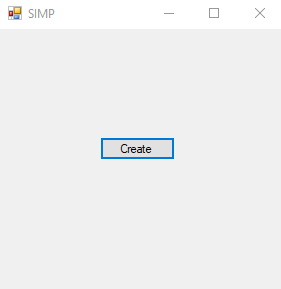
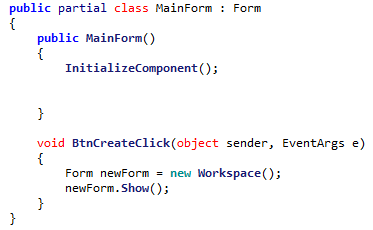
2.2 Developement

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

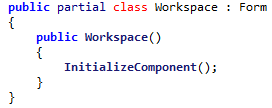
A basic design for the Main Form has been made:



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

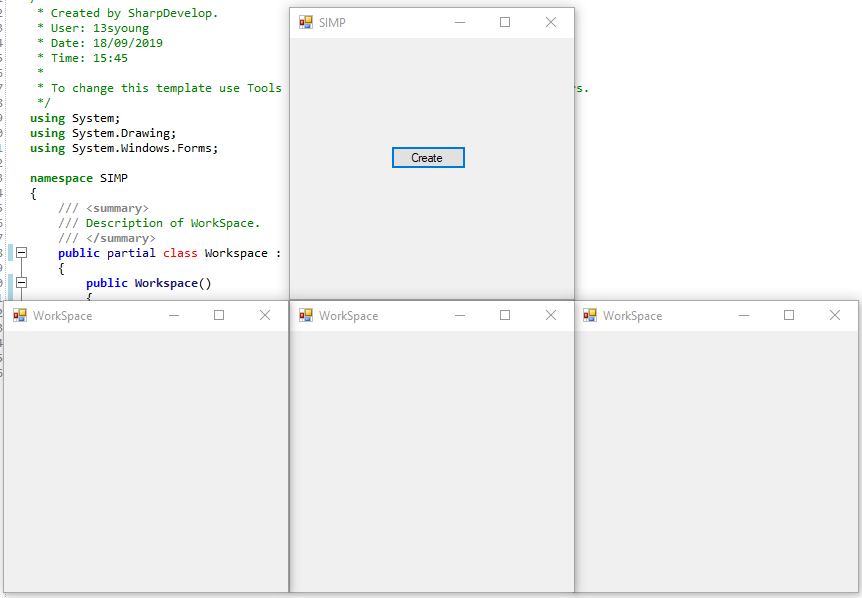


The Workspace itself only contains code for creating a form:



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

As demonstrated here, multiple Workspaces can be created:



# 19/09/2019 – Remaining Class Implementation

## Workspace implementation

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

<class>

Workspace

image

width

height

form

displayBox

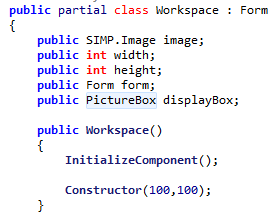
IsScrollBarVisible()

GetScrollBarMax()

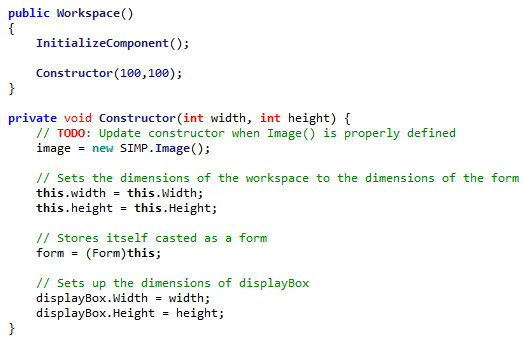
GetScrollBarValue()

SetScrollBarValue()

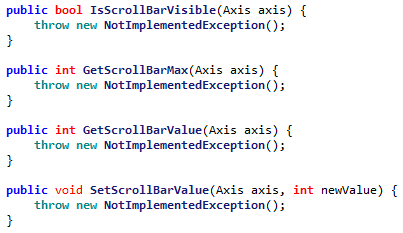
Properties:



The constructor has also been defined:



The necessary functions have also been defined:



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

## Axis Implementation

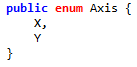
The Axis enum has been implemented in accordance to 2.1.4.2:

<enum>

Axis

X

Y



## Image Implementation

The image implementation has been completed in accordance to 2.1.4.1.

<class>

Image

colours[,]

width

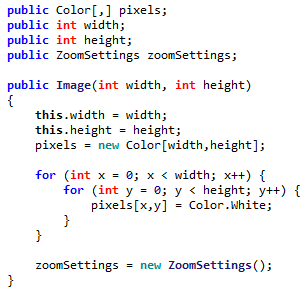
height

zoomSetting

GetPixel()

SetPixel()

GetDisplayImage()



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

class image {

constructor(\_width, \_height) {

width = \_width

height = \_height

pixels = new array of Color(width,height)

// *gives every pixel a default white colour*

foreach Color in Pixels {

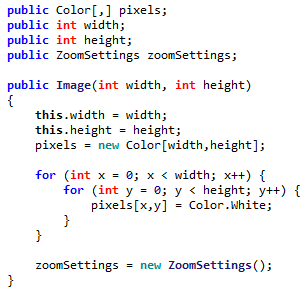
Color = White

}

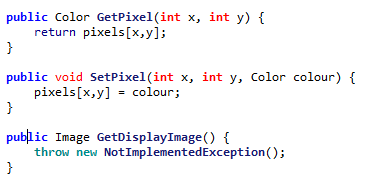
zoomSettings = new ZoomSetting()

}

}



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



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

## ZoomSettings implementation

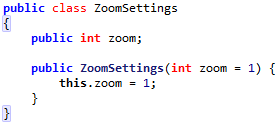
ZoomSettings has been implemented in accordance to 2.1.4.1:

<class>

ZoomSettings

centreLocation

zoomAmount



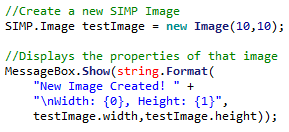
However centreLocation has not been implemented. This is since it will only be relevant when the zooming has been fully implemented.

In doing this, Algorithm 2.1 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:



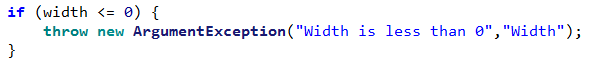
This is the part that will be changed during the testing phase

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| new Image(10,10) | 1 | An image of size 10fpx, 10fpx |  | The valid image is created |
| new Image(10,5) | 2 | A 10fpx, 5fpx image |  | The valid image is created |
| new Image(5,10) | 3 | A 5fpx, 10fpx image |  | The valid image is created |
| new Image(10,1) | 4 | A 10fpx, 1fpx image |  | The valid image is created |
| new Image(1,10) | 5 | A 1fpx, 10fpx image |  | The valid image is created |
| new Image(1,1) | 6 | A 1fpx, 1fpx image |  | The valid image is created |
| new Image(10,0) | 7 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(0,10) | 8 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(0,0) | 9 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(10000,10) | 10 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(10,10000) | 11 | The parameters are rejected and no image is created |  | The invalid image is not stopped |
| new Image(-1,10) | 12 | The parameters are rejected and no image is created | System.OverflowException: Arithmetic operation resulted in an overflow. | A potentially confusing error is thrown |
| new Image(10,-1) | 13 | The parameters are rejected and no image is created | System.OverflowException: Arithmetic operation resulted in an overflow. | A potentially confusing error is thrown |
| new Image(10) | 14 | The parameters are rejected and no image is created | 'SIMP.Image' does not contain a constructor that takes 1 arguments | The argument is rejected correctly |
| new Image(“10”,”10”) | 15 | The parameters are rejected and no image is created | Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int' | The argument is rejected correctly |

To fix these errors, some extra validation is needed in the Image constructor.

### Fixing Error #7 & #9 & #12

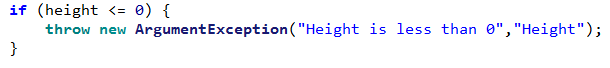
To fix these errors, a small check can be implemented:



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

### Fixing Error #8, #13

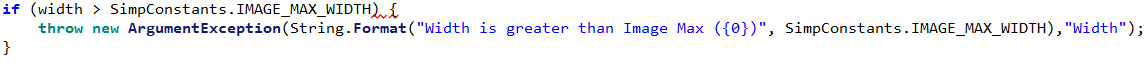
To fix these errors, a small check can be implemented:



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

### Fixing Error #10

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



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

### Fixing Error #11

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



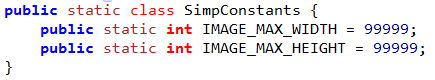
This throws a descriptive error about why the image was rejected.

## Unit 2.2 Unit Test II

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| new Image(10,10) | 1 | An image of size 10fpx, 10fpx |  | The valid image is created |
| new Image(10,5) | 2 | A 10fpx, 5fpx image |  | The valid image is created |
| new Image(5,10) | 3 | A 5fpx, 10fpx image |  | The valid image is created |
| new Image(10,1) | 4 | A 10fpx, 1fpx image |  | The valid image is created |
| new Image(1,10) | 5 | A 1fpx, 10fpx image |  | The valid image is created |
| new Image(1,1) | 6 | A 1fpx, 1fpx image |  | The valid image is created |
| new Image(10,0) | 7 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| new Image(0,10) | 8 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(0,0) | 9 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(10000,10) | 10 | The parameters are rejected and no image is created | System.ArgumentException: Width is greater than Image Max (9999) | A descriptive error is thrown |
| new Image(10,10000) | 11 | The parameters are rejected and no image is created | System.ArgumentException: Height is greater than Image Max (9999) | A descriptive error is thrown |
| new Image(-1,10) | 12 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| new Image(10,-1) | 13 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| new Image(10) | 14 | The parameters are rejected and no image is created | 'SIMP.Image' does not contain a constructor that takes 1 arguments | The argument is rejected correctly |
| new Image(“10”,”10”) | 15 | The parameters are rejected and no image is created | Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int' | The argument is rejected correctly |

## SimpConstants implementation

SimpConstants is a small class that contains static constants to be used throughout the program. It consists of two constants currently:

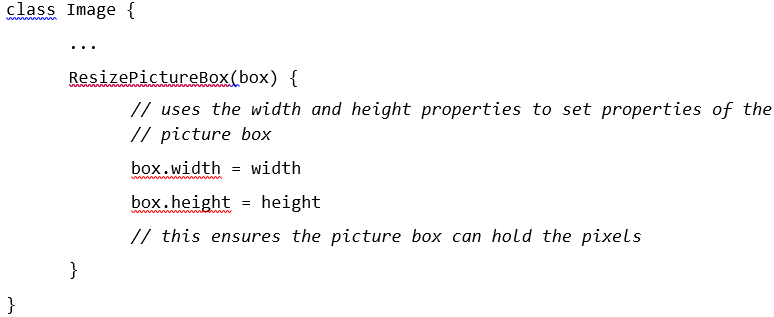
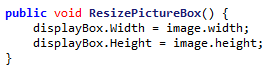


I **discussed with my stakeholders**, and they agreed that 99,999 pixels would be the largest that they’d need for an image.

# 21/09/2019 Picture Box setup

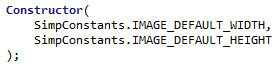
## Algorithm 2.3 – Resizing Picture Box

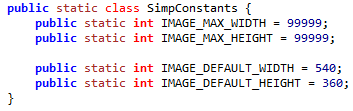
The algorithm for resizing the picture box has been implemented, though contains **changes** from the planned pseudocode:



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

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





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

## Unit 2.3 Unit Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Input Box(10,10) | 1 | An image of size 10fpx, 10fpx |  | The Picture Box is correctly resized. |
| Input Box(10,5) | 2 | A 10fpx, 5fpx image |  | The Picture Box is correctly resized. |
| Input Box(5,10) | 3 | A 5fpx, 10fpx image |  | The Picture Box is correctly resized. |
| Input Box(10,1) | 4 | A 10fpx, 1fpx image |  | The Picture Box is correctly resized.  (Though is very thin) |
| Input Box(1,10) | 5 | A 1fpx, 10fpx image |  | The Picture Box is correctly resized.  (Though is very thin) |
| Input Box(1,1) | 6 | A 1fpx, 1fpx image |  | The Picture Box is correctly resized.  (Though is very small) |
| Input Box(10,0) | 7 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| Input Box(0,10) | 8 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(0,0) | 9 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(10000,10) | 10 | The parameters are rejected and no image is created | System.ArgumentException: Width is greater than Image Max (9999) | A descriptive error is thrown |
| Input Box(10,10000) | 11 | The parameters are rejected and no image is created | System.ArgumentException: Height is greater than Image Max (9999) | A descriptive error is thrown |
| Input Box(-1,10) | 12 | The parameters are rejected and no image is created | System.ArgumentException: Width is 0 or less | A descriptive error is thrown |
| Input Box(10,-1) | 13 | The parameters are rejected and no image is created | System.ArgumentException: Height is 0 or less | A descriptive error is thrown |
| Input Box(10) | 14 | The parameters are rejected and no image is created | 'SIMP.Image' does not contain a constructor that takes 1 arguments | The argument is rejected correctly |
| Input Box(“10”,”10”) | 15 | The parameters are rejected and no image is created | Argument 1: cannot convert from 'string' to 'int' Argument 2: cannot convert from 'string' to 'int' | The argument is rejected correctly |

As this code is only used from the properties of the Image, this means that it reuses its code from the Image sanitation.

## Algorithm 2.4 – Displaying Image

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

GetDisplayImage() {

*// creates a C# ‘image’ object to store the output image*

Drawing.Image image = new Drawing.Image(width,height)

for x = 1 to width {

for y = 1 to height {

colour = pixels[x,y]

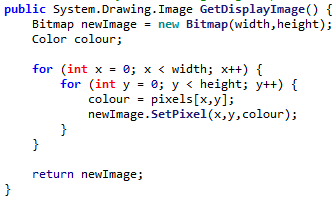
Image.SetPixel(x,y,colour)

}

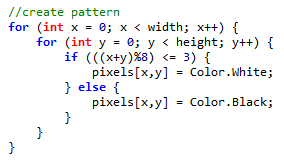
}

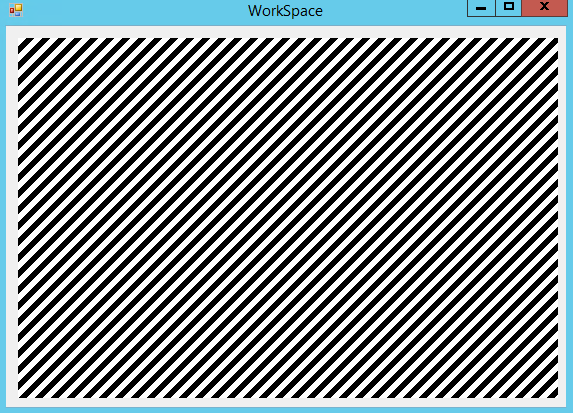
RETURN image

}



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





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

# 22/09/2019 Setting Pixels

## Algorithm 2.5 – Settings Pixels at runtime

In order to implement this, a small implementation for SetPixel has been added, following the original pseudocode:

SetPixel(x,y,colour) {

pixels[x,y] = colour

Display()

}



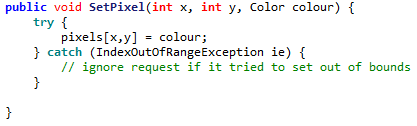
However there has been a change, as the call to Display() has been removed. This makes it so that the image will *not* be redrawn each time a pixel it set, and this helps to optimise large groups of pixel manipulation. This also means that it will be up to the calling function to re-display when necessary

## Unit 2.5 Unit Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| SetPixel(5,5,Black) | 1 | A pixel near the middle of the image is set to black |  | This proves a pixel can be placed on the image. |
| SetPixel(5,5,Green) | 2 | A pixel near the middle of the image is set to yellow |  | This proves multiple colours of pixel can be set |
| SetPixel(0,0,Black) | 3 | A pixel in the top-left corner is set to black |  | This proves the top-left corner is properly accessible |
| SetPixel(9,9,Black) | 4 | A pixel in the bottom-right corner is set to black |  | This proves the bottom-right corner is properly accessible. Both extremes have now been tested |
| SetPixel(-1,0,Black) | 5 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(0,-1,Black) | 6 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(10,0,Black) | 7 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |
| SetPixel(0,10,Black) | 8 | No pixel is set as it is out of bounds | System.IndexOutOfRangeException: Index was outside the bounds of the array. | There is not an in-bounds check for this |

### Fixing Error #5 & #6 & #7 & #8

To fix these errors, an extra check can be added onto the SetPixel function:



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

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

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

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

## Unit 2.5 Unit Test II

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SetPixel(5,5,Black) | 1 | A pixel near the middle of the image is set to black |  | This proves a pixel can be placed on the image. |
| SetPixel(5,5,Green) | 2 | A pixel near the middle of the image is set to yellow |  | This proves multiple colours of pixel can be set |
| SetPixel(0,0,Black) | 3 | A pixel in the top-left corner is set to black |  | This proves the top-left corner is properly accessible |
| SetPixel(9,9,Black) | 4 | A pixel in the bottom-right corner is set to black |  | This proves the bottom-right corner is properly accessible. Both extremes have now been tested |
| SetPixel(-1,0,Black) | 5 | No pixel is set as it is out of bounds |  | This proves a too small X is rejected |
| SetPixel(0,-1,Black) | 6 | No pixel is set as it is out of bounds |  | This proves a too large X is rejected |
| SetPixel(10,0,Black) | 7 | No pixel is set as it is out of bounds |  | This proves a too small Y is rejected |
| SetPixel(0,10,Black) | 8 | No pixel is set as it is out of bounds |  | This proves a too large Y is rejected |

# 23/09/2019 Resizing Picture Box

## Algorithm 2.6 – Resizing Picture Box

The code for resizing the Picture Box has been implemented, in accordance to planned pseudocode 2.6

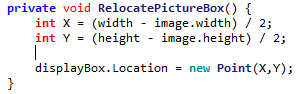
RelocatePictureBox {

Integer X = (width – image.width) / 2

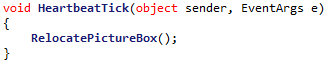
Integer Y = (height – image.height) / 2

displayBox.Location = new Location(X,Y)

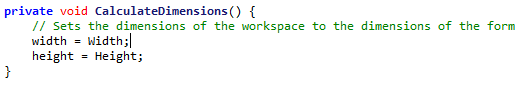
}



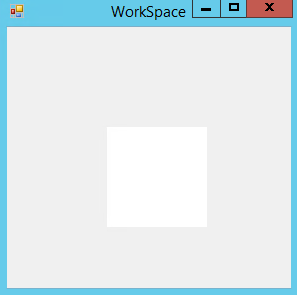
However, some additional code must be added to make the picture box constantly update. To do this a small timer module has been added to the code which calls the following procedure every millisecond:



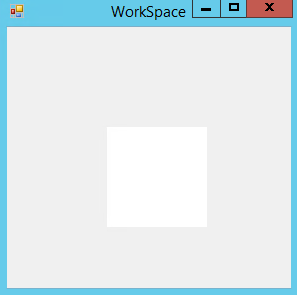
However this is still not enough, as ‘width’ and ‘height’ are currently constant and do not update when the form is resized. This can be solved by adding a new event called when the workspace is resized:



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



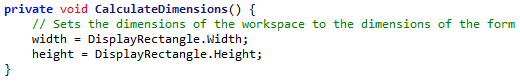
This is caused by the fact that calling the width of the form will return the **entire** width of the form, including its borders. For example:



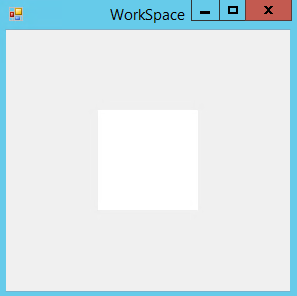
Desired Height

Height returned by ‘Height’

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



Resulting in:



# 24/09/2019 Minimum Form Size

## Algorithm 2.7 – Minimum Form Size

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

form.minimumWidth = image.width

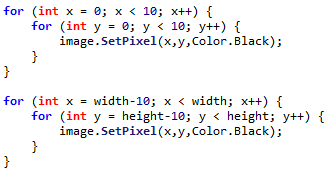
form.minimumHeight = image.height



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

## Unit Test 2.7

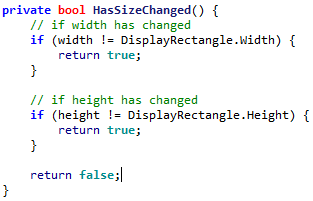
In order to test this, two boxes have been added to the bottom left and top right of the image:



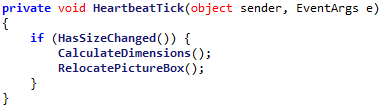
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is started at normal size | 1 | Image Displays in the middle of the form |  | The picture box is not correctly resized upon starting |
| Form is maximized | 2 | Image displays in the middle of the large form |  | The picture box is not resized upon maximizing |
| Form is minimized | 3 | No image is displayed (as form is currently invisible) | (No screenshot) | The program does not crash upon minimizing |
| Form is resized to smallest possible | 4 | The form cannot be made smaller than the image |  | The two squares in the corners are not displayed, so the minimum size is too small |
| Form is resized to minimum width maximum height | 5 | A very thin form displays the image |  | The pixels in the corners are still cut off |
| Form is resized to minimum height maximum width | 6 | A very short form displays the image |  | The pixels in the corners are not displayed |
| The form’s size is rapidly changed. | 7 | The image is very quickly moved around but remains centred |  | The picture box is only moved upon the end of resizing so the picture box aligns correctly eventually |

### Fixing Error #1 & #2

Errors #1 and #2 both stem from the Form not being resized when it should be. This is due to the ResizeEnd event that the resizing is currently linked to not triggering at all times. To fix this a function was added to check if the Form’s size has changed:

****

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



So the tests can now be repeated:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is started at normal size | 1 | Image Displays in the middle of the form |  | The picture box is resized but the picture box does not correctly display the entire image. |
| Form is maximized | 2 | Image displays in the middle of the large form |  | The picture box is correctly placed when the form is resized |

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

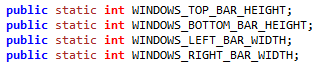
### Fixing Error #1 & #4 & #5 & #6

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

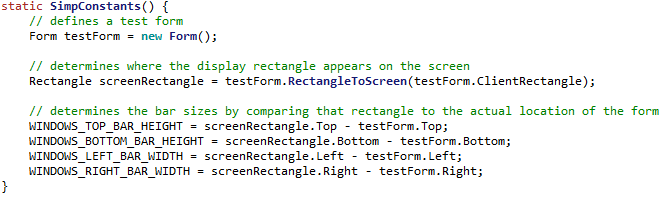


The error comes from the fact that MinimumSize does not take into account the borders around the Form, meaning elements like the top bar can cut the image off.

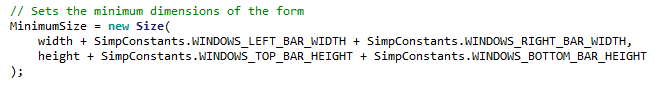
To resolve this, the size of the top bar (and other borders) needs to be determined so that they can be factored into the equation.



They can be calculated using the following code:



So now they can be included in the size calculations



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

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

# 25/09/2019 Point Redesign

Before completing any further algorithms, some upgrades need to be completed on the base classes, in accordance to 2.1.6

## FilePoint

A basic structure has been implemented for FilePoint, in accordance to 2.1.6.2

<class>

FilePoint

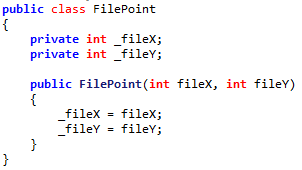
fileX

fileY

GetFilePoint

GetDisplayPoint

GetFileValue



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

## DisplayPoint

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

<class>

DisplayPoint

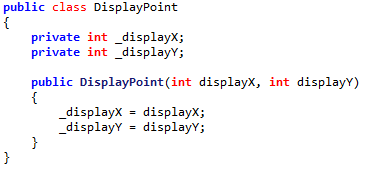
displayX

displayY

GetFilePoint

GetDisplayPoint

GetDisplayValue



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

## IPicturePoint

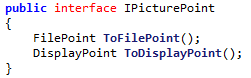
IPicturePoint has been implemented, in accordance to 2.1.6.1

<interface>

IPicturePoint

GetFilePoint

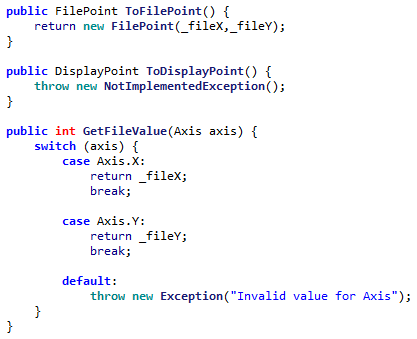
GetDisplayPoint



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

## FilePoint functions

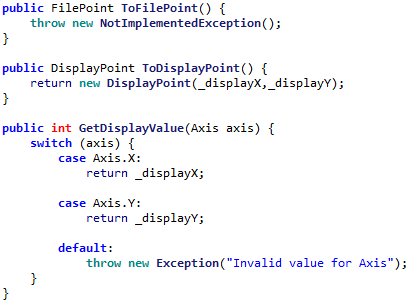
The remaining functions for FilePoint have been implemented:



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

## DisplayPoint functions

The remaining functions for DisplayPoint have been implemented:



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

# 26/09/2019 Rectangle Redesign

## IPictureRectangle

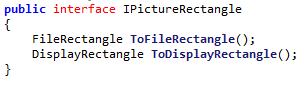
IPictureRectangle has been implemented, in accordance to 2.1.6.4

<interface>

IPictureRectangle

GetFileRectangle

GetDisplayRectangle



## FileRectangle

The FileRectangle class has been implemented in accordance to 2.1.6.5

<class>

FileRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetFileSizeInAxis



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

## DisplayRectangle

The DisplayRectangle class has been implemented in accordance to 2.1.6.6

<class>

DisplayRectangle

topLeftCorner

bottomRightCorner

\*GetFileRectangle

\*GetDisplayRectangle

GetTopLeftCorner

GetBottomRightCorner

GetDisplaySizeInAxis



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

## ZoomSettings upgrade

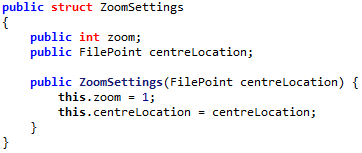
The ZoomSettings class has been upgraded in accordance to 2.1.6.7

<class>

ZoomSettings

centreLocation

zoomAmount



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

constructor(\_centreLocation) {

centreLocation = \_centreLocation

zoomAmount = 1

}

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

# 27/09/2019 Remaining Class Redesigning

## Image Redesign

The remaining property for Image has been implemented, the attachedWorkspace

<class>

Image

colours[,]

width

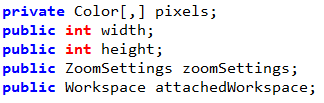
height

zoomSetting

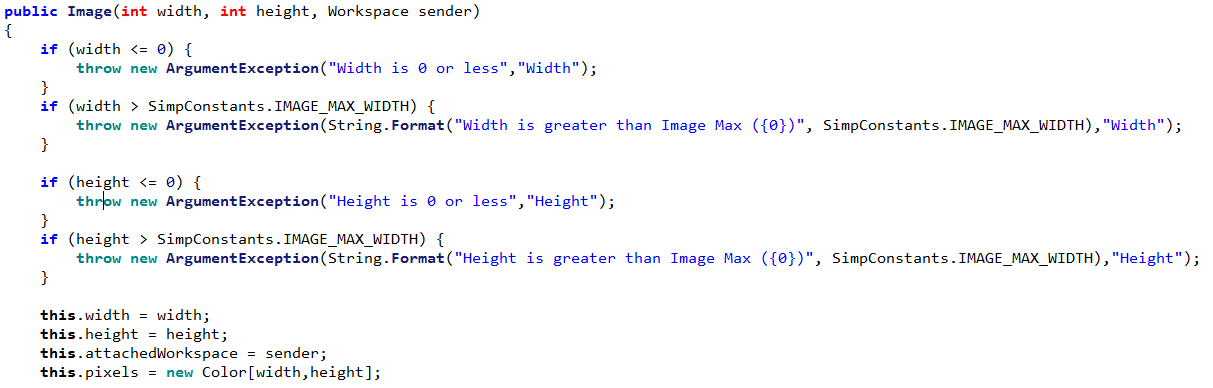
attachedWorkspace

GetPixel()

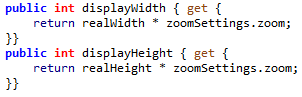
SetPixel()



So thus the constructor has been edited:



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



## Workspace Redesign

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

<class>

Workspace

displayPictureBox

attachedForm

displayWidth

displayHeight

LEFT\_PADDING

RIGHT\_PADDING

TOP\_PADDING

BOTTOM\_PADDING

GetPixel

SetPixel

GetDisplaySizeInAxis

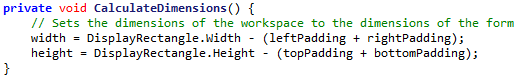
GetDisplayRectangle

Display

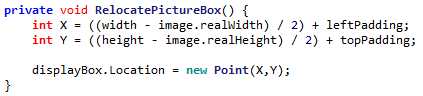
HandleResize

## Implementing Padding

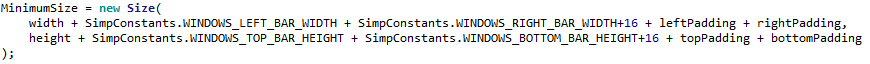
Padding has now been implemented into the relevant calculations, including ones for defining size of displayBox:



Code for determining locations of displayBox:



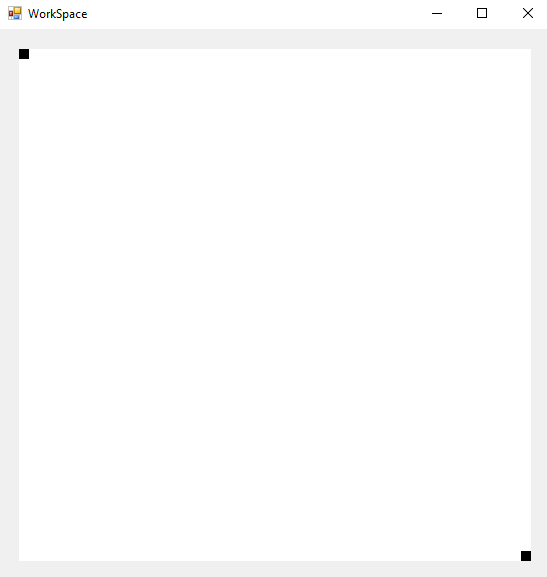
Code for determining minimum size:



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



The resulting program cannot be made smaller than this size:



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

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

|  |  |  |
| --- | --- | --- |
| **Property** | **Datatype** | **Justification** |
| LEFT\_PADDING | (**constant**) Integer | The amount of padding on the left hand side. Used for calculations involving where to place the picture box |
| RIGHT\_PADDING | (**constant**) Integer | The amount of padding on the right hand side |
| TOP\_PADDING | (**constant**) Integer | The amount of padding on the top side |
| BOTTOM\_PADDING | (**constant**) Integer | The amount of padding on the bottom side |

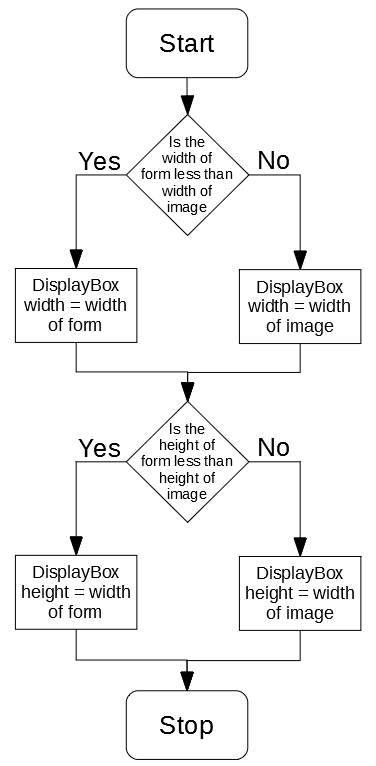
Compared to:



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

## Relocating displayBox change

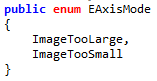
As the zoom is now being implemented, it becomes acceptable for the window to be smaller than the pictureBox. This means that the code for changing the displayBox’s size must be updated to look like:



# 28/09/2019 Implementing CheckImageSize

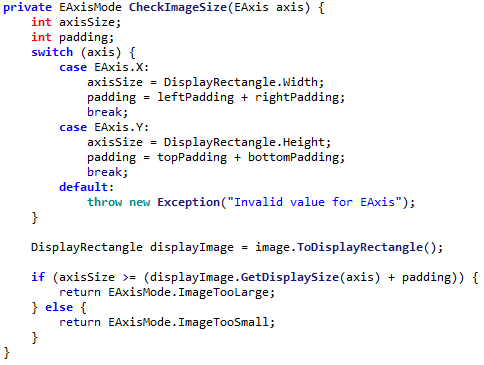
## EAxisMode

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



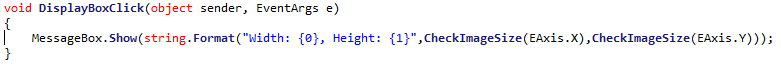
Using this enum, an algorithm for checking whether the image is too large in a specified axis can be implemented:

## Implementing CheckImageSize

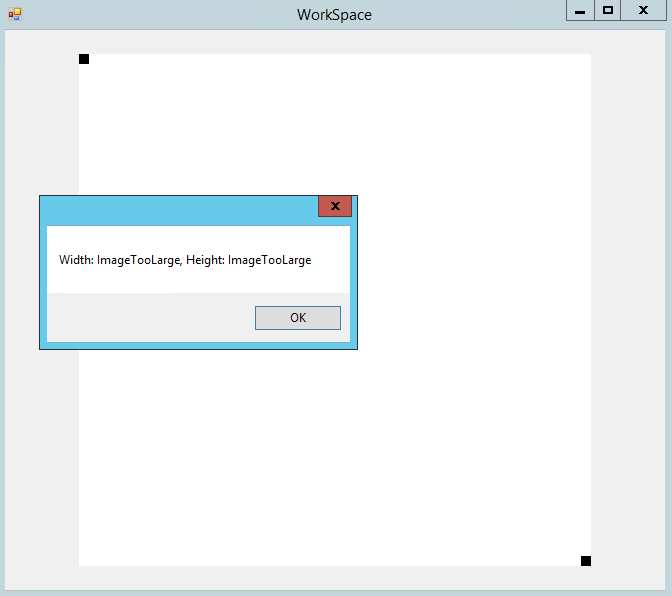


## Alpha Testing CheckImageSize #1

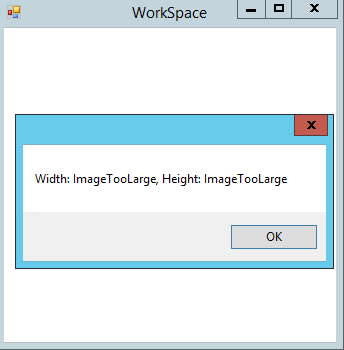
And can be **alpha tested** during development, using the following code:



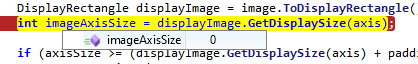
However, this testing reveals a problem, the code returns ImageTooLarge in situations where it should not:



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



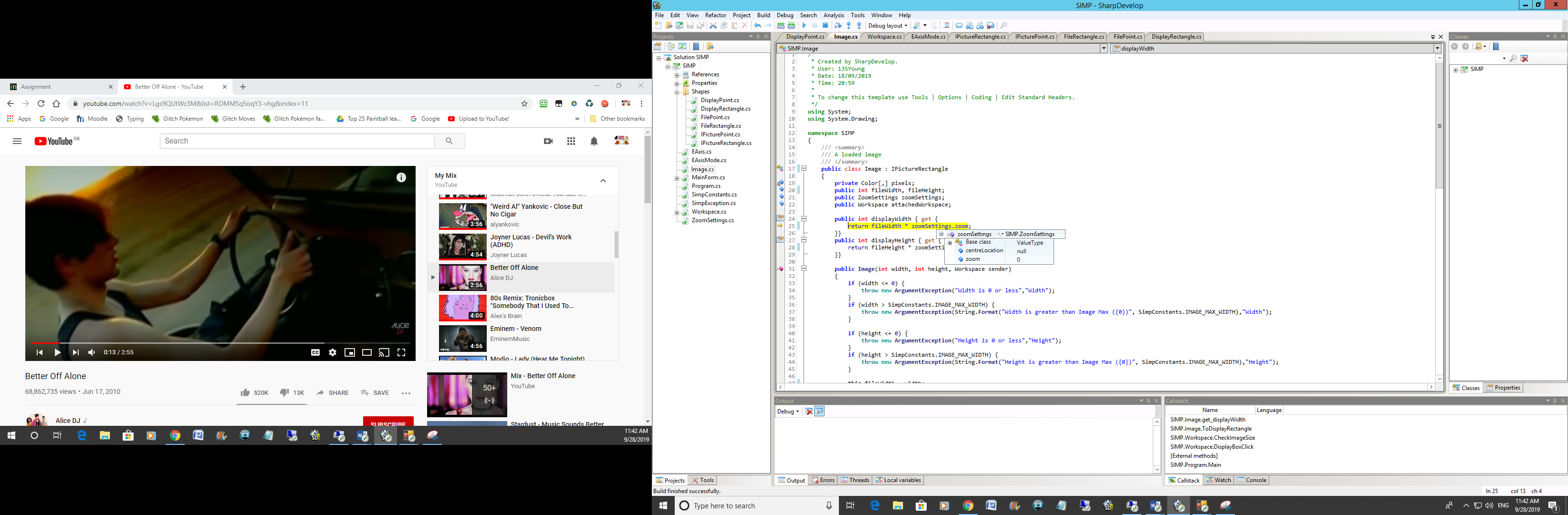
Using a **Break Point** to **debug** the code reveals that:



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

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

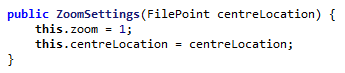
Finally the root of the issue was found, the zoom level was erraneously set to 0:



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

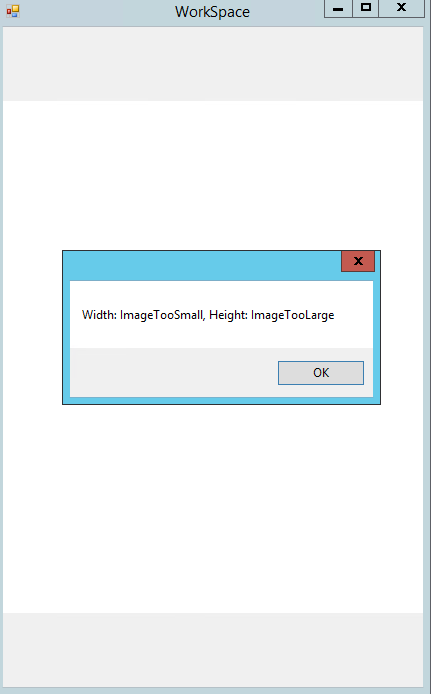
Which causes the class to be populated with null values, which in the case of an integer is 0.

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

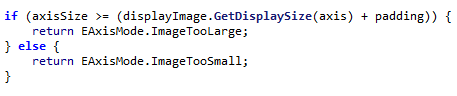


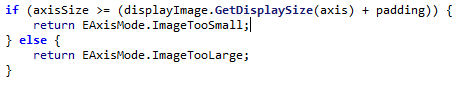
## Alpha Testing CheckImageSize #2

From testing CheckImageSize again, there is a second problem:



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



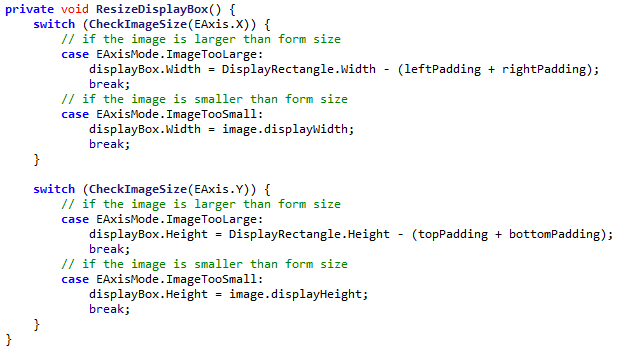


Thus, by **iterative development and bug fixing, the function has been created**.

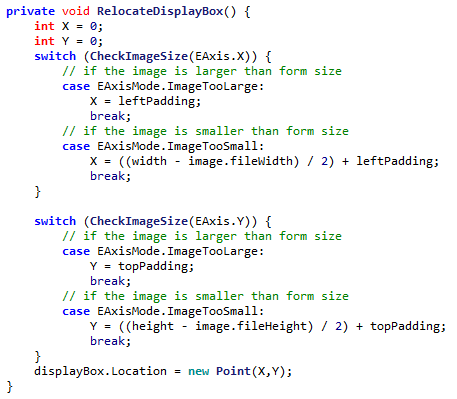
# 29/09/2019 Upgrading displayBox functionality

## Implementing proper resizing and relocating for displayBox

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

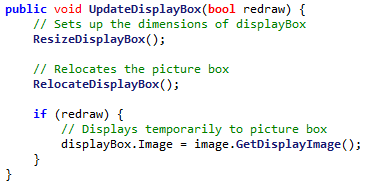


The switch cases take the role of the Diamond shape in the flowchart.

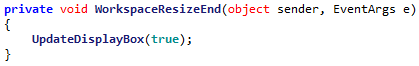
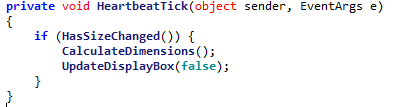


## Implementing new Image Requests

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



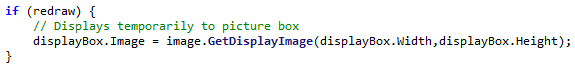
So this means that a new image will only be requested when the function is passed ‘True’. This does **not** happen on a normal resize, but does happen when the resize ends:



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



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



# 1/10/2019 Conversions between pixel types

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

FilePointToDisplayPoint(filePoint) {

displacementX = filePoint.X – centreFilePoint.X

displacementY = filePoint.Y – centreFilePoint.Y

displacementX = displacementX \* zoom

displacementY = displacementY \* zoom

newX = centreDisplayPoint.X + displacementX

newY = centreDisplayPoint.Y + displacementY

return new Point(newX, newY)

}

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

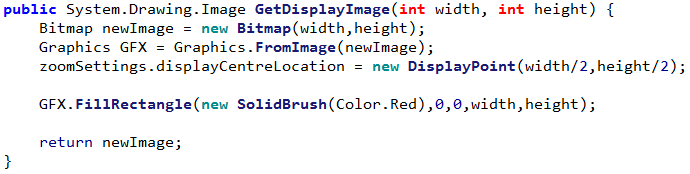
To do this, a new private parameter was added to the image, which contains a reference to the zoom settings of its image:



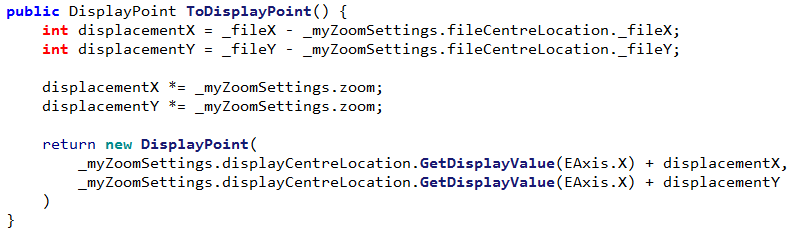
However zoomSettings does not contain any knowledge of where the display centre location is. This means that a parameter for this must be added to ZoomSettings:



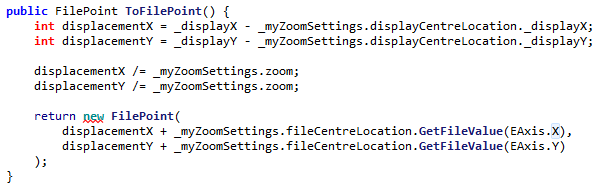
This is updated whenever a new image is requested (whenever the current displayCentreLocation might change).



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



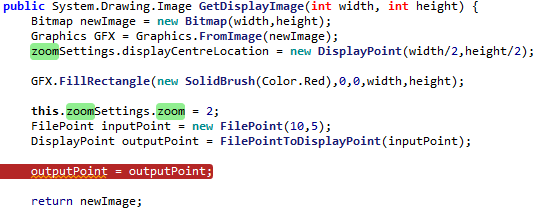
The code for converting from DIsplayPoint to FilePoint can also be implemented, in accordance to 2.8B.



# 2/10/2019 Testing Point conversions

## File Point to Display Point testing

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

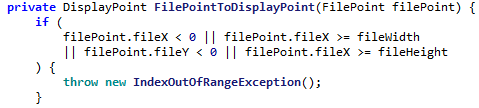


The data from the break point will be inspected to find the properties of the returned point.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| DisplayPoint(10,5) | 1 | (5,5) |  | The location of the centre of the image is returned |
| DisplayPoint(8,3) | 2 | (1,1) |  | The top-left is returned |
| DisplayPoint(12,7) | 3 | (9,9) |  | The bottom-right is returned |
| DisplayPoint(7,2) | 4 | (-1,-1) |  | A location off the edge of the image is returned |
| DisplayPoint(13,8) | 5 | (11,11) |  | A location off the edge of the image is returned |
| DisplayPoint(0,0) | 6 | (-15,-5) |  | The edge of the image is returned |
| DisplayPoint(19,9) | 7 | (23,13) |  | The edge of the image is returned |
| DisplayPoint(-1,-1) | 8 | Throws out of bounds error |  | No error is thrown |
| DisplayPoint(20,10) | 9 | Throws out of bounds error |  | No error is thrown |

### Fixing Error #8 & #9

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



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| DisplayPoint(-1,-1) | 8 | Throws out of bounds error | System.IndexOutOfRangeException: Index was outside the bounds of the array. | No error is thrown |
| DisplayPoint(20,10) | 9 | Throws out of bounds error | System.IndexOutOfRangeException: Index was outside the bounds of the array. | No error is thrown |

## Display Point to File Point testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| FilePoint(5,5) | 1 | (10,5) |  | The centre location is returned |
| FilePoint(1,1) | 2 | (8,3) |  | The correct point is returned |
| FilePoint(12,7) | 3 | (9,9) |  | The correct point is returned |
| FilePoint(-1,-1) | 4 | (7,2) |  | The correct point is returned |
| FilePoint(11,11) | 5 | (13,8) |  | The correct point is returned |
| FilePoint(-15,-5) | 6 | (0,0) |  | The correct point is returned |
| FilePoint(23,13) | 7 | (19,9) |  | The correct point is returned |
| FilePoint(6,6) | 8 | (10,5) |  | The point is rounded up |
| FilePoint(4,4) | 9 | (9,4) |  | The point is not correctly rounded down |
| FilePoint(-17,-7) | 10 | This point is rejected | System.IndexOutOfRangeException: Index was outside the bounds of the array. | The point is correctly rejected |
| FilePoint(25,15) | 11 | This point is rejected | System.IndexOutOfRangeException: Index was outside the bounds of the array. | The point is correctly rejected |

### Fixing Issue #9

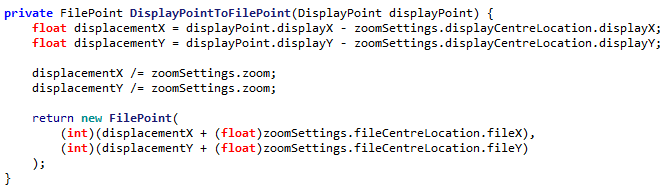
The issue in error #9 occurs because, in this situation, the displacement has a negative value. When this displacement is divided and then rounded, it is rounded **up**, as this is how negative numbers are implemented.





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

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



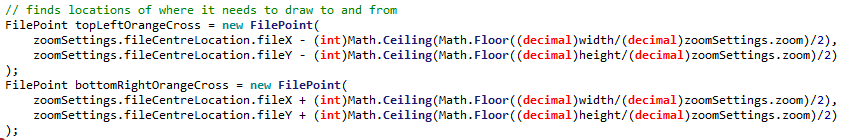
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| FilePoint(4,4) | 9 | (9,4) |  | The point is now correctly rounded down |

# 3/10/2019 Displaying an Image

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

## Algorithm 2.8B – Determining Border Locations

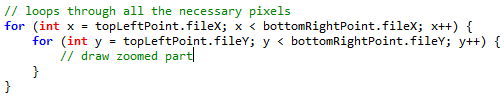
The algorithm for determining the border location has been implemented, in accordance to 2.8B



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

## Algorithm 2.8C – Finding Green Pixels

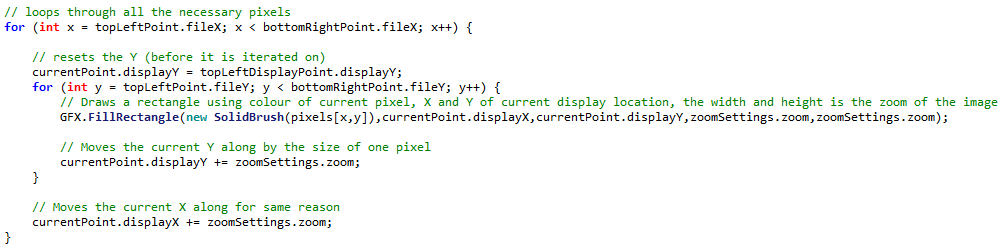
The algorithm for determining pixels to draw has been implemented, in accordance to 2.8C



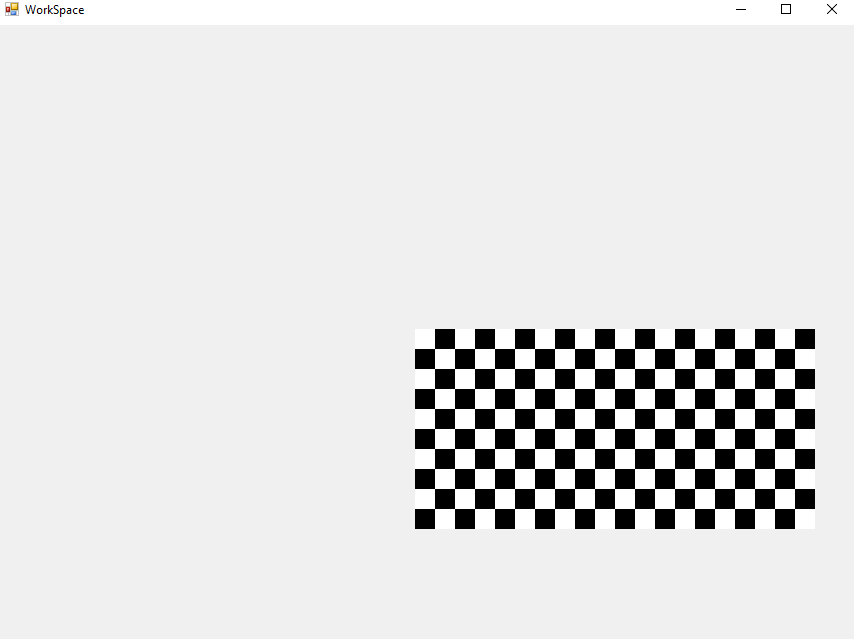
Where the comment will be replaced with the code necessary to draw the zoomed part

## Algorithm 2.9 – Draw Green Pixels

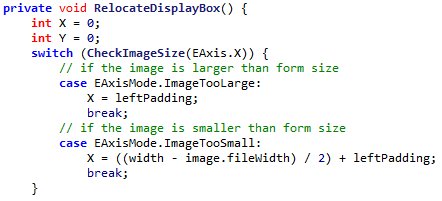
The above code has been upgraded to include drawing functionality:



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

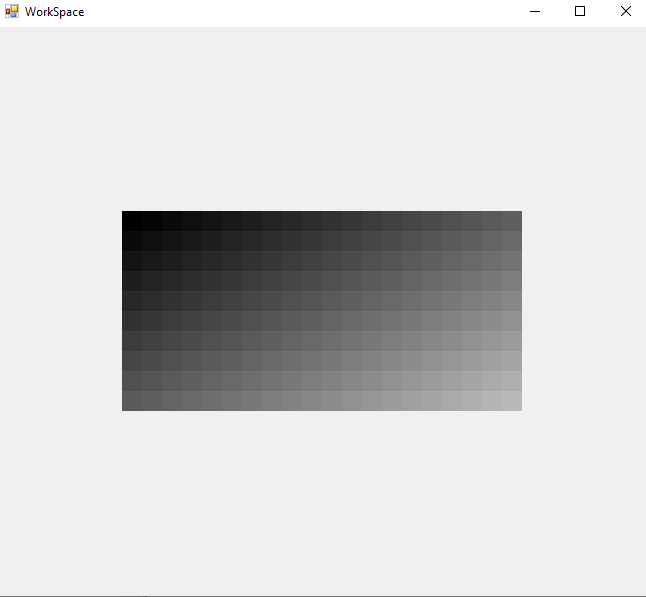


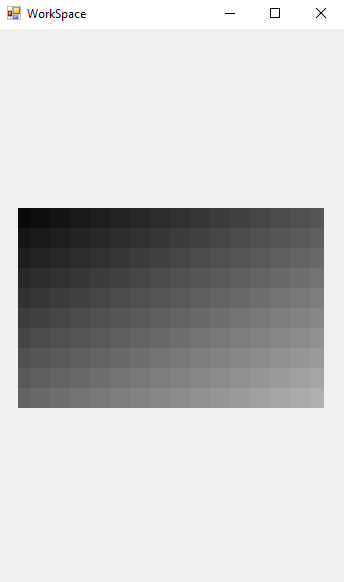
The cause of this was simple, fileWidth was erroneously used instead of displayWidth:



Changing this to displayWidth resolves the error.

**SIMP can now display images:**

****

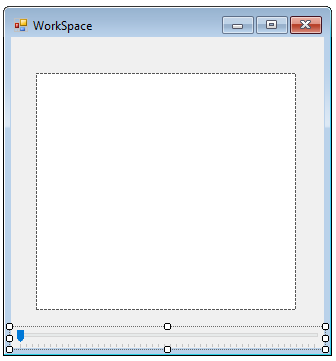
****

****

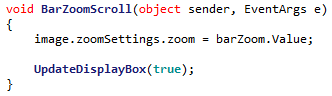
*Some example of SIMP displaying images, even when the display form is much smaller than the image*

## Adding Zoom Bar to GUI

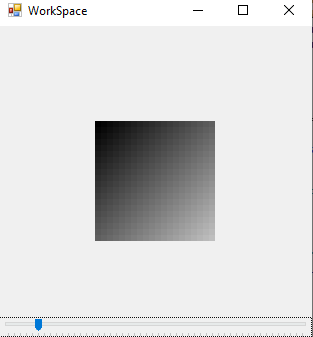
The zoom bar can be added quite simply, filling the border area at the bottom.

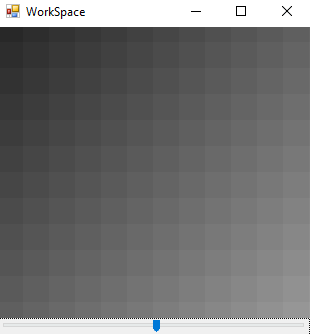


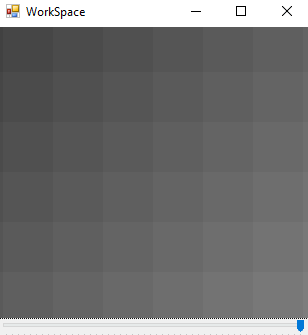
Its value can also be interpreted quite simply. Its minimum and maximum has been set to 1 and 50 respectively. When its value is changed, the image’s zoom is updated and the box is redrawn.



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

 Small amount of zoom

 Large amount of zoom

 Maximum zoom

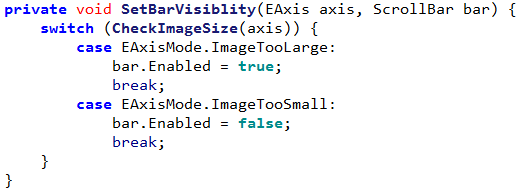
# 04/10/2019 Implementing Bars

## Determining whether Bars are visible

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

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

The completed check for this looks like:



## Algorithm 2.11A Determining Crosses in Axis

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

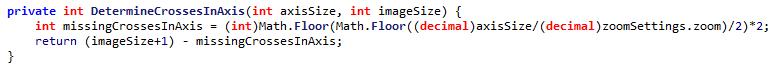
DetermineCrossesInAxis(Axis axis) {

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

MaxCrossesInAxis = image.getSizeInAxis(axis)

return MaxCrossesInAxis – MissingCrossesInAxis

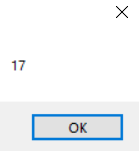
}



When testing the code with the following inputs:



The output is:



Which is what was expected in design.

## Algorithm 2.11B Setting Bar Size

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

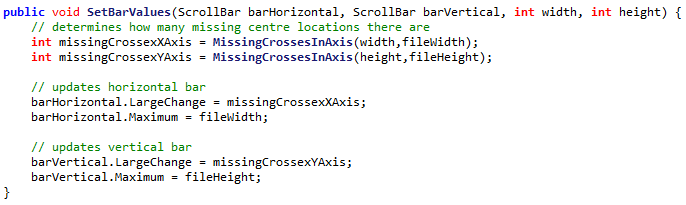
UpgradedSetupBarSize(progressBar, Axis) {

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

progressBar.barSize = MissingCrossesInAxis

progressBar.max = image.getFileWidth(Axis)

}



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

## Algorithm 2.11C Determining Centre Location from Bar Value

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

GetCentreLocationFromBar(progressBar bar, Axis axis) {

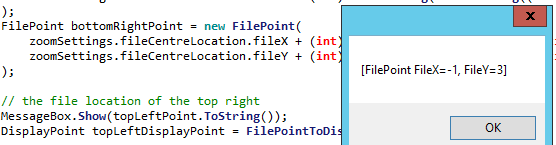
firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)

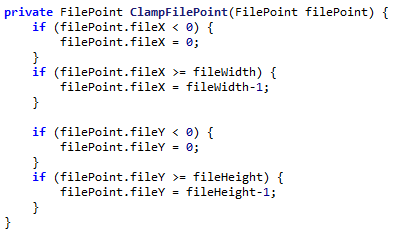
RETURN bar.value + firstCrossPosition

}

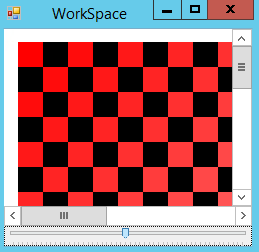
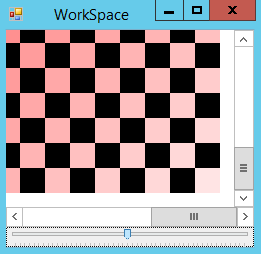
**Insert screenshot here since my laptop is too small to view it properly**

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



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

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

# 05/10/2019 Remaining Bar code

## Algorithm 2.11D Determining Bar Value from Centre Location

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

SetBarFromCentreLocation(progressBar bar, Axis axis) {

firstCrossPosition = Floor(Floor(form.getSizeInAxis(axis)/zoom)/2)

bar.value = centreLocation.getSizeInAxis(axis) - firstCrossPosition

}

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

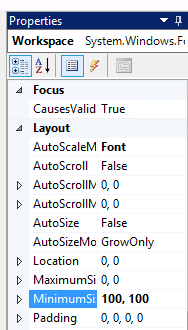
## Unit Test 2.11 #1

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is resized to half the size of image | 1 | Bar is half size of bounds and in its centre position |  | The bars are displaying the correct information |
| Form is resized to its smallest position | 2 | Bar is small but still central. | System.ArgumentException: Parameter is not valid. | An error is thrown as the image becomes too small to display. |
| Form is resized back to half size of image | 3 | Bar is half size of bounds and in its centre position | Cannot be tested as relies on success of previous test |  |
| Horizontal Scroll bar is moved to far left | 4 | The far left of the image is displayed, but no more |  | The far left of the image is displayed |
| Horizontal Scroll bar is moved to far right | 5 | The far right of the image is displayed, but no more |  | The far right of the image is displayed |
| Vertical Scroll bar is moved to highest | 6 | The highest of the image is displayed, but no more |  | The top of the image is displayed |
| Vertical Scroll bar is moved to lowest | 7 | The lowest of the image is displayed, but no more |  | The bottom of the image is displayed |
| Horizontal scroll bar is moved far left, then form size increased | 8 | Centre locations is moved to the left when needed | System.ArgumentOutOfRangeException: Value of '-1' is not valid for 'Value'. | The bar value cannot be properly determined as centre location becomes invalid |
| Form is resized to smallest, then maximized | 9 | The bars disappear and the view is normal | System.ArgumentException: Parameter is not valid. | An error is thrown as the image becomes too small to display. |

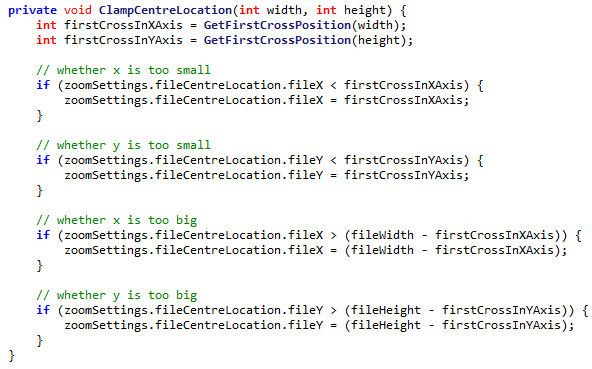
### Fixing Error #2 & #9

In order to fix the error with the form getting too small, a minimum size can be enforced via a property:



### Fixing Error #8

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

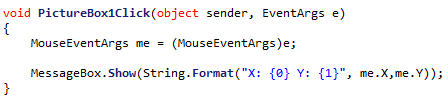


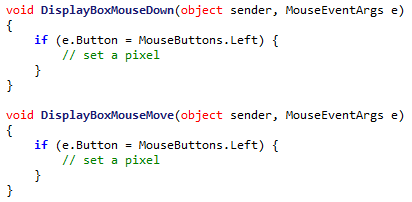
## Unit Test 2.11 #2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | ID | Expected Result | Actual Result | Comment |
| Form is resized to half the size of image | 1 | Bar is half size of bounds and in its centre position |  | The bars are displaying the correct information |
| Form is resized to its smallest position | 2 | Bar is small but still central. |  | The minimum form size prevents form from getting too small |
| Form is resized back to half size of image | 3 | Bar is half size of bounds and in its centre position |  |  |
| Horizontal Scroll bar is moved to far left | 4 | The far left of the image is displayed, but no more |  | The far left of the image is displayed |
| Horizontal Scroll bar is moved to far right | 5 | The far right of the image is displayed, but no more |  | The far right of the image is displayed |
| Vertical Scroll bar is moved to highest | 6 | The highest of the image is displayed, but no more |  | The top of the image is displayed |
| Vertical Scroll bar is moved to lowest | 7 | The lowest of the image is displayed, but no more |  | The bottom of the image is displayed |
| Horizontal scroll bar is moved far left, then form size increased | 8 | Centre locations is moved to the left when needed |  | The centre location of the image now correctly updates to the centre of the image, resolving the invalid bar value |
| Form is resized to smallest, then maximized | 9 | The bars disappear and the view is normal |  | The system copes with this sudden change of size. |

## Algorithm 2.13 Detecting Mouse Clicks

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

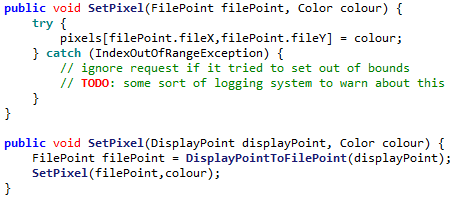




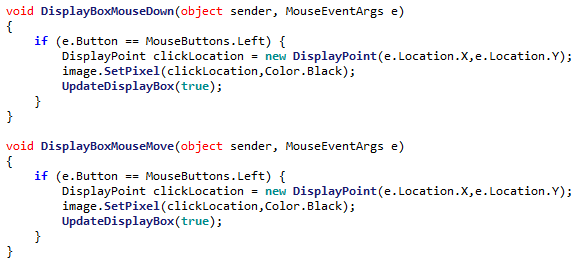
However this implementation uses two events rather than one, for when the mouse is held down and moved.

## Algorithm 2.14 & Algorithm 2.15

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



And then adding a call from WorkSpace:



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

2.3 Testing

# 2.3.1 Full Testing

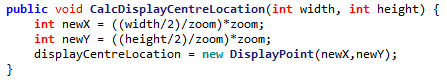
## Initial Testing Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | ID | | Expected Result | Actual Result | Comment |
| new Image(10,10) | | 1 | An image of size 10fpx, 10fpx |  | A 10px by 10px image is successfully created. |
| new Image(10,5) | | 2 | A 10fpx, 5fpx image |  | The slightly thinner image is successfully created |
| new Image(5,10) | | 3 | A 5fpx, 10fpx image |  | A tall image is successfully created. However there are problems drawing on the odd edge of this sort of image, as only half of the edge is displayed. |
| new Image(10,1) | | 4 | A 10fpx, 1fpx image |  | An image is created, but is much too thin (only displaying half of a pixel). |
| new Image(1,10) | | 5 | A 1fpx, 10fpx image |  | An image is created, but is much too thin as only a half pixel is displayed |
| new Image(1,1) | | 6 | A 1fpx, 1fpx image |  | A very small picture is created, but it is not centrally aligned in the image. |
| new Image(10,0) | | 7 | The parameters are rejected and no image is created |  | A height of 0 is not accepted, and autocorrects to 1 |
| new Image(0,10) | | 8 | The parameters are rejected and no image is created |  | A width of 0 is not accepted, and autocorrects to 1 |
| new Image(0,0) | | 9 | The parameters are rejected and no image is created |  | Both boxes are checked independent of each other so this poses no new issue |
| new Image(100000,10) | | 10 | The parameters are rejected and no image is created |  | The width is autocorrected back down to the allowed maximum |
| new Image(10,100000) | | 11 | The parameters are rejected and no image is created |  | The height is autocorrected back down to the allowed maximum |
| new Image(-1,10) | | 12 | The parameters are rejected and no image is created |  | The invalid width is corrected |
| new Image(10,-1) | | 13 | The parameters are rejected and no image is created |  | The invalid height is corrected |
| new Image(10) | | 14 | The parameters are rejected and no image is created |  | In cases where a parameter is left blank the last valid value is used instead (in this case 50) |
| new Image(“10”,”10”) | | 15 | The parameters are rejected and no image is created |  | It is impossible to create an image from text |
| SetPixel(5,5,Black) | | 16 | A pixel near the middle of the image is set to black |  |  |
| SetPixel(5,5,Green) | | 17 | A pixel near the middle of the image is set to yellow | This is currently impossible, however previous tests indicate that colour displaying is possible. |  |
| SetPixel(0,0,Black) | | 18 | A pixel in the top-left corner is set to black |  | A pixel at the top left is set |
| SetPixel(9,9,Black) | | 19 | A pixel in the bottom-right corner is set to black |  |  |
| SetPixel(-1,0,Black) | | 20 | No pixel is set as it is out of bounds |  | There is no change as attempts to set pixels out of bounds are ignored |
| SetPixel(0,-1,Black) | | 21 | No pixel is set as it is out of bounds |  | There is no change as attempts to set pixels out of bounds are ignored |
| SetPixel(10,0,Black) | | 22 | No pixel is set as it is out of bounds |  | There is no change as attempts to set pixels out of bounds are ignored |
| SetPixel(0,10,Black) | | 23 | No pixel is set as it is out of bounds |  | There is no change as attempts to set pixels out of bounds are ignored |
| Form is started at normal size | | 24 | Image Displays in the middle of the form |  |  |
| Form is maximized | | 25 | Image displays in the middle of the large form |  | The image (while small) is correctly displayed in the middle of the large form |
| Form is minimized | | 26 | No image is displayed (as form is currently invisible) | No image | There is no problem when the program is minimized |
| Form is resized to smallest possible | | 27 | The form cannot be made smaller than the image |  | A small amount of the image is shown, however the minimize button is displayed over the icon |
| Form is resized to minimum width maximum height | | 28 | A very thin form displays the image |  | A very wide form correctly displays the image in its centre |
| Form is resized to minimum height maximum width | | 29 | A very short form displays the image |  | A very tall form displays the image, but the top control bar is still displayed incorrectly |
| The form’s size is rapidly changed. | | 30 | The image is very quickly moved around but remains centred | No image, but program remains stable | The program can be very quickly manipulated but still appear smooth |
| Form is resized to half the size of image | | 31 | Bar is half size of bounds and in its centre position |  | The bar is in its correct position |
| Form is resized to its smallest position | | 32 | Bar is small but still central. |  | The bar still remains close to the centre of the image |
| Form is resized back to half size of image | | 33 | Bar is half size of bounds and in its centre position |  |  |
| Horizontal Scroll bar is moved to far left | | 34 | The far left of the image is displayed, but no more |  | The far left is displayed correctly |
| Horizontal Scroll bar is moved to far right | | 35 | The far right of the image is displayed, but no more |  | The far right is displayed correctly |
| Vertical Scroll bar is moved to highest | | 36 | The highest of the image is displayed, but no more |  | The top right is now correctly displayed |
| Vertical Scroll bar is moved to lowest | | 37 | The lowest of the image is displayed, but no more |  | The bottom right is now correctly displayed |
| Horizontal scroll bar is moved far right, then form size increased | | 38 | Centre locations is moved to the left when needed |  | The centre location is now correctly moved when the size increases |
| Form is resized to smallest, then maximized | | 39 | The bars disappear and the view is normal |  | The program can handle such a sudden change in size |

## Error Handling

### Fixing Error #3, #4, #5 and #6

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


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


So, performing the tests again:

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

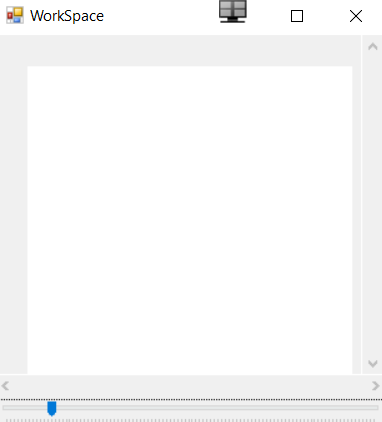
This now means that SIMP has passed **Alpha Testing**. It will now be handed to the Stakeholders for their initial thoughts and **Beta Testing**.

# 2.3.2 Client Testing & Feedback

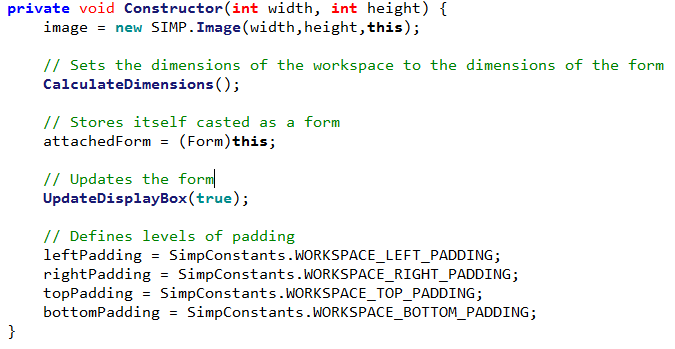
## Alex G Client Feedback

### Alignment at startup issue

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

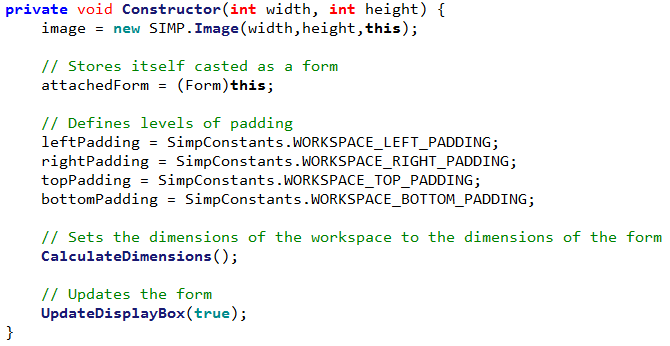


The error with the code happens because of the way the constructor is ordered:

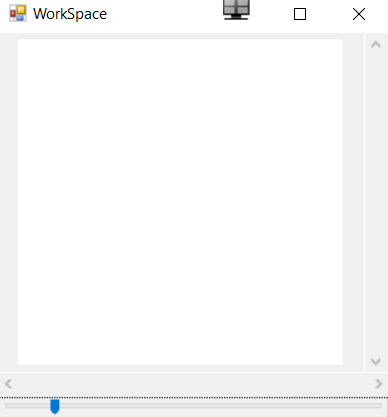


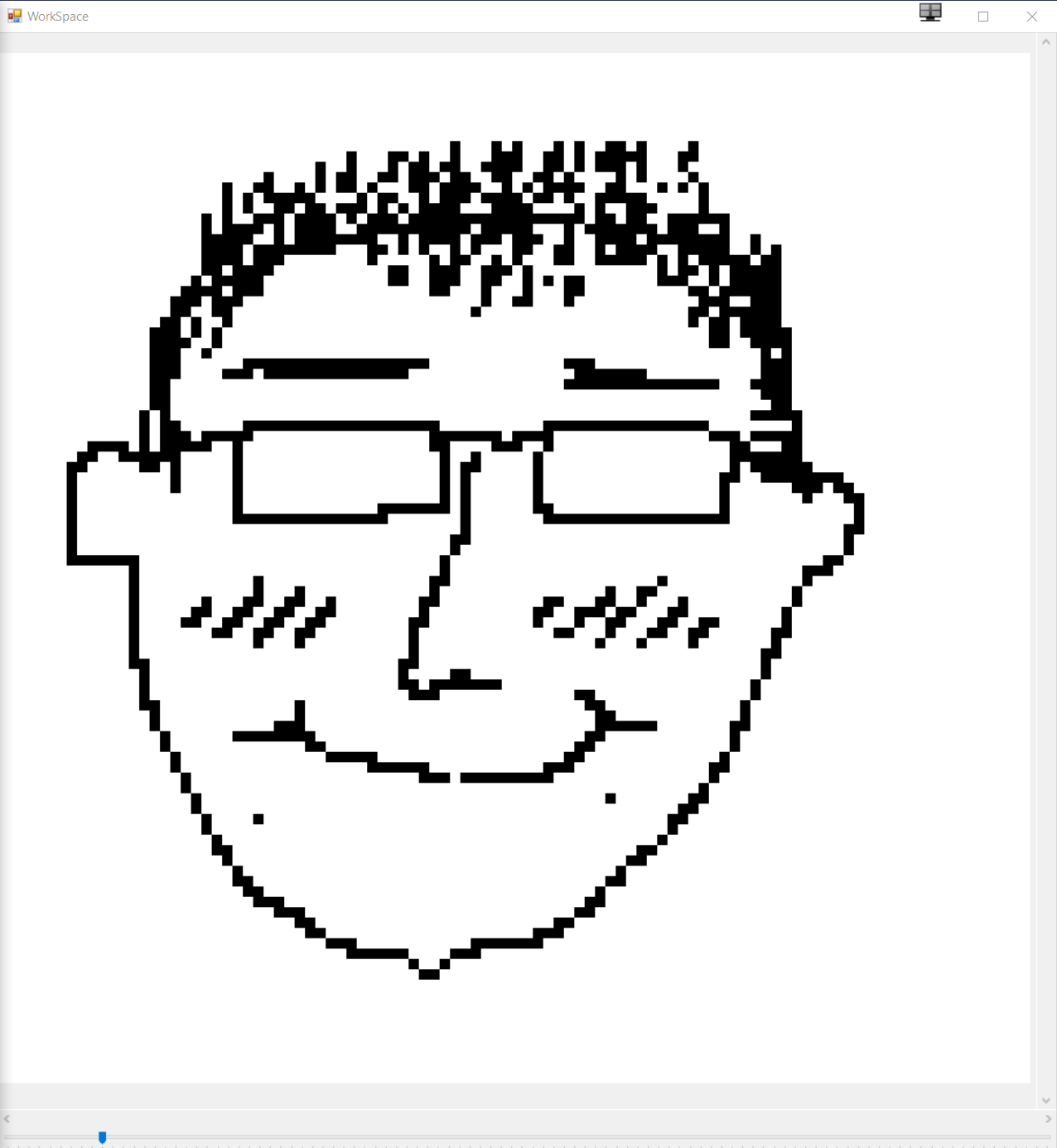
As the dimensions and updates are done *before* the padding is defined, so the program initially displays as if the padding is 0

A simple code re-order resolves this:



So the program now starts correctly.



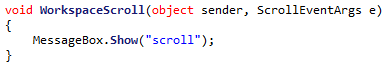
However, he was able draw an image 

### Scrolling Upgrade

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

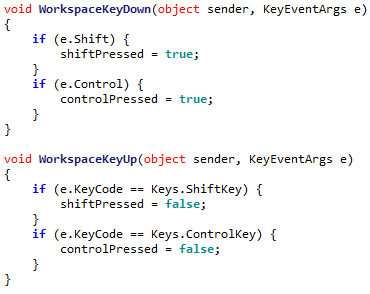
He suggested that scrolling normally should move the vertical bar upwards, pressing shift+scroll should move the horizontal bar, and pressing control+scroll should zoom in or out.

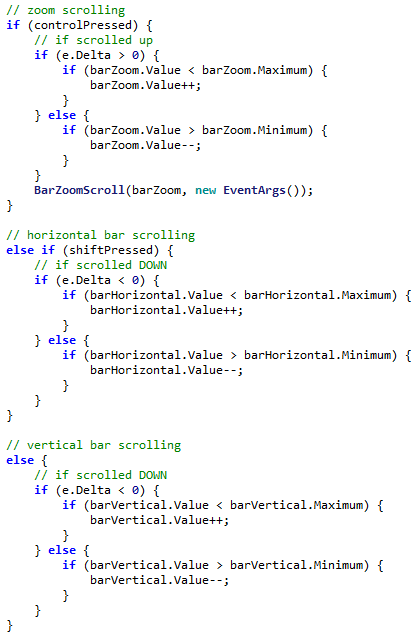
However, a first attempt to code this proved unsuccessful:



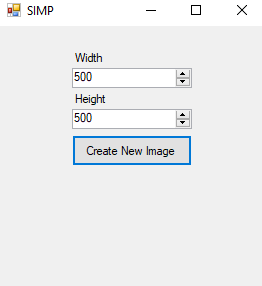
The MessageBox did not show. This was because the ‘scroll’ event is designed for controls that have an attached scroll bar. The WorkSpace did not have a scroll bar directly attached to it so did not trigger this event.

This can be solved by using the MouseWheel event:  


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

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


### Permitting Custom Image Sizes

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


### Optimising Rectangle Drawing

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

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



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



This then means the drawing code can be implemented as:



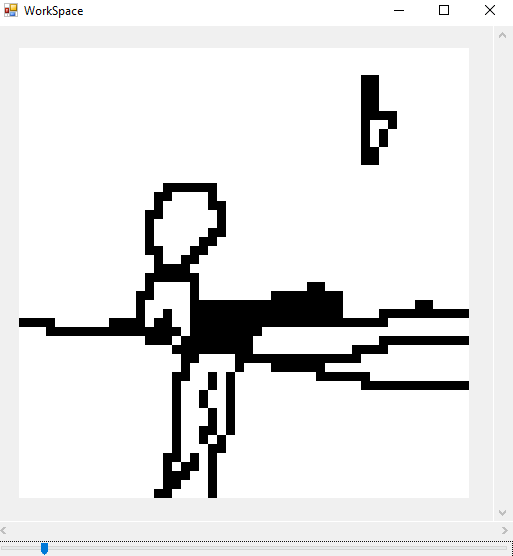
Which drastically reduces the code complexity, and increases performance.

## Alex H Client Feedback

### User Experience

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

Regardless he did draw a picture using the program



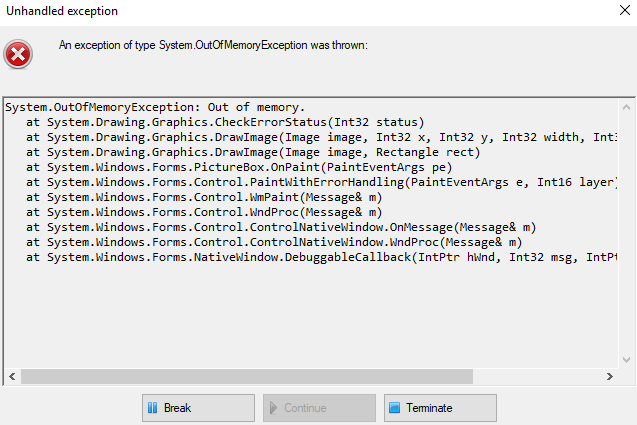
### Inconsistent Brush

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



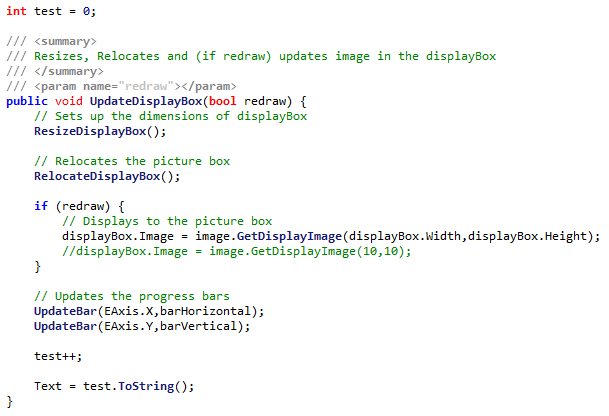
### Memory Leak Crash

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

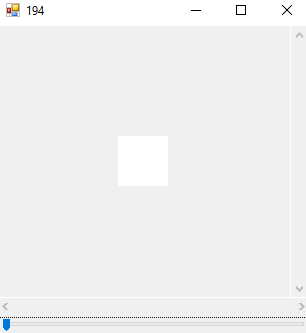
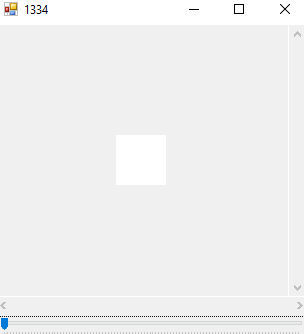


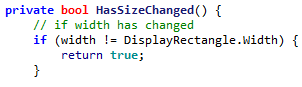
The error occurred because each time a new image is created, the old image is not removed from memory. This means if too many images are created at the same time then they are made faster than the garbage collector can delete them, and the program crashes.

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



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

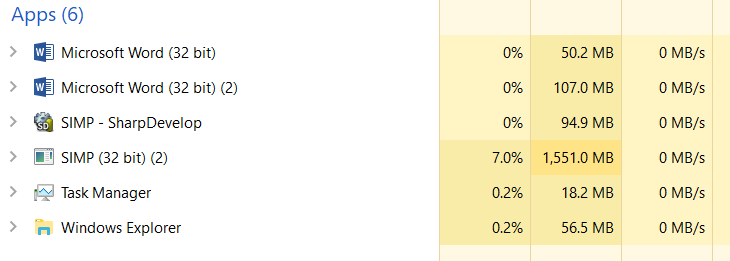
 🡪 

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

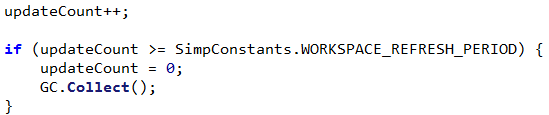
However was being set to the size of the form factored with padding:  

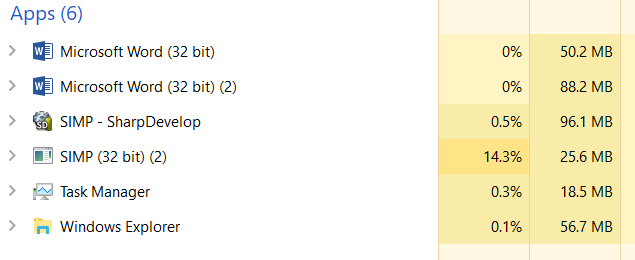

This meant that HasSizeChanged always returned true, and the image was updated many extra times per second.

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



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

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


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


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

## Dheshpreet Feedback

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



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

# 2.3.3 Success Criteria Evaluation

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **Proof** | **Code** |
| Section B – Other editing tools | | |
| Image viewer | Screenshot of a currently being viewed image | B1 |
| Bitmap image editor | Screenshot of a zoom in on the image showing the pixels | B2 |
| Zoom in (no zoom out) | Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller | B9 |

So this makes the full diagram:

Not completed

To be done this section

Completed

|  |  |  |
| --- | --- | --- |
| **Feature** | **Proof** | **Code** |
| Section A - Brushes | | |
| Variable brush width | Screenshot of strokes of the same brush showing different widths | A1 |
| Hard brushes | Screenshot showing the hard edge of the brush (colour to no colour) | A2 |
| Shape creation tools | Screenshot showing the shape toolbar and a small selection of drawn shapes | A3 |
| Fill (bucket) tool | Screenshot showing a before and after of filling a large area | A4 |
| Single pixel pencil | Screenshot showing a stroke of the single pixel brush | A5 |
| Rubber | Screenshot showing a densely packed picture being rubbed out | A6 |
| Section B – Other editing tools | | |
| Image viewer | Screenshot of a currently being viewed image | B1 |
| Bitmap image editor | Screenshot of a zoom in on the image showing the pixels | B2 |
| RGB colour picker | Screenshot showing a system for entering an RGB colour | B3 |
| RGB direct input | Screenshot showing the user entering “FF0000” (or equivalent) and the programming outputting red | B4 |
| Layer system | Screenshot of layer navigator | B5 |
| Rectangle selection tool | Screenshot showing a rectangle selection on the image | B6 |
| Magic selection tool | Screenshot showing a complex selection around non-linear shape | B7 |
| Transparent pixels | Screenshot showing a layer with blank pixels (one layer on top of another). Partial transparency is not required | B8 |
| Zoom in (no zoom out) | Screenshot of an image at smallest zoom, followed by a screenshot at max zoom showing a portion of an image much smaller | B9 |
| Text | Screenshot of the text “Hello World” on the image | B10 |
| Eyedropper tool | Screenshot of an imported image, with the colour stroke of a colour taken from that image beneath it | B11 |
| *Image effects* | *Screenshot of an image before and after an effect is applied* | B12 |
| *Rotating Images* | *Screenshot of an image in 4 different rotations, normal, 90°, 180° and 270°* | B13 |
| *Clipping masks* | *Screenshot of an image being clipped onto a complex selection* | B14 |
| Section C – File System | | |
| Creating a new image | Screenshot of a blank 300x300 square image | C1 |
| Importing images | Screenshot of the file browser showing an image preview, and screenshot showing the image in the program | C2 |
| Exporting images | Screenshot showing a custom image in the program, followed by an image showing the file browser showing the image in a folder | C3 |
| Supporting PNG and JPEG | Screenshot showing the file browser which accepts both PNG and JPEG images | C4 |
| Saving and loading from a proprietary format | Screenshot showing the user saving an image, screenshot of the image in the file browser, and the program after the image is loaded | C5 |
| Section D – Usability | | |
| Program should be stable and not crash. | A complete testing table, showing no failed tests, followed 75% yes response to asking stakeholders “Did you encounter any errors while using the program?” | D1 |
| Program should be easy to use | 75% yes response to asking stakeholders “Did you find the program easy to use?” | D2 |
| Features should be easily accessible | From the default state of the program, any feature will need to be activated by no less than 4 clicks | D3 |