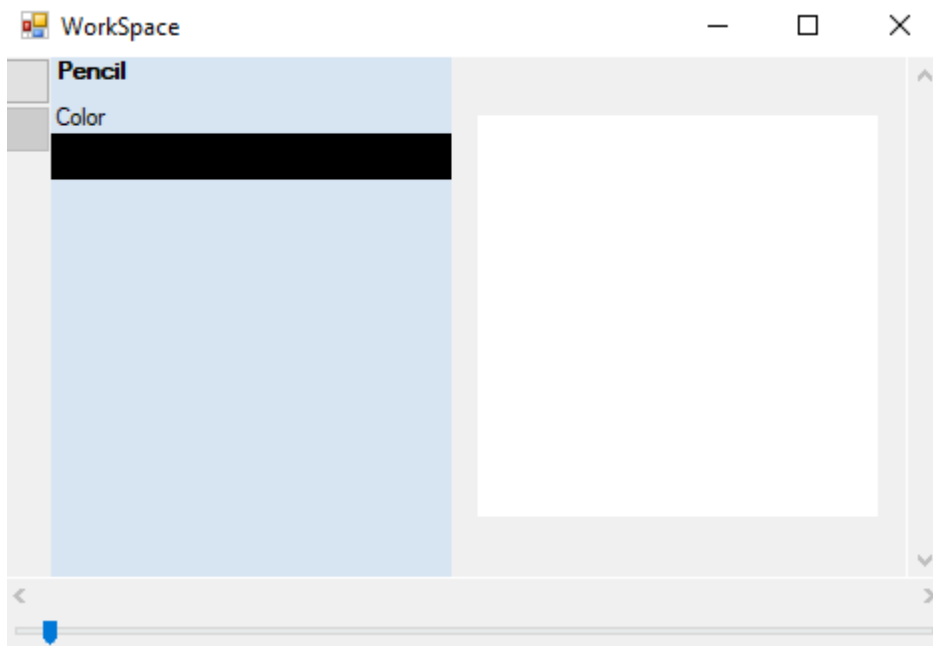
Then, with a check on the display code, hidden properties will not be shown:

```
// doesn't show invisibles
if (property.propertyType == PropertyType.Hidden) {
    continue;
}
```



Tool title visibility fix

The title of the current tool has also been invisible, though this was missed by the stakeholders. The error stemmed from a missing capital, which has now been fixed:



Changing starting tool

Changing the starting tool simply requires the program on startup to set the tool to the first items in the tool list:

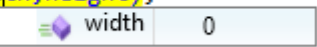
```
currentTool = tools[0];  
ShowTool();
```

So the program now starts with the brush selected.

Fixing crash when resizing window

Currently there is an issue where changing the form size to the size of the properties bar causes a crash. This is because the window is smaller than the available space to put an image in, so the size is interpreted as zero, and is invalid for an image:

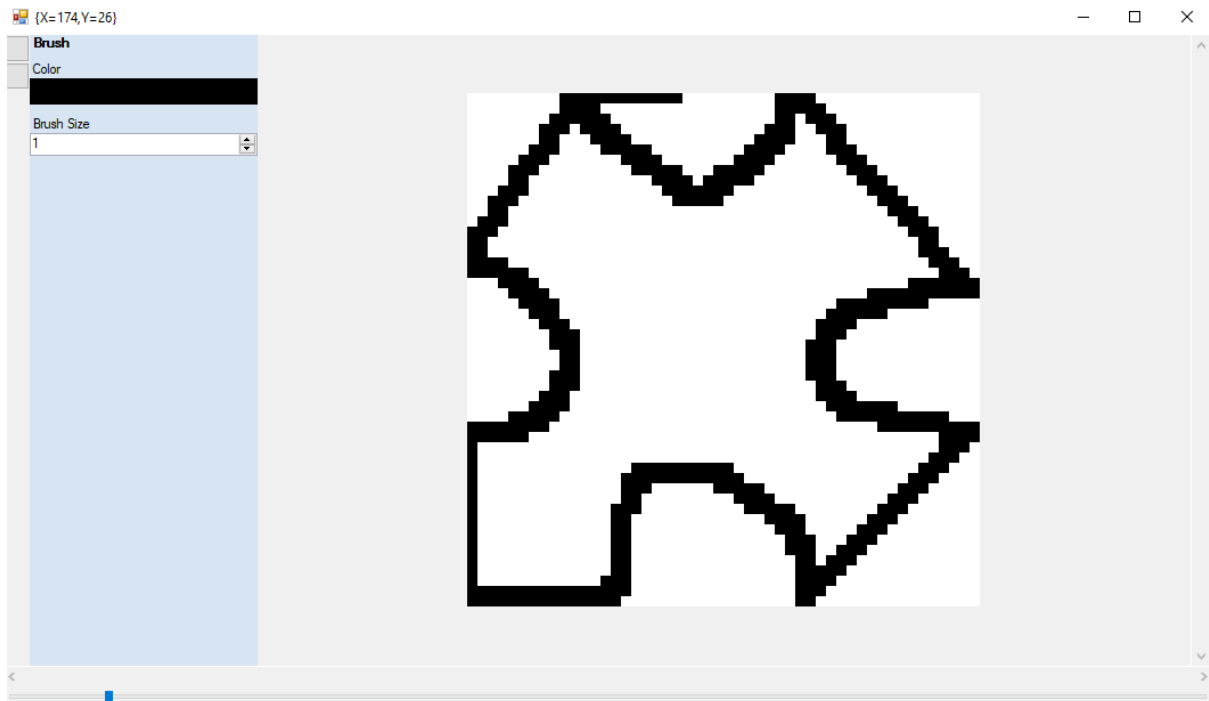
```
Bitmap newImage = new Bitmap(width,height);  
Graphics GFX = Graphics.FromImage(newImage);  
GFX.Clear(SystemColors.Control);
```

A screenshot of a Windows Forms property grid. The 'width' property is selected, and its value is set to 0. The grid has a light blue header and a white body. The 'width' property is in the first column, and its value '0' is in the second column.

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

```
private int _leftPadding, _rightPadding, _topPadding, _bottomPadding;  
  
public int leftPadding {  
    get {  
        return _leftPadding;  
    }  
    set {  
        _leftPadding = value;  
        SetMinimumSize();  
    }  
}  
  
public int rightPadding {  
    get {  
        return _rightPadding;  
    }  
    set {  
        _rightPadding = value;  
        SetMinimumSize();  
    }  
}  
  
public int topPadding {  
    get {  
        return _topPadding;  
    }  
    set {  
        _topPadding = value;  
        SetMinimumSize();  
    }  
}  
  
public int bottomPadding {  
    get {  
        return _bottomPadding;  
    }  
    set {  
        _bottomPadding = value;  
        SetMinimumSize();  
    }  
}  
  
private void SetMinimumSize() {  
    this.MinimumSize = new Size(_leftPadding + _rightPadding + SimpConstants.WINDOWS_RIGHT_BAR_WIDTH + SimpConstants.WINDOWS_LEFT_BAR_WIDTH,  
                                _topPadding + _bottomPadding + SimpConstants.WINDOWS_TOP_BAR_HEIGHT + SimpConstants.WINDOWS_BOTTOM_BAR_HEIGHT);  
}
```

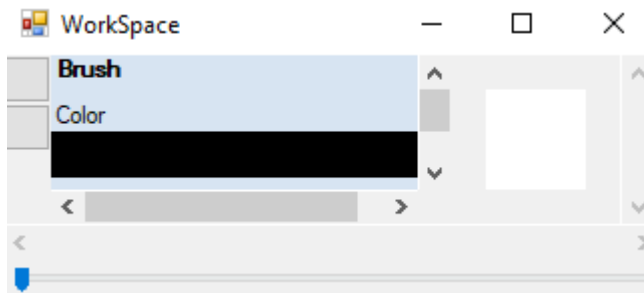

However, after implementing this, a problem arose where when leaving the canvas and coming back on, a line would be drawn from the last location on the canvas to location or re-entry. For example:



In order to fix this, an extra line to update the `oldLocation` constantly makes the program forget about where the line last was on the canvas:

[illegible]

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



So this fixes the error.

Fixing Algorithmic Error

When designing a system for keeping a point inside of a picture box, an error was appearing where under some circumstances an out of bounds location was passed (even though there were checks in all 4 dimensions). The error turned out to be:

```
void PicColourSquareMouseMove(object sender, MouseEventArgs e)
{
    if (e.Button != MouseButtons.None) {
        Point newPoint = e.Location;
        if (e.Location.X < 0) {
            newPoint = new Point(0, e.Location.Y);
        }
        if (e.Location.X >= picColourSquare.Width) {
            newPoint = new Point(picColourSquare.Width, e.Location.Y);
        }
        if (e.Location.Y < 0) {
            newPoint = new Point(e.Location.X, 0);
        }
        if (e.Location.Y >= picColourSquare.Height) {
            newPoint = new Point(e.Location.X, picColourSquare.Height);
        }
        PicColourSquareClick(sender, new MouseEventArgs(e.Button, e.Clicks, newPoint.X, newPoint.Y, e.Delta));
    }
}
```

The X coord is fixed here...

However here, the potentially invalid X coord is used

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

Implementing image functions

The code for implementing undo in a brush has been implemented, however it has an issue. The system needs to make sure that no duplicate points are added, however the system does not quite work:

```
if (!setPixels.ContainsKey(stampPoint)) { // if this pixel is not yet in the history
    setPixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
    if (!strokePixels.ContainsKey(stampPoint)) {
        MessageBox.Show(stampPoint.ToString());
        strokePixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
    }
}
```

The reason being that because stampPoint (a FilePoint) is an object, its **hash** will be compared to any others in the list, not its location. This means that any equal points with a different hash can still be added to the list.

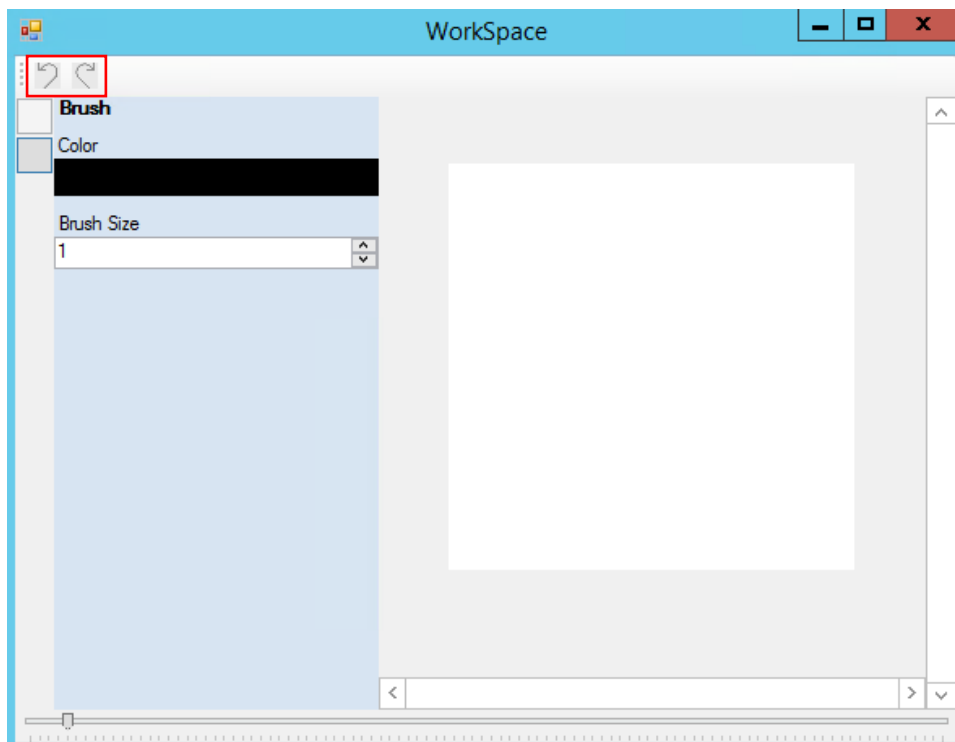
To fix this, a list of primitive strings can be used instead to check whether an item is in the list:

```
if (!setHashes.Contains(stampPoint.ToString())) { // if this pixel is not yet in the history
    setPixels.Add(stampPoint, currentColor);
    setHashes.Add(stampPoint.ToString());
    if (!strokeHashes.Contains(stampPoint.ToString())) {
        strokePixels.Add(stampPoint, myWorkspace.image.GetPixel(stampPoint));
        strokeHashes.Add(stampPoint.ToString());
    }
}
```

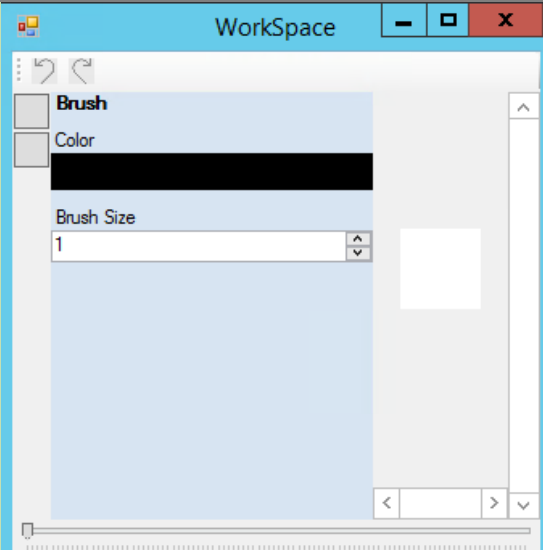
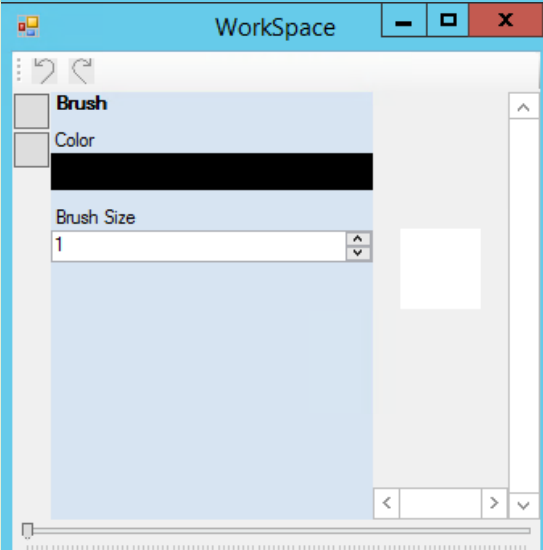
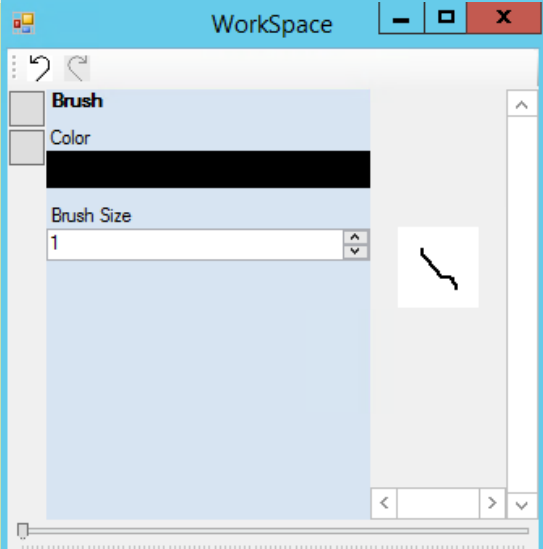
Now the 'hash' (a string representation) will be compared instead

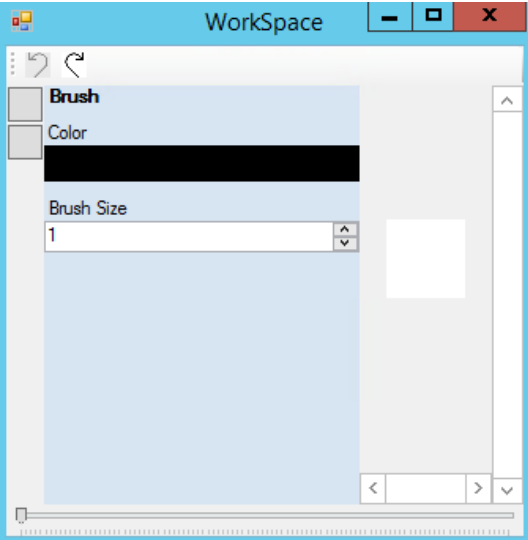
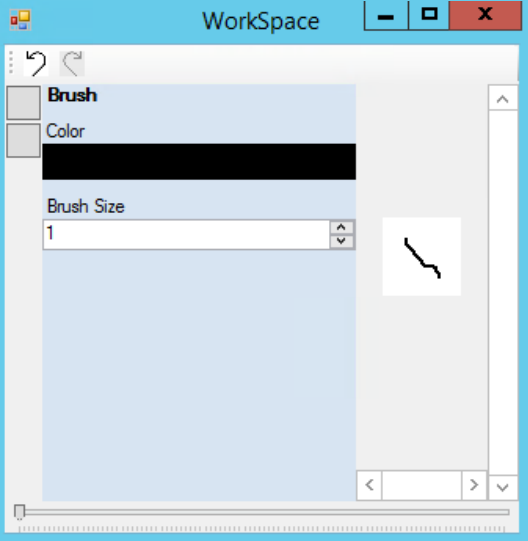
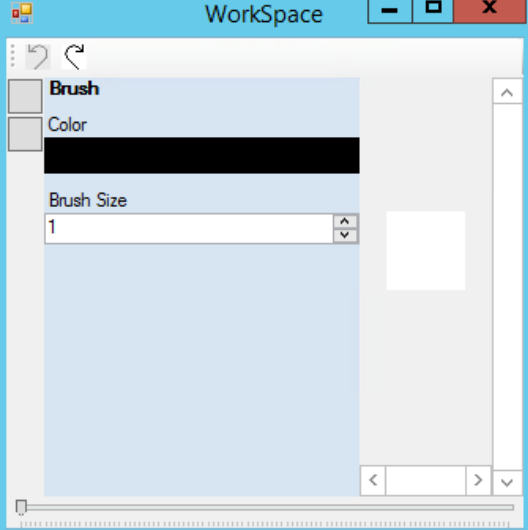
Now, the literal string will be compared, and this is much more effective.

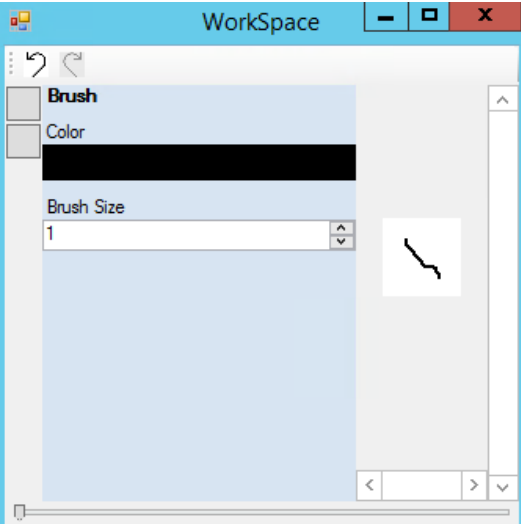
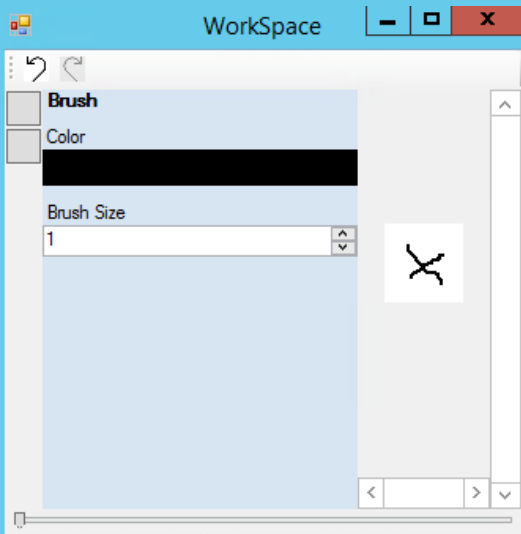
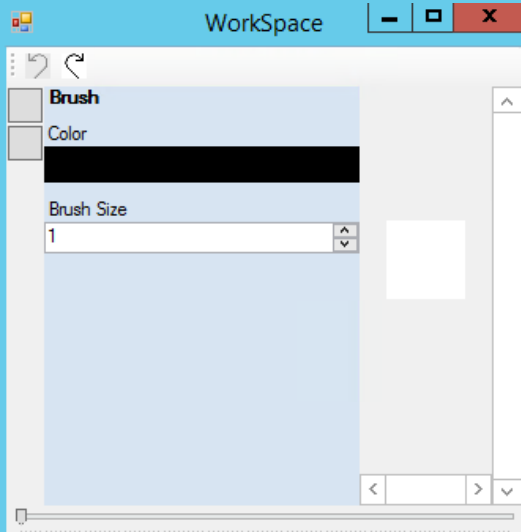
After adding some undo and redo buttons, it is now time to do the unit test:

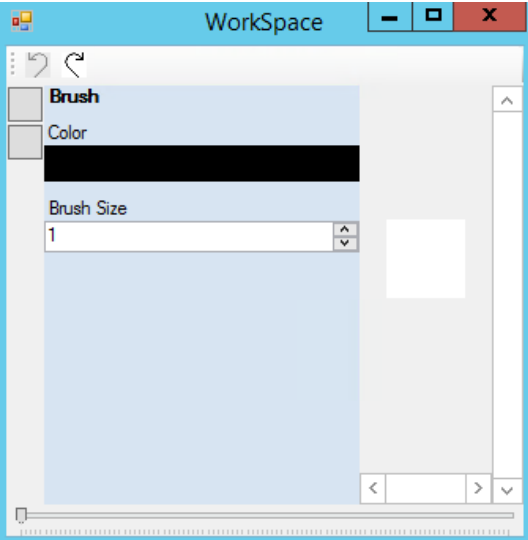
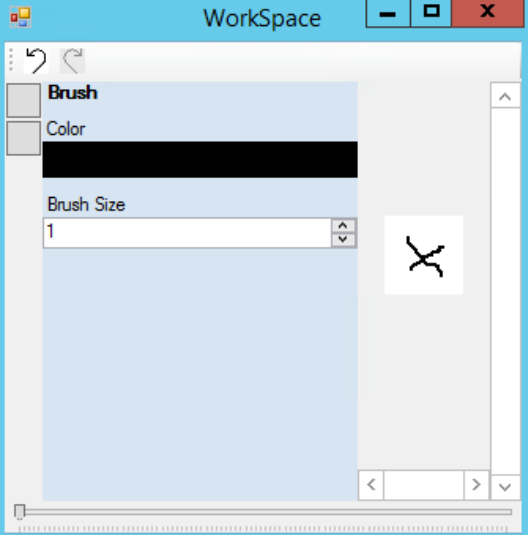


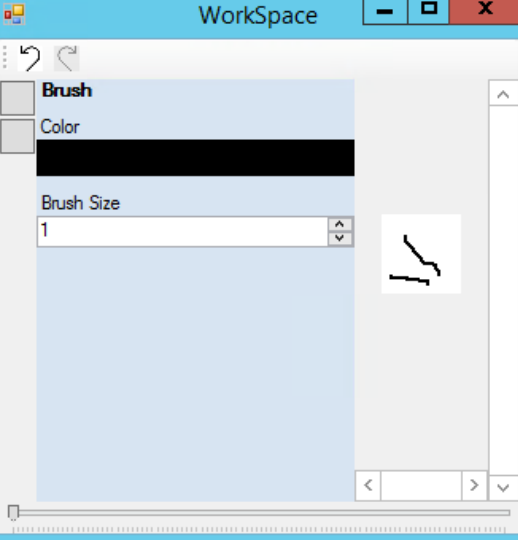
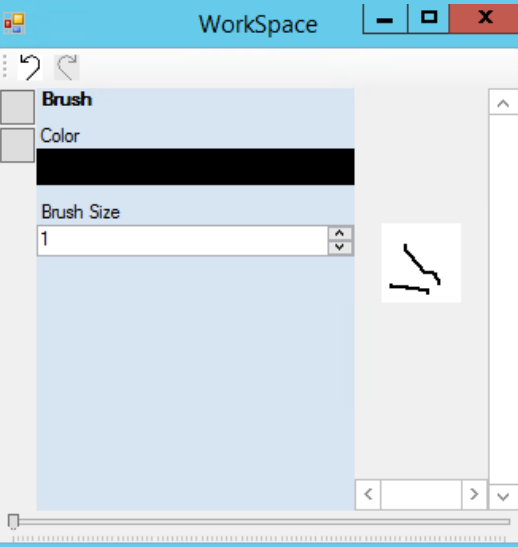
23/11/2019 Undo Unit Test

Test	ID	Expected Result	Actual Result	Comment
Undo an action (on default image)	1	Cannot be done		The undo button is correctly disabled on startup
Redo an action (on default image)	2	Cannot be done		The redo button is correctly disabled on startup
Perform an action	3	Action is performed onto image		The action can still be performed and the undo button becomes enabled

Undo the action	4	Action is undone		The action is reverted and the redo button is now enabled.
Redo the action	5	Action is redone		The action is redone and the undo button is now enabled
Undo the action	6	Action is undone		The action is undone again

Redo the action	7	Action is redone	 <p>The screenshot shows the 'Workspace' window with a toolbar at the top containing undo and redo icons. Below the toolbar, there are settings for 'Brush' (a square icon), 'Color' (a black bar), and 'Brush Size' (a dropdown menu set to '1'). The main canvas on the right displays a single black brush stroke.</p>	The action is redone again
Perform another action	8	Action is done	 <p>The screenshot shows the 'Workspace' window with the same settings as the previous one. The main canvas now displays two black brush strokes: a single line and an 'X' shape.</p>	A new action is performed on top of the old one
Undo both actions	9	Both actions redone	 <p>The screenshot shows the 'Workspace' window with the same settings. The main canvas is now blank, indicating that both previous actions have been undone.</p>	Both actions can be undone by pressing undo twice

Undo an action	10	Cannot be done		Undo is currently disabled after undoing the first two actions so cannot be pressed
Redo both actions	11	Both actions redone		Both actions are put back onto the image
Redo an action	12	Cannot be done		The redo button is disabled after redoing twice

<p><i>Undo an action, and preform a new action</i></p>	<p>13 New action is performed on top of old action</p>		<p>The action is preformed successfully</p>
<p><i>Redo an action</i></p>	<p>14 Cannot be done</p>		<p>The redo button is disabled as there is now nothing to redo</p>

24/11/2019 Layers

Implementation

The Layer class has been implemented in accordance to Algorithms 3.9, 3.10 and 3.11. The algorithm for drawing a pixel was slightly edited:

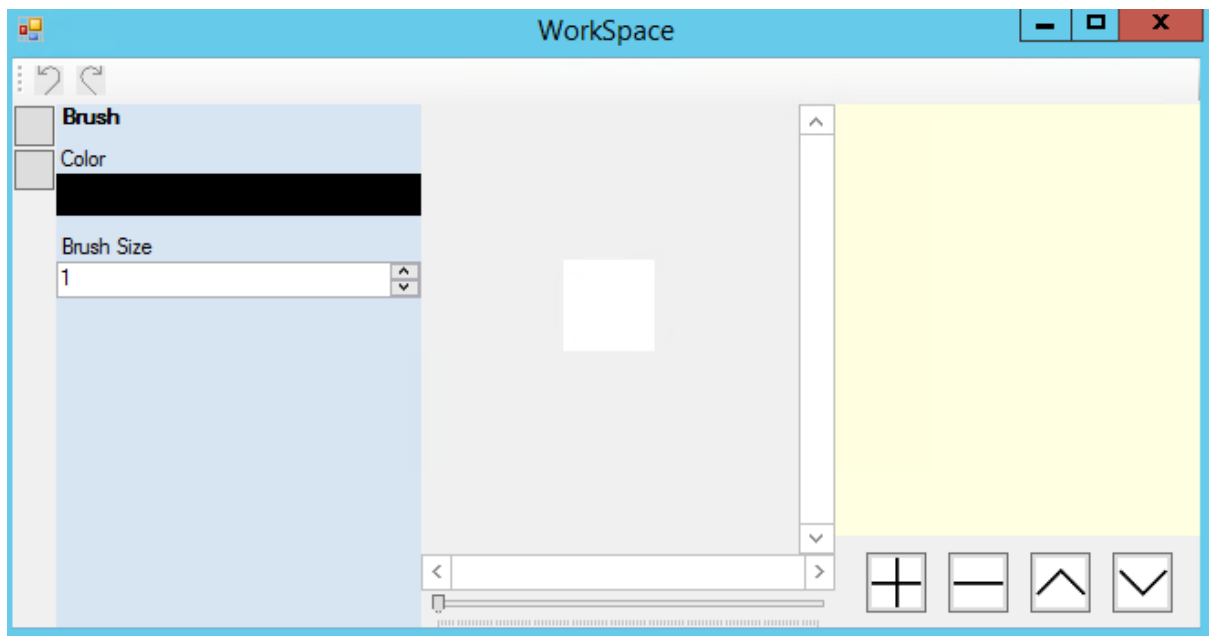
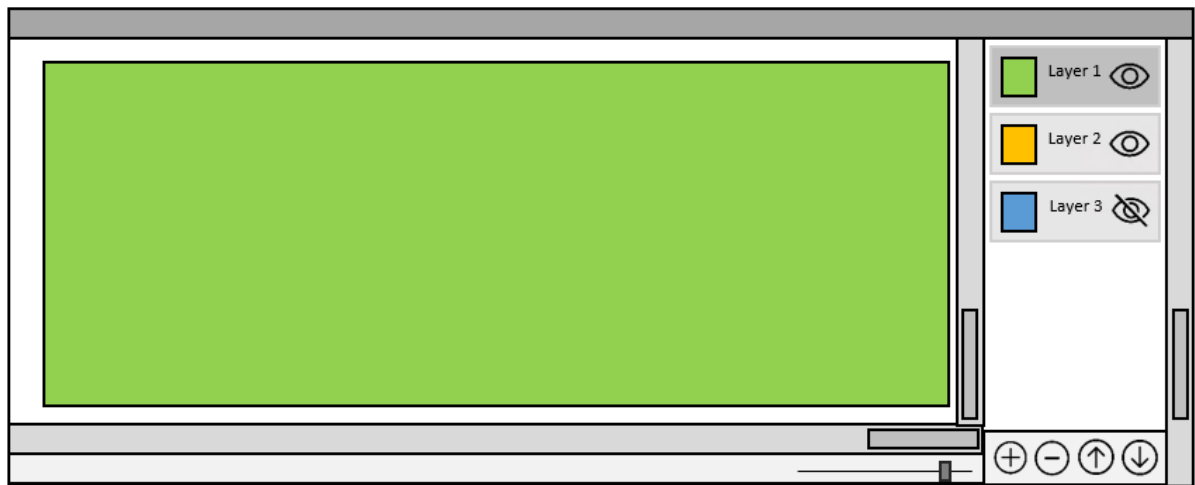
```
DisplayPixel(x,y) {  
    FOR EACH layer IN layers  
        IF layer.pixels[x,y] is transparent  
            STOP REPEAT  
        ELSE  
            //Draw pixel on image (as before)  
        END IF  
    NEXT  
}
```

```
foreach (Layer layer in layers) {  
    // repeats through layers until a solid pixel is found  
    if (layer.pixels[x,y].Color != Color.Transparent) {  
        // Draws a rectangle using colour of current pixel, X  
        GFX.FillRectangle(layer.pixels[x,y],currentPoint.displ  
        break; // a pixel has been drawn - draw no more here  
    }  
}
```

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

Layer Selector Design

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



Though the actual layer displaying will be implemented shortly.

27/11/2019 Implementing Layer Buttons

Layer Preview Rendering Optimisation

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

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

```
private void UpdateLayersSelector() {  
    foreach (Layer layer in image.layers) {  
        System.Drawing.Image newImage = new System.Drawing.Bitmap(40,40);  
        Graphics GFX = Graphics.FromImage(newImage);  
        for (int x = 0; x < 40; x++) {  
            for (int y = 0; y < 40; y++) {  
                int imageX = (x * image.fileWidth) / 40;  
                int imageY = (y * image.fileHeight) / 40;  
                GFX.FillRectangle(layer.pixels[imageX,imageY],x,y,1,1);  
            }  
        }  
        previewImages[layer].Image = newImage;  
    }  
}
```

However this is quite inefficient, as it redraws every layer each time, even when only one layer is updated. This can be resolved by replacing the foreach with just the currentLayer.

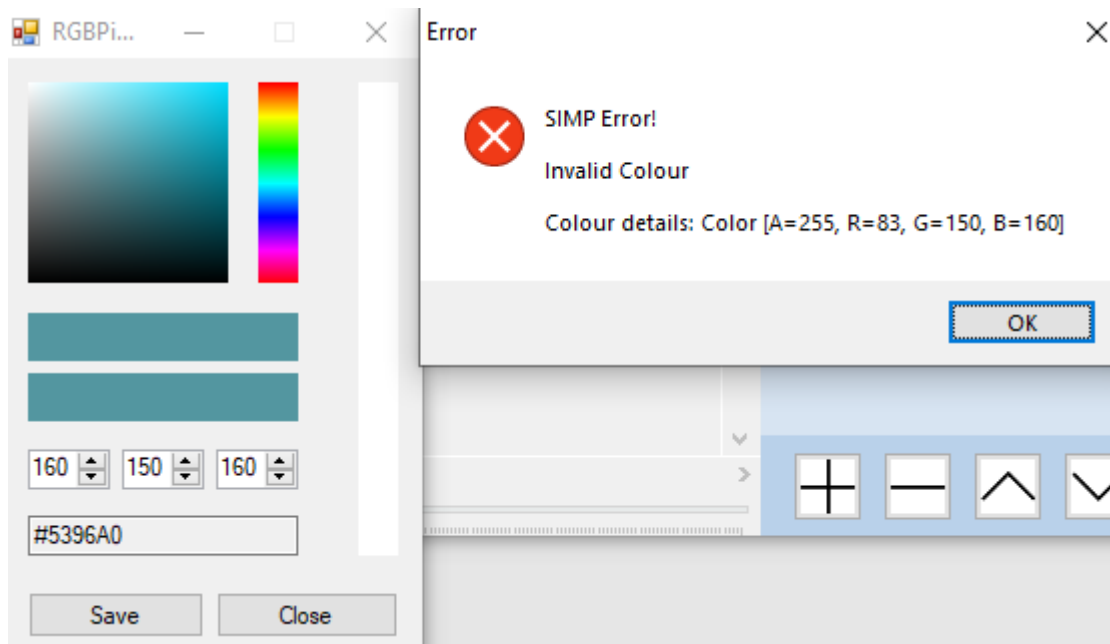
```
private void UpdateLayersSelector(bool full) {  
    if (full) {  
        foreach (Layer layer in image.layers) {  
            UpdateLayer(layer);  
        }  
    } else {  
        UpdateLayer(image.currentLayer);  
    }  
}
```

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

Colour Interpretation Error

An issue that many stakeholders had reported was a crash when using the colour picker. The error appeared to stem from some colour values in the RGB conversions, however as the program force closed it was never clear what the problem colour was.

So, to combat this, an error handler was implemented, asking the user to report any such crash causing colours. Soon enough, a problem colour was isolated:



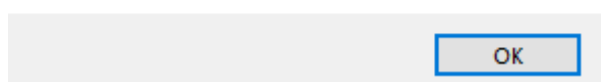
The colour is R160, G150, B160. This was able to be reported due to the error handler. Now this problem can be investigated.

When attempting to run a conversion on this colour from RGB to HSV, the problem becomes apparent:

```
MessageBox.Show(SIMPRGB.RGBtoHSV(160,150,160).ToString());
```

×

[HSVColor H=-60, S=0.0625, V=0.627450980392157]



The Hue is -60, which is clearly invalid.

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

```
if (Δ == 0) {  
    h = 0;  
} else if (Cmax == R) {  
    h = 60 * ((G-B)/Δ)%6;  
} else if (Cmax == G) {  
    h = 60 * ((B-R)/Δ)+2;  
} else if (Cmax == B) {  
    h = 60 * ((R-G)/Δ)+4;  
}
```

Any of these subtractions could introduce a negative sign, if $R > G > B$ for example.

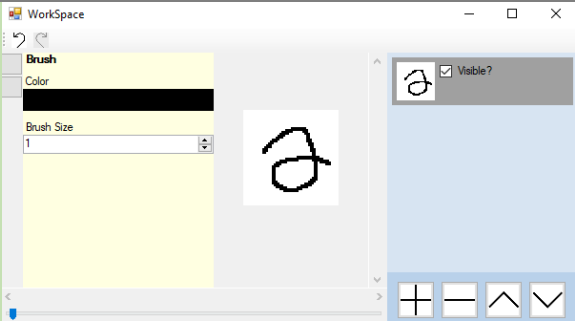
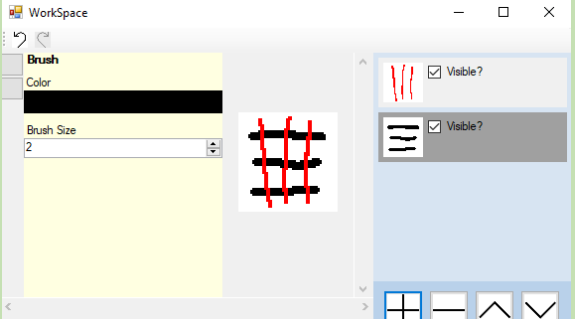
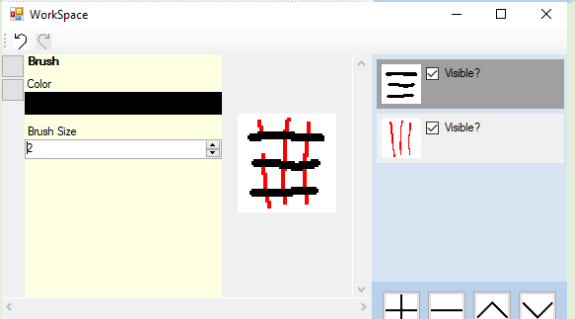
To fix this, as Hue is interpreted as an 'angle' (between 0° - 360°), 360° can be added to 'wrap' it around if necessary, similar to how a turn of -40° is equal to a turn of 320°

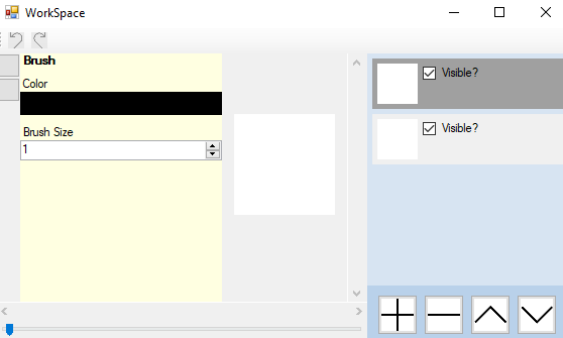
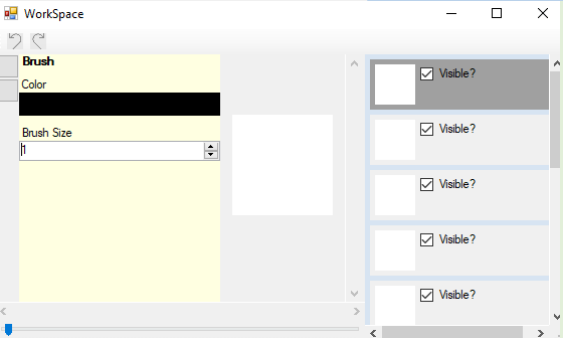

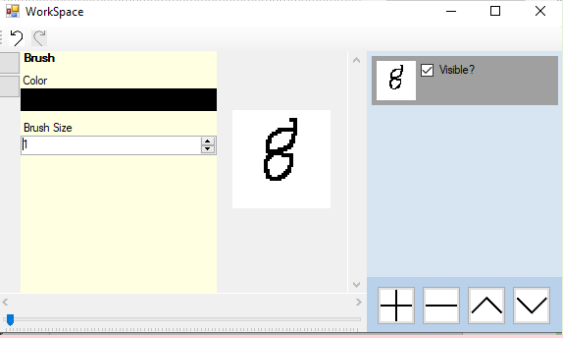
```
if (h<0) {  
    h += 360;  
}
```

This means the problem colour can now successfully be manipulated:



28/11/2019 Layer Unit Test

Test	ID	Expected Result	Actual Result	Comment
Select the top layer and draw onto it	1	There is drawing on layer		The layer can be drawn upon and the preview updates
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one	System.ArgumentOutOfRangeException	The program crashes as there is no prevention of removing only layer
Select lower layer and draw onto it at same location as top layer	3	There is drawing on lower layer, but it is obviously below		The program obviously draws the lower layer below the red one
Move lower layer up	4	The lower layers moves on top of previous top layer		The program now puts the thicker lines on top
Move the highest layer up	5	Should be impossible as it cannot go up	System.ArgumentOutOfRangeException	The program does not have any checking for this
Move the lowest layer down	6	Should be impossible as it cannot go down	System.ArgumentOutOfRangeException	The program also does not check for this

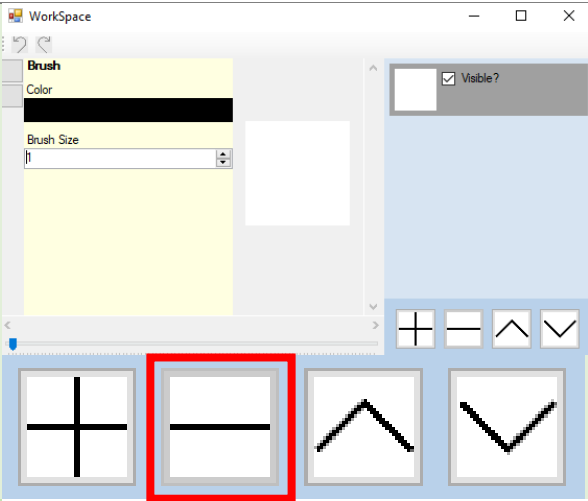
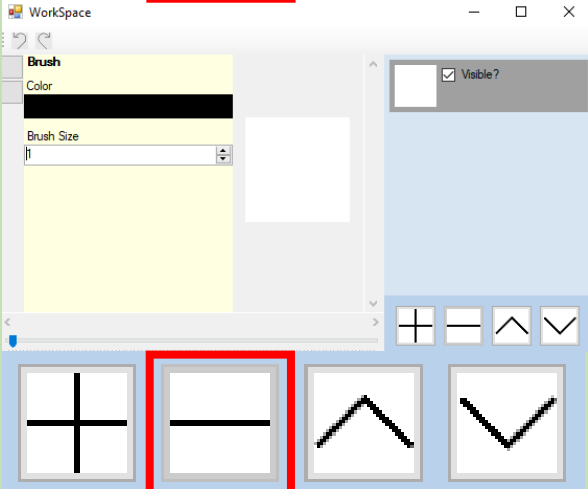
Add a new layer	7	A new (empty) layer is created		The program does this
Add 10 new layers	8	Many new layers are added		The program is able to create a large amount of layers
Layers are labelled then randomly rearranged	9	The layers are moved		The program does not crash when moving about this large amount of layers
The 11 layers are deleted	10	The layers disappear in the order in which they are added in the list		The program is able to then remove all of the layers
The final layer is deleted	11	Prevented as there is only one layer	System.ArgumentOutOfRangeException	There is still no check for this

Fixing Error 2 & 11

Test	ID	Expected Result	Actual Result	Comment
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one	System.ArgumentOutOfRangeException	The program crashes as there is no prevention of removing only layer
The final layer is deleted	11	Prevented as there is only one layer	System.ArgumentOutOfRangeException	There is still no check for this

In order to fix this error, upon refreshing the amount of layers can be checked:

```
btnRemoveLayer.Enabled = image.layers.Count > 1;
```

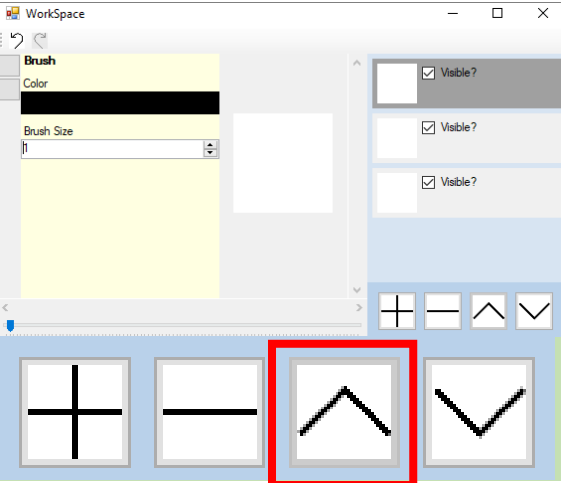
Test	ID	Expected Result	Actual Result	Comment
Attempt to delete the layer	2	The layer cannot be deleted as it is the only one		The remove button is now disabled
The final layer is deleted	11	Prevented as there is only one layer		The remove button is now disabled

Fixing Error 5

Test	ID	Expected Result	Actual Result	Comment
Move the highest layer up	5	Should be impossible as it cannot go up	System.ArgumentOutOfRangeException	The program does not have any checking for this

To fix this, a line of code can be implemented to check where the current layer is:

```
btnLayerUp.Enabled = image.layers.IndexOf(image.currentLayer) != 0;
```

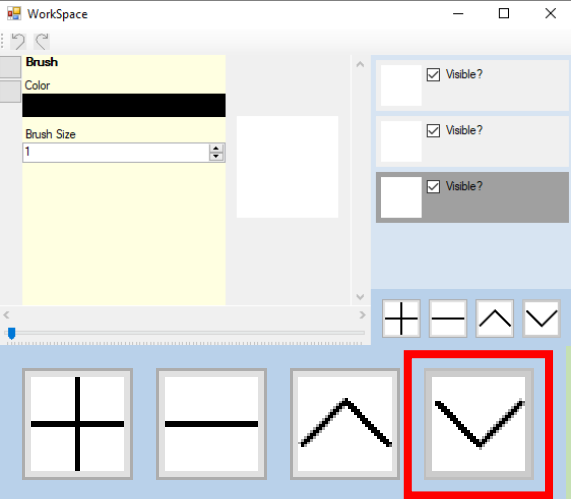
Test	ID	Expected Result	Actual Result	Comment
Move the highest layer up	5	Should be impossible as it cannot go up		The program now disables the button when needed

Fixing Error 6

Test	ID	Expected Result	Actual Result	Comment
Move the lowest layer down	6	Should be impossible as it cannot go down	System.ArgumentOutOfRangeException	The program also does not check for this

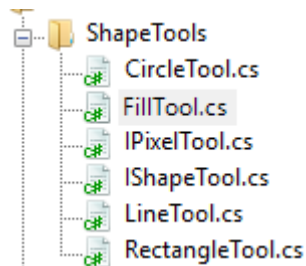
This can be also fixed with a small line of code:

```
btnLayerDown.Enabled = image.layers.IndexOf(image.currentLayer) != (image.layers.Count - 1);
```

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

29/11/2019 Shape framework

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



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

```
public override void HandleMouseMove(FilePoint oldLocation, FilePoint newLocation) {...}
public override void HandleMouseClicked(FilePoint clickLocation, System.Windows.Forms.MouseButtons button) {...}
public override void HandleMouseUp(FilePoint clickLocation, System.Windows.Forms.MouseButtons button) {...}
public override void HandleMouseDown(FilePoint clickLocation, System.Windows.Forms.MouseButtons button) {...}
public override void AddShapePoint(int x, int y) {...}
```

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

```
public override void GenShape()
{
    throw new NotImplementedException();
}
```

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

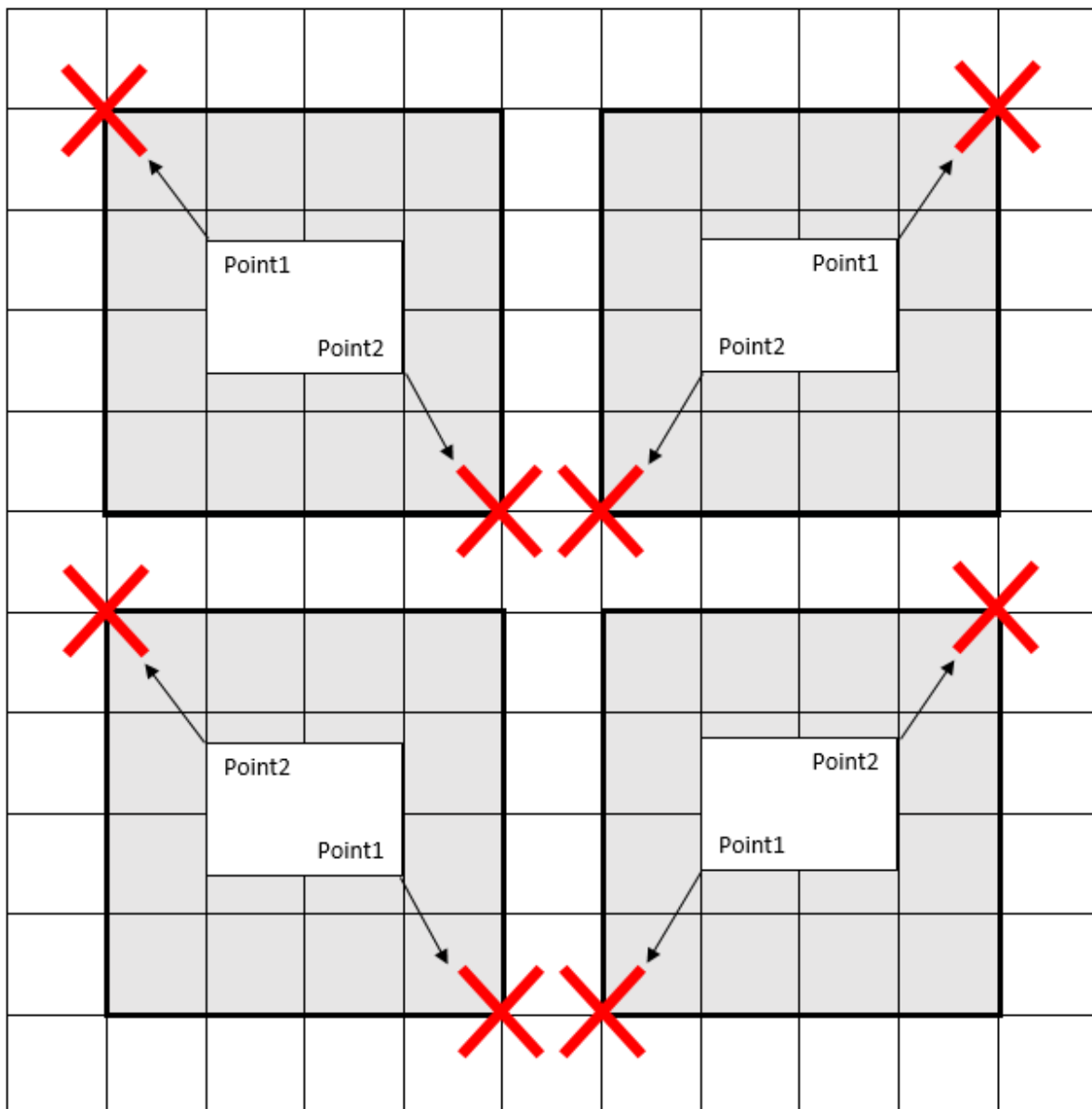
```
internal void ResolvePoints() {
    int highX, highY, lowX, lowY = 0;

    if (point1.fileX > point2.fileX) {
        highX = point1.fileX;
        lowX = point2.fileX;
    } else {
        highX = point2.fileX;
        lowX = point1.fileX;
    }

    if (point1.fileY > point2.fileY) {
        highY = point1.fileY;
        lowY = point2.fileY;
    } else {
        highY = point2.fileY;
        lowY = point1.fileY;
    }

    point1 = new FilePoint(lowX, lowY);
    point2 = new FilePoint(highX, highY);
}
```

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



However this does not apply for the LineTool, as the line drawing algorithm will already resolve points by itself. So thus the LineTool class needs to **override** the existing method in order to disable it:

```
internal override void ResolvePoints() {
    // do nothing
}
```

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

```
internal override void GenShape() {
    for (int x = point1.fileX; x < point2.fileX; x++) {
        for (int y = point1.fileY; y < point2.fileY; y++) {
            shapePoints.Add(new FilePoint(x,y));
        }
    }
}
```

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

$$\frac{(x - (x' + 0.5))^2}{w^2} + \frac{(y - (y' + 0.5))^2}{h^2} \leq 1$$

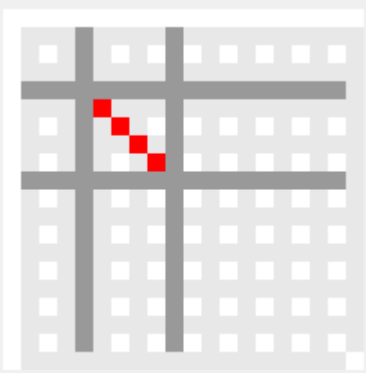
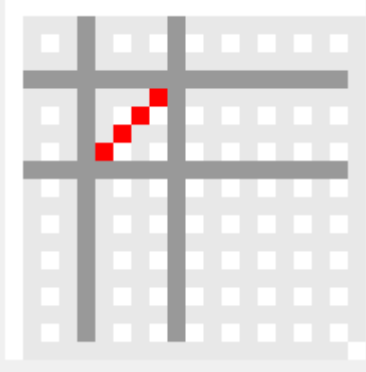
To make this into the diamond formula, the square is switched for an Absolute operation:

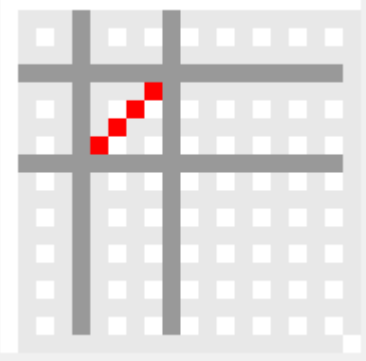
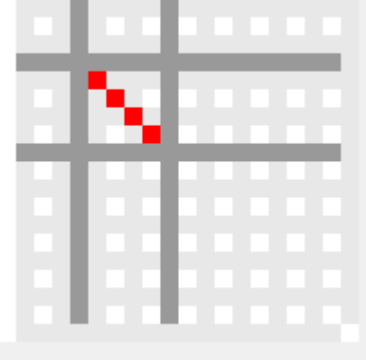
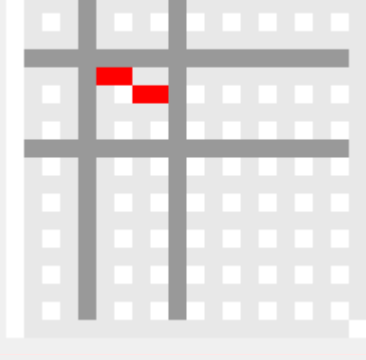
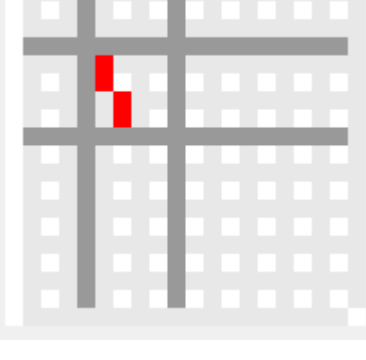
$$\frac{|x - (x' + 0.5)|}{w} + \frac{|y - (y' + 0.5)|}{h} \leq 1$$

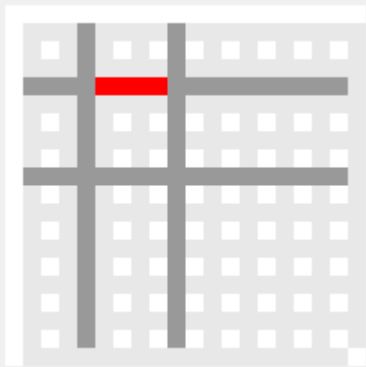
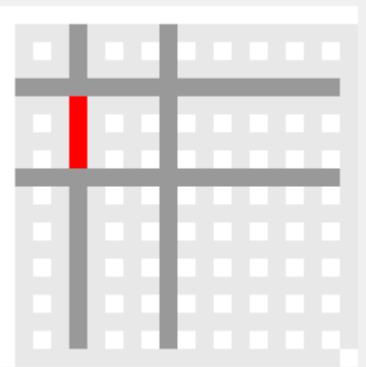
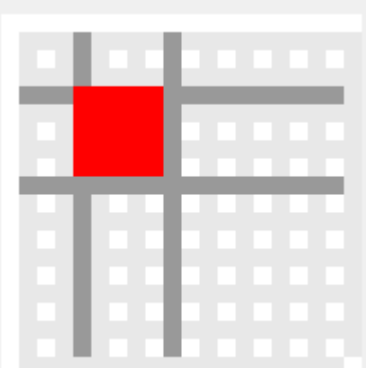
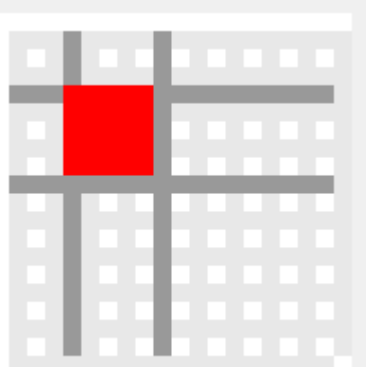
Thus the shape tools all have formulas, and can be implemented.

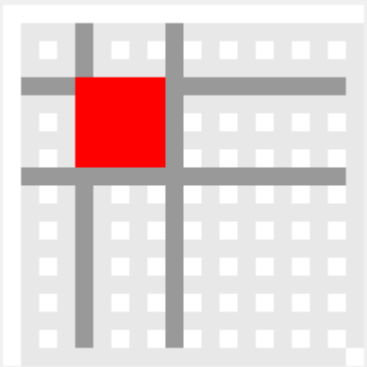
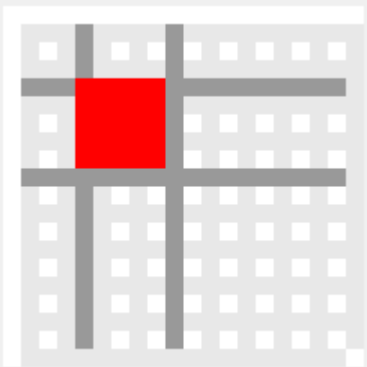
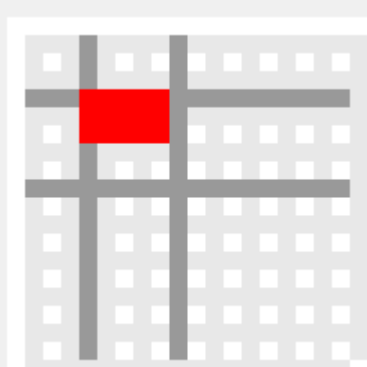
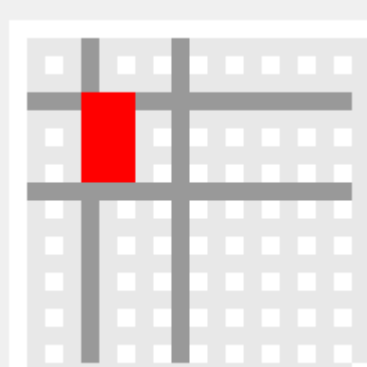
After this, it is now time to do the full section testing.



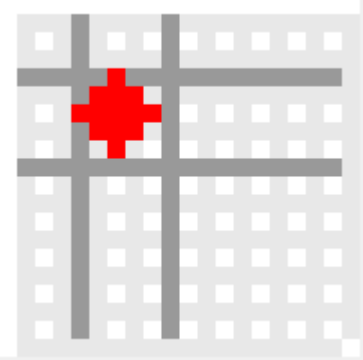

02/12/2019 Shape Unit Test

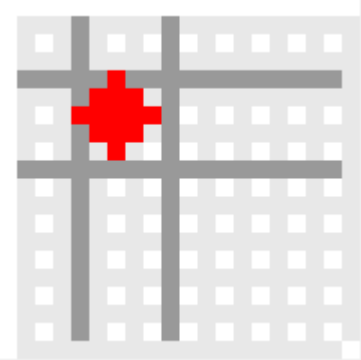
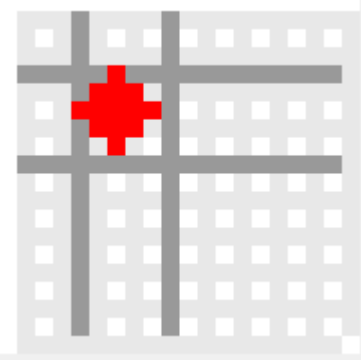
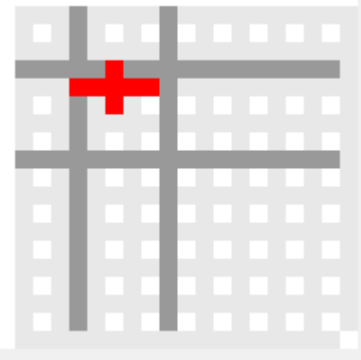
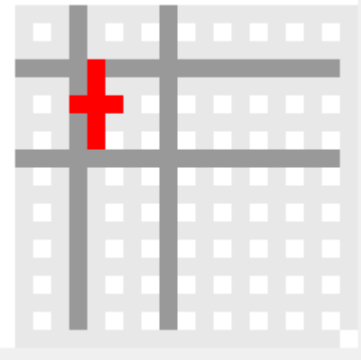
Test	ID	Expected Result	Actual Result	Comment
Create line with points (5,5) & (10,10)	1	A line is drawn from (5,5) to (10,10)		The line does not include the two starting points
Create line with points (5,10) & (10,5)	2	A line is drawn from (5,10) to (10,5)		The line does not include the two starting points

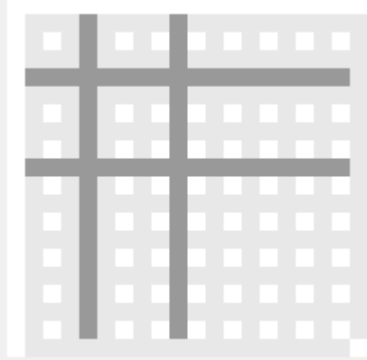

Create line with points (10,5) & (5,10)	3	A line is drawn from (10,5) to (5,10)		The line does not include the two starting points
Create line with points (10,10) & (5,5)	4	A line is drawn from (10,10) to (5,5)		The line does not include the two starting points
Create line with points (5,5) & (10,8)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

Create line with points (5,5) & (10,5)	7	A single row is drawn		The line does not include the two starting points
Create line with points (5,5) & (5,10)	8	A single column is drawn		The line does not include the two starting points
Create rectangle with points (5,5) & (10,10)	9	A rectangle is drawn from (5,5) to (10,10)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,10) & (10,5)	10	A rectangle is drawn from (5,10) to (10,5)		The rectangle has not included the coords of the lower point in its calculations

Create rectangle with points (10,5) & (5,10)	11 A rectangle is drawn from (10,5) to (5,10)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (10,10) & (5,5)	12 A rectangle is drawn from (10,10) to (5,5)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (10,8)	13 A shorter rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (8,10)	14 A thinner rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations

Create rectangle with points (5,5) & (10,5)	15	A single row is drawn		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (5,10)	16	A single column is drawn		The rectangle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (10,10)	17	A circle is drawn from (5,5) to (10,10)		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,10) & (10,5)	18	A circle is drawn from (5,10) to (10,5)		The circle has not included the coords of the lower point in its calculations

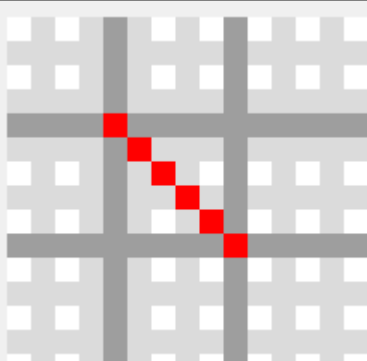
Create circle with points (10,5) & (5,10)	19	A circle is drawn from (10,5) to (5,10)		The circle has not included the coords of the lower point in its calculations
Create circle with points (10,10) & (5,5)	20	A circle is drawn from (10,10) to (5,5)		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (10,8)	21	A shorter circle is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (8,10)	22	A thinner circle is drawn		The circle has not included the coords of the lower point in its calculations

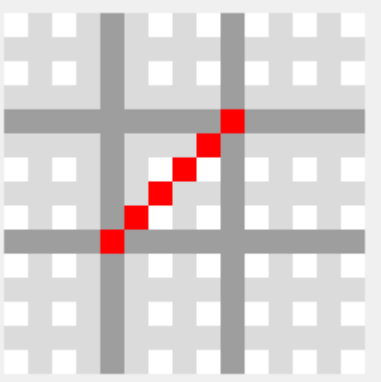
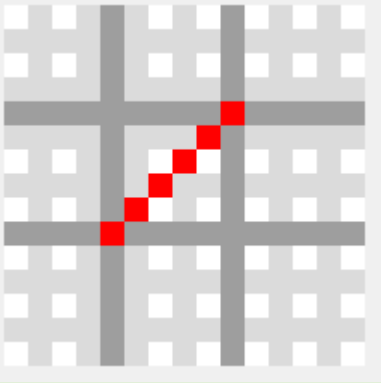
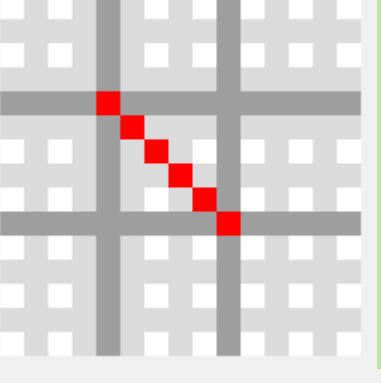
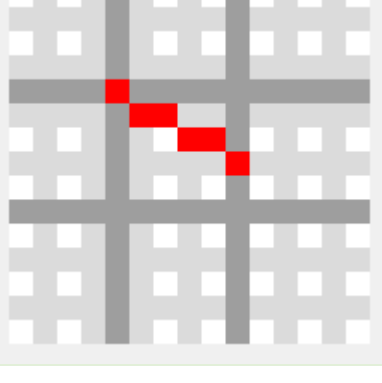
Create circle with points (5,5) & (10,5)	23	A single row is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (5,10)	24	A single column is drawn		The circle has not included the coords of the lower point in its calculations

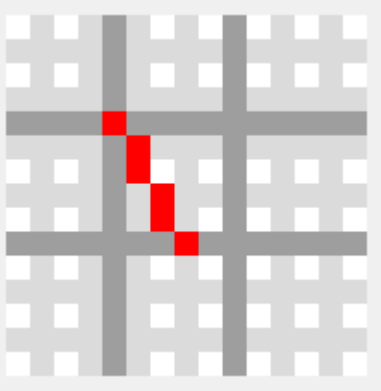

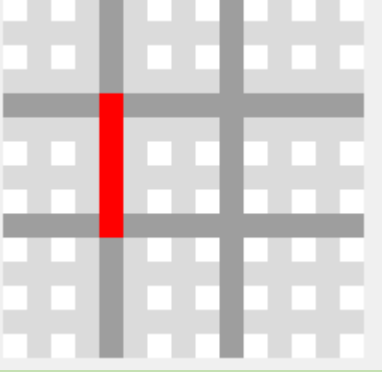
Fixing Error #1-#8

To fix these errors, the program can be made to forcefully set the start and end points of the line:

```
// makes the the start and end points are part of the line
AddShapePoint(point1.fileX,point1.fileY);
AddShapePoint(point2.fileX,point2.fileY);
...
```

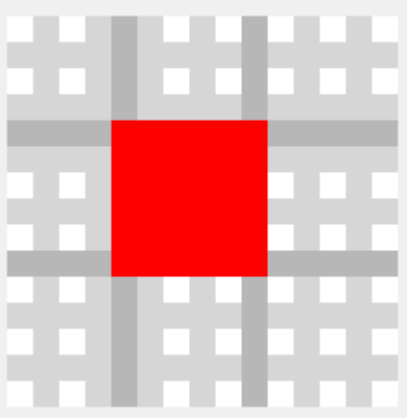
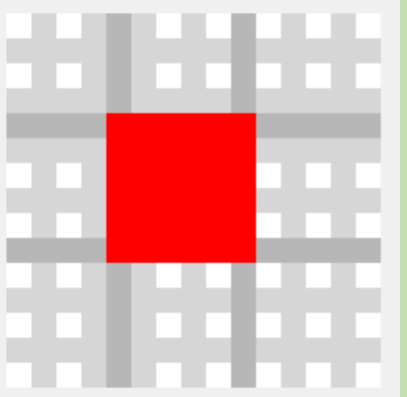
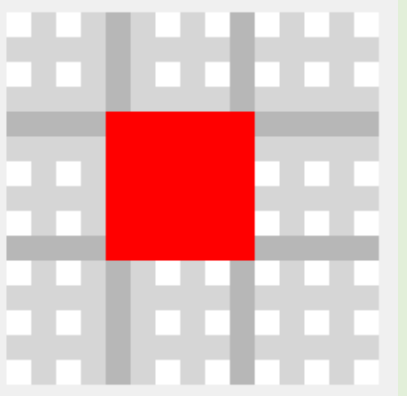
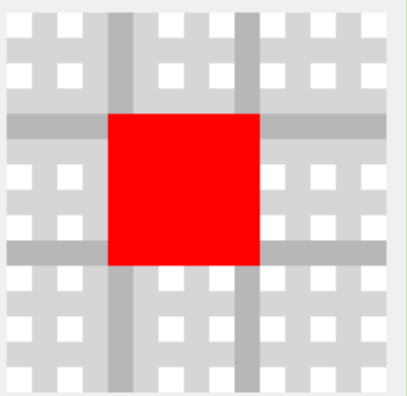
Test	ID	Expected Result	Actual Result	Comment
Create line with points (5,5) & (10,10)	1	A line is drawn from (5,5) to (10,10)		The line does not include the two starting points

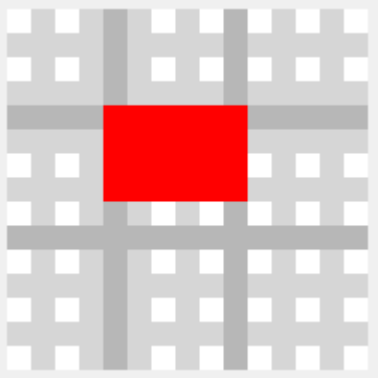


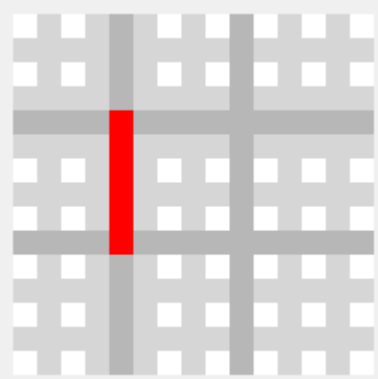
Create line with points (5,10) & (10,5)	2	A line is drawn from (5,10) to (10,5)		The line does not include the two starting points
Create line with points (10,5) & (5,10)	3	A line is drawn from (10,5) to (5,10)		The line does not include the two starting points
Create line with points (10,10) & (5,5)	4	A line is drawn from (10,10) to (5,5)		The line does not include the two starting points
Create line with points (5,5) & (10,8)	5	A shorter line is drawn		The line does not include the two starting points

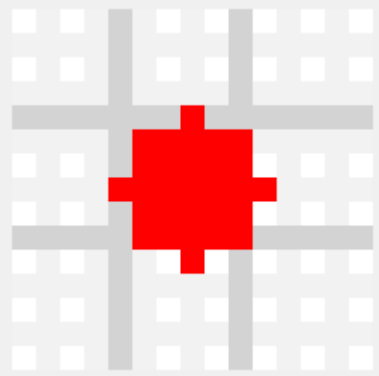
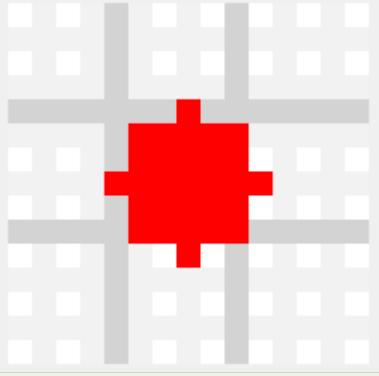
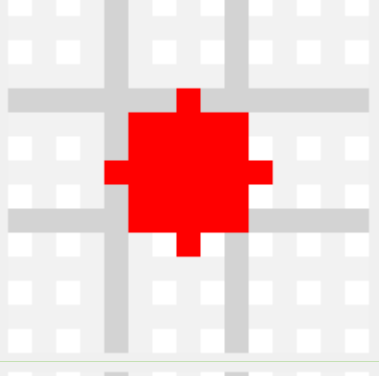
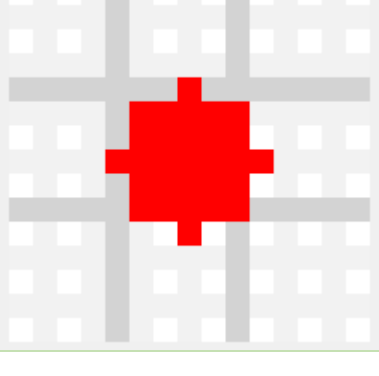
<p>Create line with points (5,5) & (8,10)</p>	6	A thinner line is drawn		The line does not include the two starting points
<p>Create line with points (5,5) & (10,5)</p>	7	A single row is drawn		The line does not include the two starting points
<p>Create line with points (5,5) & (5,10)</p>	8	A single column is drawn		The line does not include the two starting points

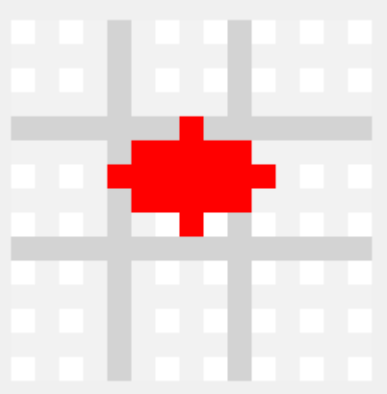
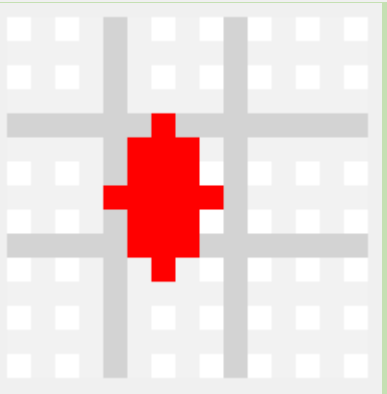
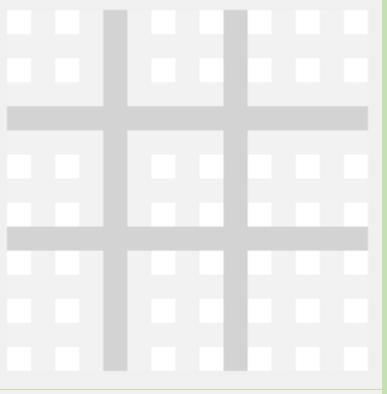
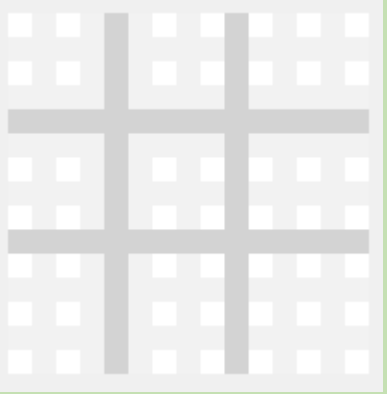
Fixing Error #9-#24

To fix this error, the terminator for the for loop can now become \geq instead of $>$

Test	ID	Expected Result	Actual Result	Comment
Create rectangle with points (5,5) & (10,10)	9	A rectangle is drawn from (5,5) to (10,10)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,10) & (10,5)	10	A rectangle is drawn from (5,10) to (10,5)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (10,5) & (5,10)	11	A rectangle is drawn from (10,5) to (5,10)		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (10,10) & (5,5)	12	A rectangle is drawn from (10,10) to (5,5)		The rectangle has not included the coords of the lower point in its calculations

Create rectangle with points (5,5) & (10,8)	13 A shorter rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (8,10)	14 A thinner rectangle is drawn		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (10,5)	15 A single row is drawn		The rectangle has not included the coords of the lower point in its calculations
Create rectangle with points (5,5) & (5,10)	16 A single column is drawn		The rectangle has not included the coords of the lower point in its calculations

Create circle with points (5,5) & (10,10)	17	A circle is drawn from (5,5) to (10,10)		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,10) & (10,5)	18	A circle is drawn from (5,10) to (10,5)		The circle has not included the coords of the lower point in its calculations
Create circle with points (10,5) & (5,10)	19	A circle is drawn from (10,5) to (5,10)		The circle has not included the coords of the lower point in its calculations
Create circle with points (10,10) & (5,5)	20	A circle is drawn from (10,10) to (5,5)		The circle has not included the coords of the lower point in its calculations

Create circle with points (5,5) & (10,8)	21	A shorter circle is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (8,10)	22	A thinner circle is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (10,5)	23	A single row is drawn		The circle has not included the coords of the lower point in its calculations
Create circle with points (5,5) & (5,10)	24	A single column is drawn		The circle has not included the coords of the lower point in its calculations

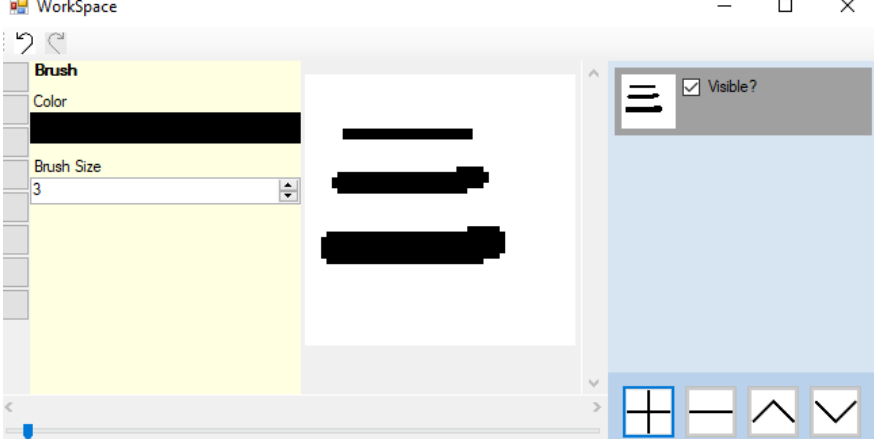
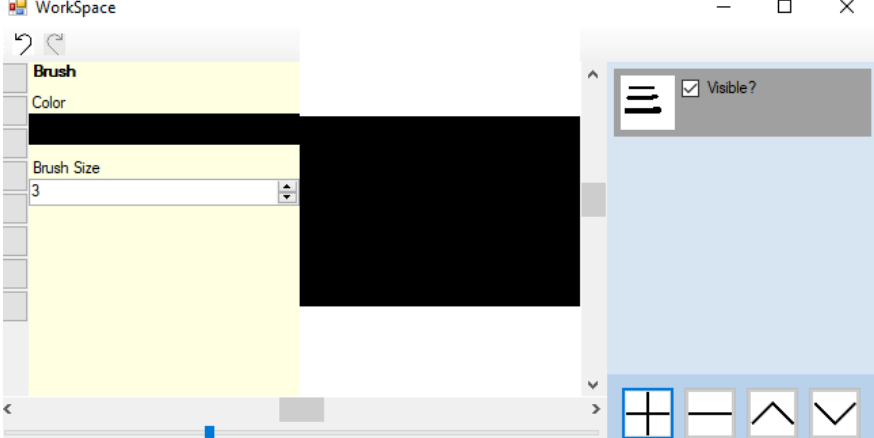
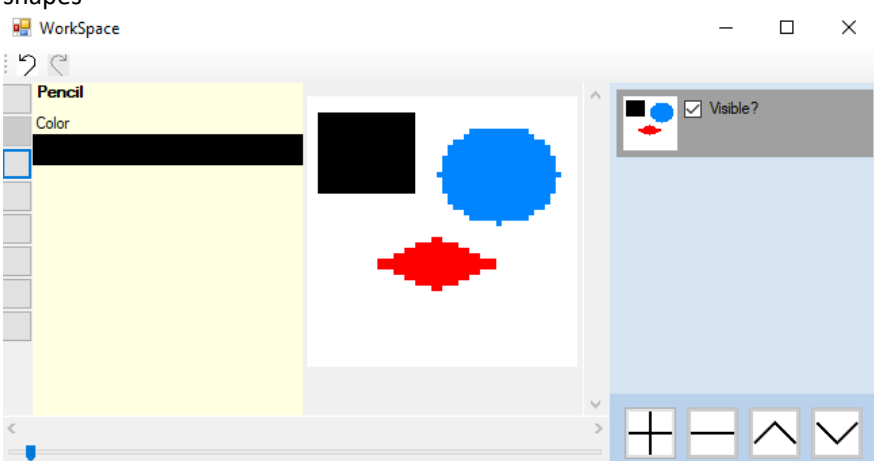
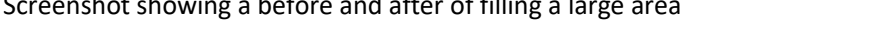
3.2.1 Stakeholder Feedback Drawings

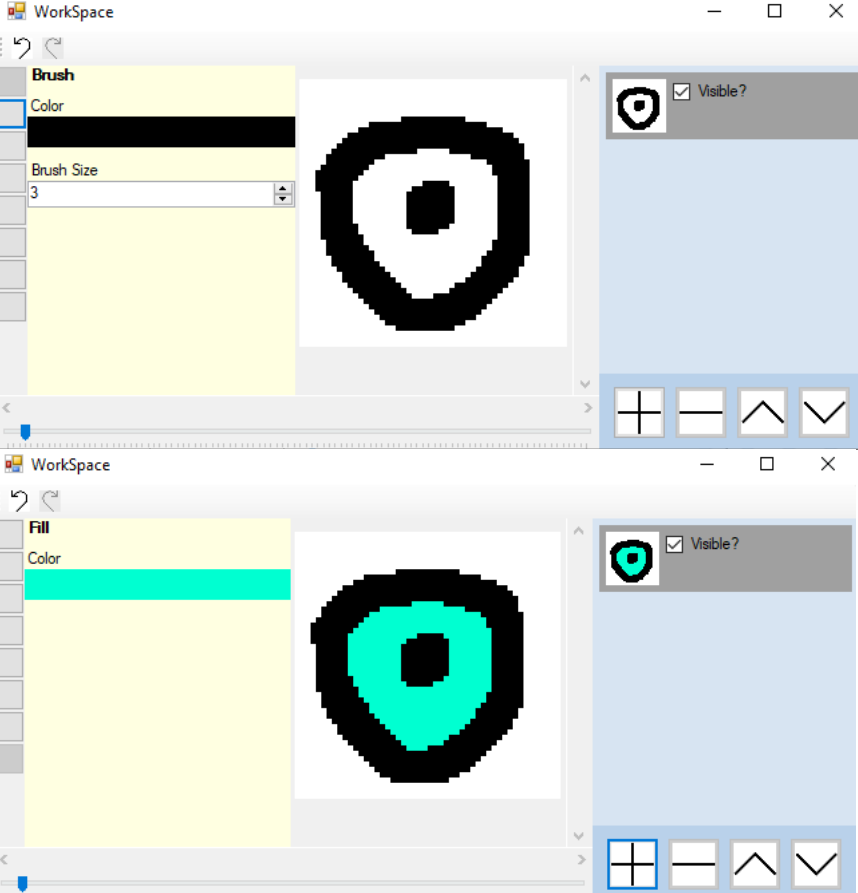

My stakeholders each drew images using SIMP, with the latest tools.

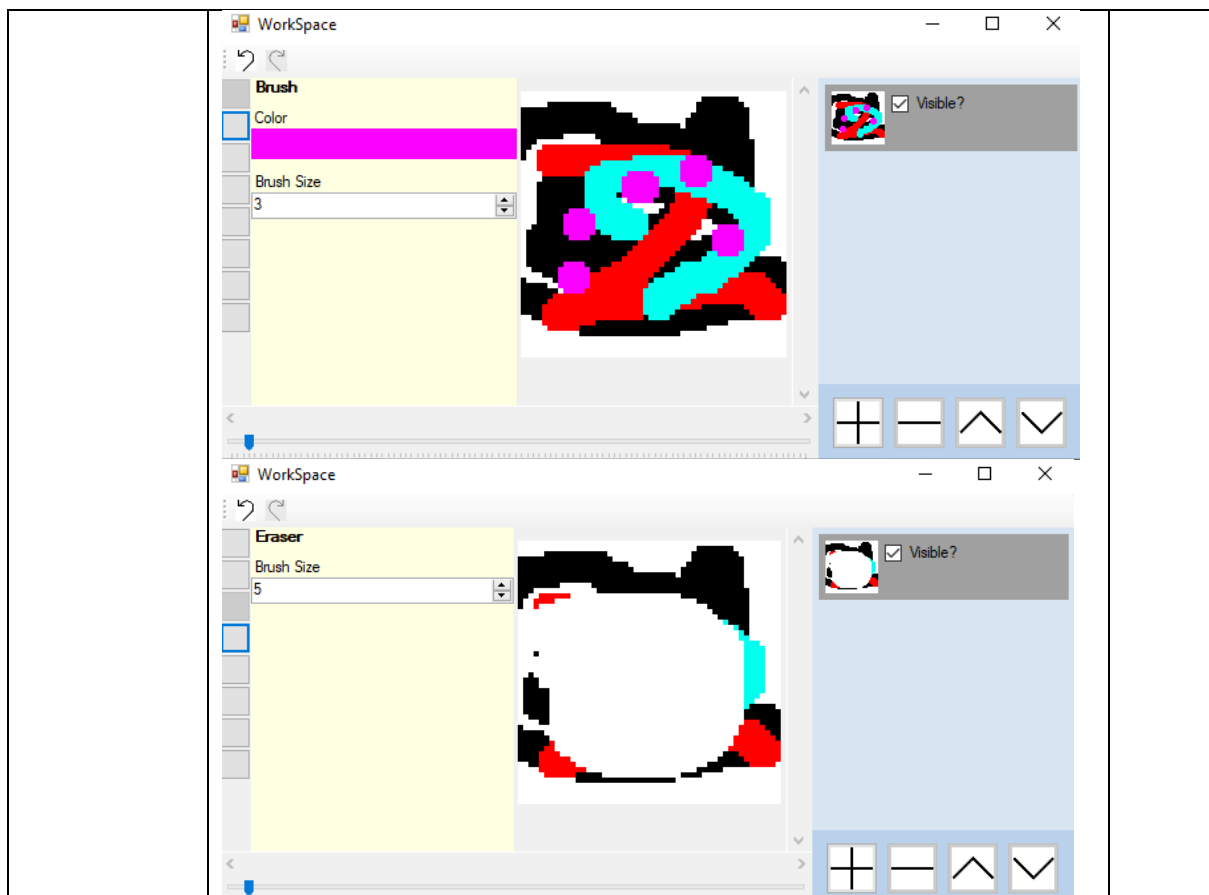
Dheshpreet



3.2.2 Success Criteria Evaluation

Feature	Proof	Code
Section A - Brushes		
Variable brush width	<p>Screenshot of strokes of the same brush showing different widths</p> 	A1
Hard brushes	<p>Screenshot showing the hard edge of the brush (colour to no colour)</p> 	A2
Shape creation tools	<p>Screenshot showing the shape toolbar and a small selection of drawn shapes</p> 	A3
Fill (bucket) tool	<p>Screenshot showing a before and after of filling a large area</p> 	A4

		
Single pixel pencil	<p>Screenshot showing a stroke of the single pixel brush</p> 	A5
Rubber	<p>Screenshot showing a densely packed picture being rubbed out</p>	A6



Section B – Other editing tools

RGB colour
picker

Screenshot showing a system for entering an RGB colour

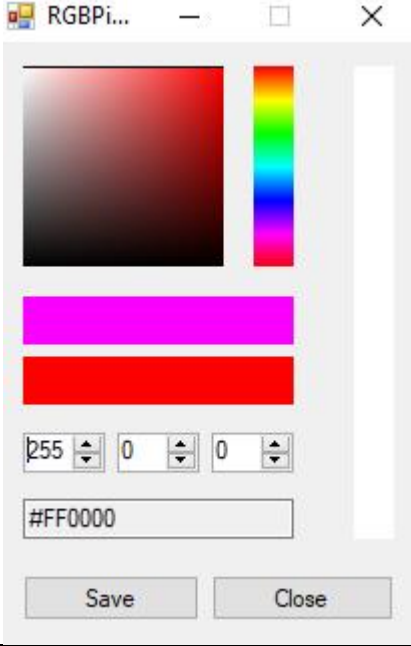
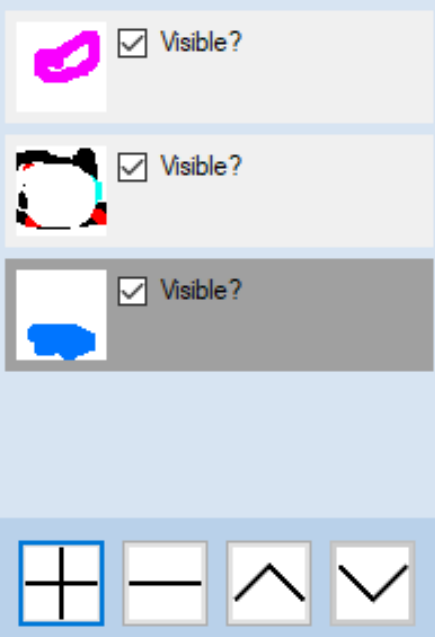
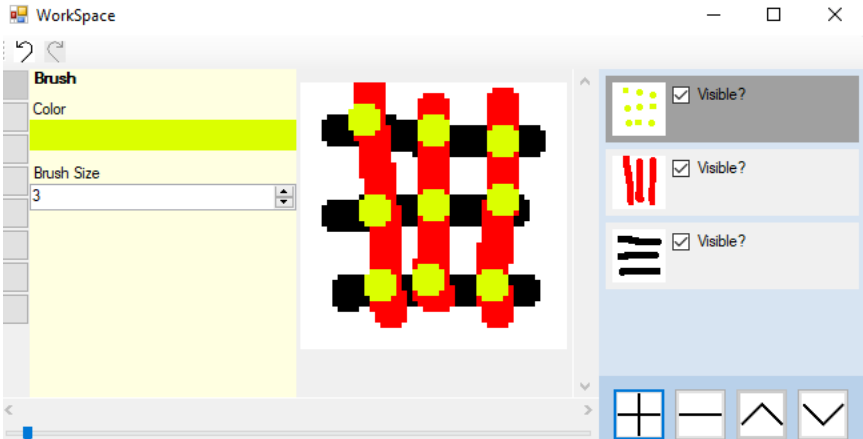


B3

RGB direct
input

Screenshot showing the user entering "FF0000" (or equivalent) and the programming outputting red

B4

		
Layer system	<p>Screenshot of layer navigator</p> 	B5
Transparent pixels	<p>Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required</p> 	B8

Section C – File System

Creating a new image



C1

So this makes the current success criteria:

Not completed

Completed

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

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