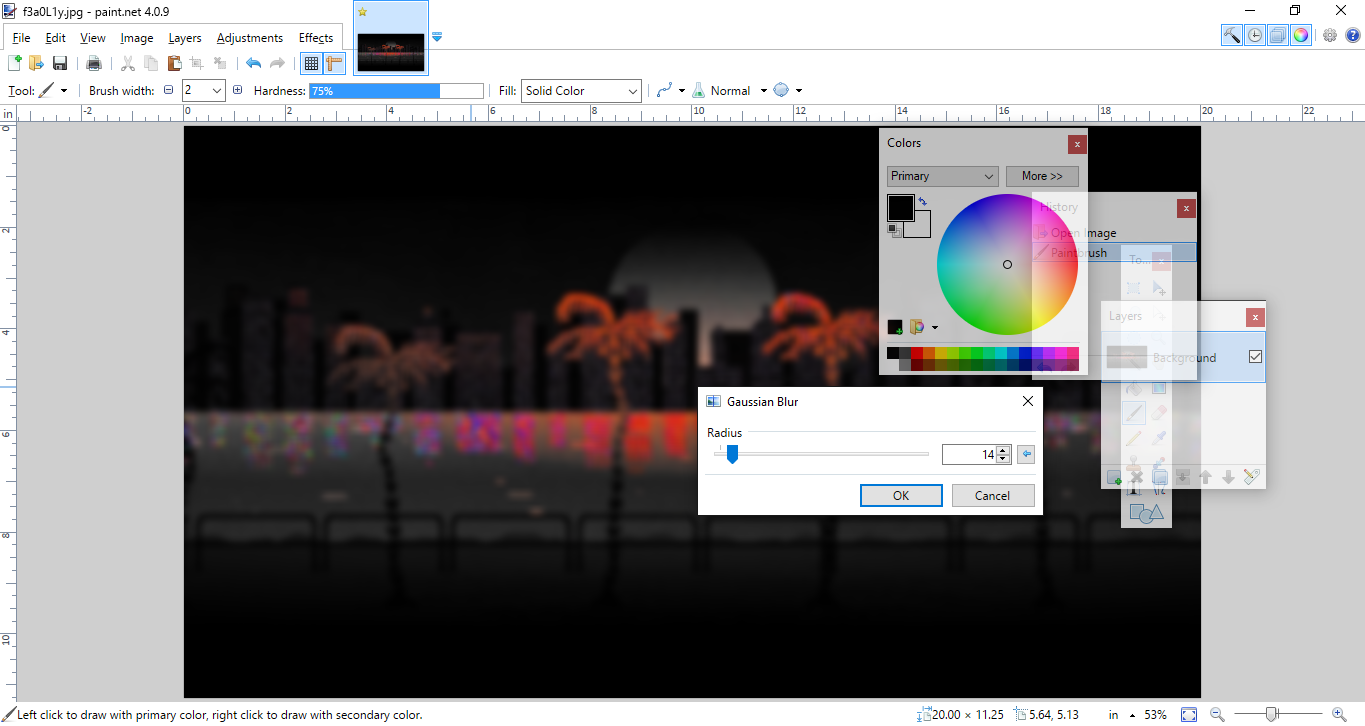
Final project

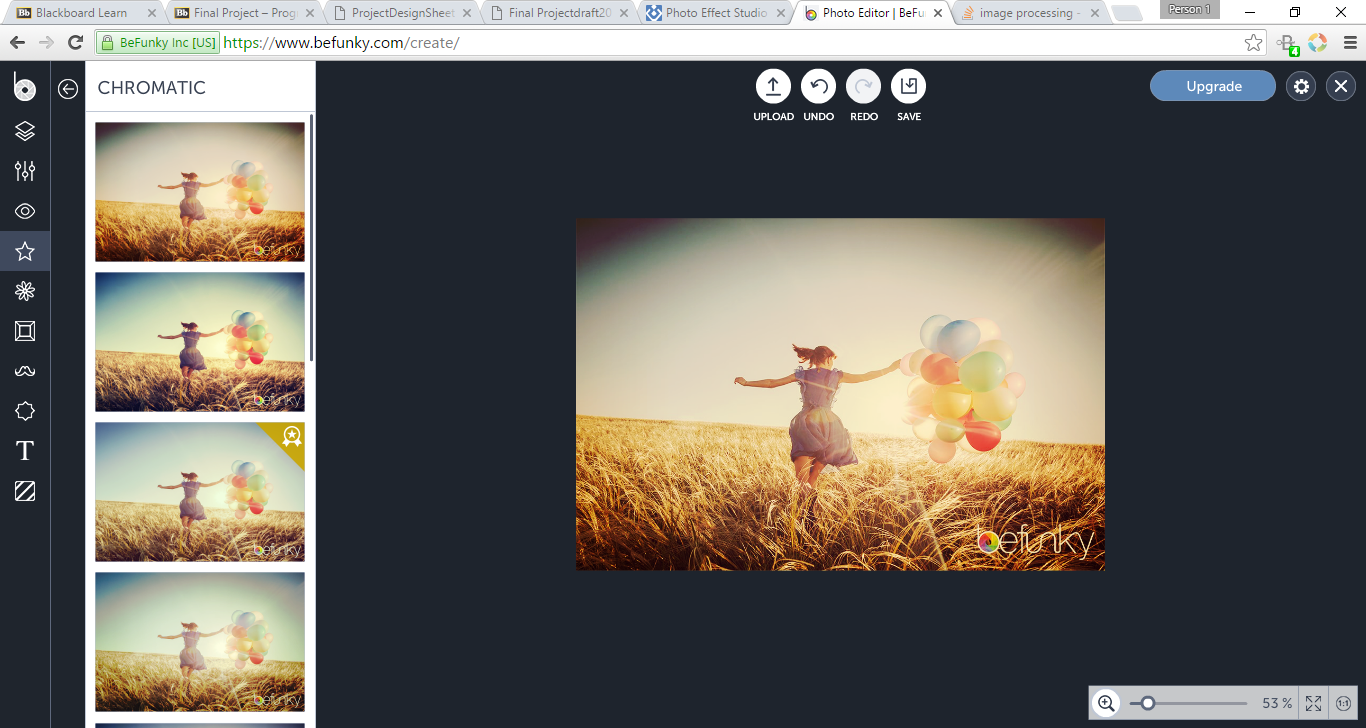
Analysis

Before the program requirements could be drawn up, several existing programs were reviewed to see what the key features are. By reviewing three programs the common characteristics can be derived and implemented in the project.

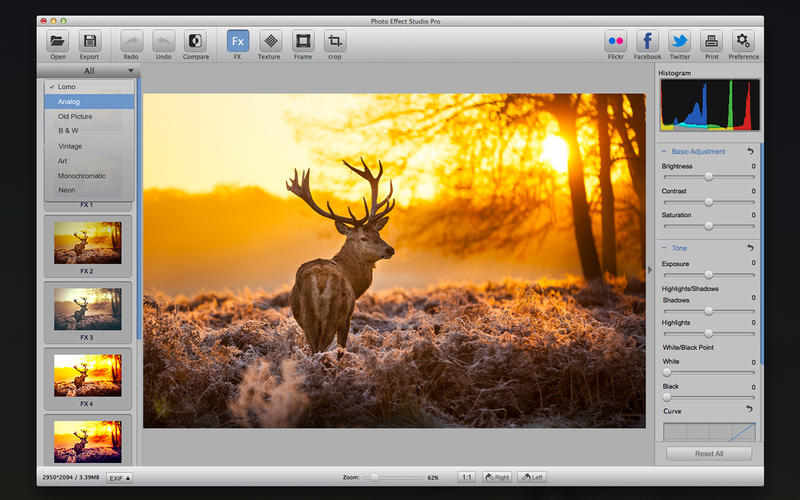
Program 1 (Paint.net): Paint.net is a photo editing software package for the Windows operating system. Paint.net is not just intended for image manipulation but has a list of effects which can used to modify photos in way similar to this project. This project will not offer the other options of the digital photo editing that are included in Paint.net such as image resizing, shape rendering and painting tools.



Program 2 (Befunky): Befunky is an online photo editor with an easy to use UI and variety of different effects. There are several unique effects and some which its shares with the other programs. Befunky’s effects are more focused on editing faces then the other two programs but does share the more popular effects with the others although they can be hard to fine as there is not a popular tab like Paint.net. The main difference between this program and the others are that Befunky is an online program where as the others are desktop appilications.



Program 3 (Photo Effect Studio Pro): Photo Effect Studio Pro has a similar UI layout to Befunky but allows more control of the effects as well as more of them. It can offers these additional effects as it is not restricted to being a web application as well as it being more expensive. Similar to Befunky it previews effects in the left pane.



All three of these programs focus on a clean, user friendly UI. All three do not just focus on one type of image manipulation instead offering lots of different effects such as adding ‘stickers’ or rendering geometric shapes. This in turn can stop the UI from being easy to navigate. By offering the most popular visual effects a very simple to operate program can be maintained.

Requirements

This section starts to lay out specifics of the actual project. It contains the requirements of the project and includes an informal program specification.

Program Requirements

* The program must read a picture file from a user specified file location.
* The program must offer different options of picture modification to the user.
* The program must successfully apply the correct visual effect to the picture.
* The program must offer the user the option to save the new image.
* The program must write the picture back to the same location (overwriting the previous photo).
* The program must do the whole operation in a reasonable amount of time, for example less than 20 secs.
* The program must be able to be executed successfully on the Lab computers.
* The program must handle the event if the user tries to open a non image file.

An optional feature will be to apply an effect to all images in a directory saving the user the time to activate the program for desired images.

Informal Program Specification

* The program must read a picture file from a user specified file location.

-The user will enter a memory location which the program will access and will save to a 2D array/vector. The user will enter the memory array by entering it into the console.

* The program must offer different options of picture modification to the user.

- There will a menu offering different effect options to the user. The user will select different options by entering the corresponding number which is displayed in the menu.

* The program must successfully apply the correct visual effect to the picture.

- Each effect will have a different mathematical operation applied to it. The operation will be tested to ensure the desired output is produced consistently.

* The program must offer the user the option to save the new image.

-The user may not like the new effect on the image so the option to not save the image must be available. This will implemented by a yes/no question after the image has been processed.

* The program must write the picture back to the same location (overwriting the previous image).
* The program will extract the pixels from the array and write them back to the original location of access.
* The program must do the whole operation in a reasonable amount of time, for example less than 20 secs.

- When writing the program nested loops will be avoided. Avoiding these will help to improve the efficiency of the program. Pseudocode and flow charts will be prepared prior to starting the coding stage.

* The program must be able to be executed successfully on the Lab computers.

- The program will be developed on these computers so compatibility problems should not exist. The program will also be complied into a executable so that it should run without any complications

Comparison with Existing Programs

This program differs from all the programs but in varying ways. It will deviate from all three by not attempting to be a full photo editing software package, instead being a streamlined program only offering a small selection of effect options and these operations will completed in shorter amount of time as you do not have to go through lots of menus as with the other programs. The project is not web based like Befunky and will instead be downloaded and run as an executable file (.exe). The user interface will also be more basic as it will be in the console. This is due to time constraints as well designed user interfaces take a long time to design and implement. A console UI is also sufficient for the level of complexity of this new project.

Design

Problem Decomposition

To make the problem easier to solve and program it must be broken up into smaller parts which can be implemented into sub-routines and functions.

These sections are:

* Ask the user for the desired image location to read from. The user will enter this into the console.
* Storing the image file into the program in a matrix format. The image will be stored in a 1D array.
* Show the starting image to the user.
* Ask the user to select which effect to apply to the image. Effects include a sepia filter, edge detection, a black white filter and a Gaussian blur. Although these may change.
* Add effect to all coordinates in the matrix.
* Displays the altered image.
* Asks the user if they want to save the new image.
* If so overwrites the old image.
* Does the user want to do anything else? (Exit or return to the beginning selection menu or return to effect selection).

Flowchart

C:\Users\Alastair\Downloads\Untitled Diagram (2).png

Development

The functions and sub routines are:-

**Main: -** This function is where the main menu of the program is and the options call the other functions depending on the users input. When the user loads in an image this function displays it to them to verify that the correct image has been loaded in. It also contains a loop which asks the user if they want to load another image from a file once they have edited an image and decided whether to save it or not.

**DisplaySurfaceUntilClose: -**This function was given in the SDL wrapper and has not been edited. It keeps the image open until the user closes the window.

**LoadImage: -** Images are loaded through this function. First the user is asked for the directory of the image and will repeat asking if an invalid directory is entered. Then the image is loaded using a function from the header “SDL\_Wrapper.h”. Then if the image cannot be loaded it will provide an appropriate error message then ask the user for another directory.

**SavingImage: -** This function is called when an effect is applied and the user must decide whether they want to save the new image. The first question asks if they want to say the new image; a no will end the function but a yes will pass them onto the next question. The second question is if they want to overwrite the current image, a yes will save the new image in the same file location of the original image whereas a no will prompt the user to enter a new directory address to save the new image. The loops in this function are for asking the questions again if the user enters data that lie within the boundaries of valid data.

**Truncate: -** This function is called at the end of a calculation before the value is assigned to the Uint8 variable. If checks the number and if its greater than 255 it sets it equal to 255 and if its less than 0 its sets it to zero. This is to stop overflow and underflow in the images.

All the functions following these have the same structure of looping through the image array for all xy values, starting at the bottom left pixel and finishing at the top left.

**BlackandWhite: -** This function makes the loaded image black and white. All the red,blue, green(RGB) parts of a pixel are set equal to the sum of the RGB values at the pixel multipled by different values differing between red,blue and green. The multiplication is compensate for the imperfections in the human eye to make it appear to be a true black and white image.

**Sepia: -** The loaded image is turned sepia by the one. The RGB values are read for each pixel and changed by a formula which turns the pixel into its brown colour. The truncate function is called here to prevent the RGB values from overflowing. The RGB values are then saved back to the array.

**Brightness: -** This effect was easy to implement as the user is asked for a value in between ± 255 to alter the image by. The user then enters the number they want to alter the image by and that value is added to the image. The new value that overwrites the old red, green, blue is prevent from overflowing or underflowing by again using the truncate function. Again the new image is placed in the array.

**Contrast: -** Similar to the last effect the user is asked for a value between -255 and 255 values to alter the contrast, with ±255 altering the contrast the most whereas a value 0 does not change the image. The value entered is converted to the value used in the calculation by making the value a ratio of 255 in between 0 and 129.5. This contrast value is then multiplied by RGB values-128. After the calculation the 128 is added back onto the value. The new pixel replaces the old and the process happens again for the next pixel.

**ImageInversion: -** The output of this function is a ‘Negative’ of the image. This effect is produced by subtracting the current RGB values from 255 for each. This is done for all pixels in the image.

**Mirror: -** This function is slightly different from the rest by have two counters which start either side of the centre line of pixels and then traveling in opposite direction with the pixel at the right pointer being set to the pixel at the left pointer , creating a mirror effect. The right side is the side which is mirrored. Once the counters have reached the edge of the image they will reset to their original state either side of the centre line of the image and then work their way through that row. This action is repeated until the bottom of the image has been reached.

Testing

It is essential to prepare a test plan before testing the program is make sure areas of the program are tested to a sufficient standard. This plan will outline all the tests to be performed on the program in an attempt in find as many as bugs as is possible.

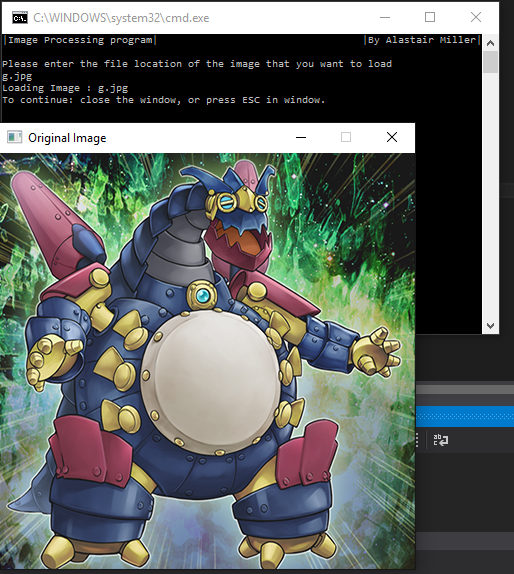
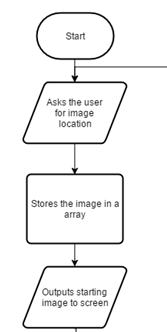
|  |  |
| --- | --- |
| Test number | Purpose of test |
| 1 | Test the flow of control: Follow all menu options to check whether they lead to their respective sub-routines. Users can only choose appropriate options.( Top-down testing) |
| 2 | Correct validation of the input data. (Bottom-up testing) |
| 3 | Images are opened and saved correctly .(System testing) |
| 4 | All decision and loop statements are completed correctly, as well as all program calculations.(White box testing) |
| 5 | The system produces the correct output results based on the specification.(Black Box testing) |

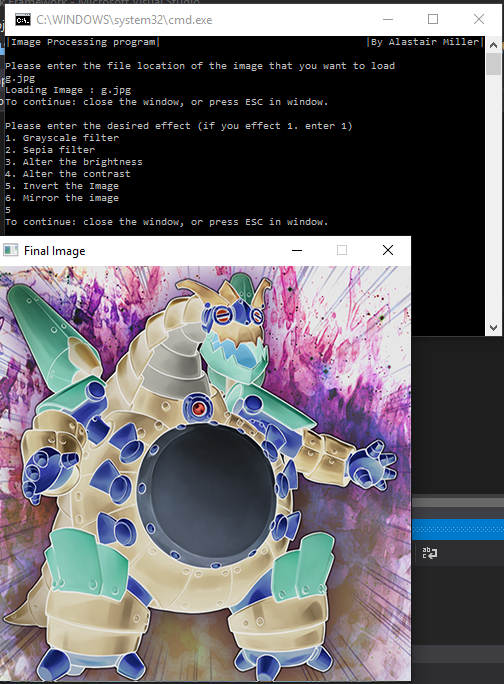
Detailed Test Plan

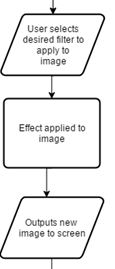
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | Purpose | Description | Test data | Expected result | Actual result |
| 1.1 | To check if all parts of the flowchart can be reached in the actual program. | The flowchart will be followed through in the program until all blocks have been reached and then the program should be ended by reaching the end block. | N/A | Success | Success, the all parts worked as expected.  Figure 1.1a |
| 2.1 | Validate the input to load the image. | A test to check if the program can cope with expected and unexpected user input | g.jpg  g  C:\Users\Alastair\Desktop\p.jpg  C:\Users\NOTREAL.jpg | Success  Fail  Success  Fail | Figure 2.1a  Figure 2.1b  Figure 2.1c  Figure 2.1d |
| 2.2 | Validate the input for the menu. | A series of tests to check whether a user can successfully navigate to the features they want. It is also to test if the program can handle invalid input. | 1  Abc  6  3 | Success  Fail  Fail  Success | Figure 2.2a  Figure 2.2b  Figure 2.2c  Figure 2.2d |
| 2.3 | Validate the various yes/no statements | This test is to check if the yes/no statements are working as expected with error handling. Only one statement will be tested as they have all been coded the same. | Yes  Y  y  n  123241 | Fail  Success  Success  Success  Fail | Figure 2.3a  Figure 2.3b  Figure 2.3c  Figure 2.3d  Figure 2.3e |
| 3.1 | Opening images | Make sure that images are loaded into the program correctly. | A single pixel  A very large image  A standard size image | Success  Success  Success | Figure 3.1a  Figure 3.1b  Figure 3.1c |
| 3.2 | Saving images | Apply effects to these images used in the previous test to see if these images can be saved. | A single pixel  A very large image  A standard size image | Success  Success  Success | Figure 3.2a  Figure 3.2b  Figure 3.2c |
| 4.1 | To test if all decisions are made as expected in the specification | Each effect and all yes and no statements will be tested to make sure they meet the specification. | N/A | Success | All effects and decision acted as expected and met the specification. |
| 5.1 | Does the program sufficiently meet the specification in regards to output | An image will be processed through the program for each effect and saved to a different file location | g.jpg | Success for each effect and the saving | Figure 5.1a  Figure 5.1b  Figure 5.1c  Figure 5.1d  Figure 5.1e  Figure 5.1f  Figure 5.1g |

Test result evidence

Figure 1.1a:

The following evidence is one of the ways through the program.





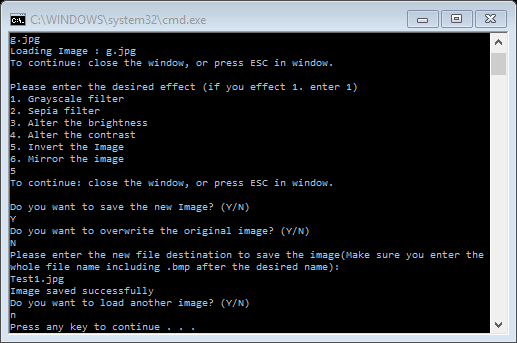
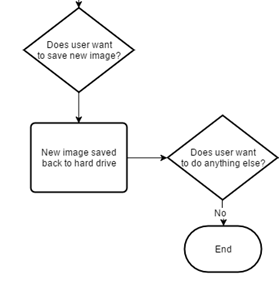


Figure 2.1a:

Success(As expected)

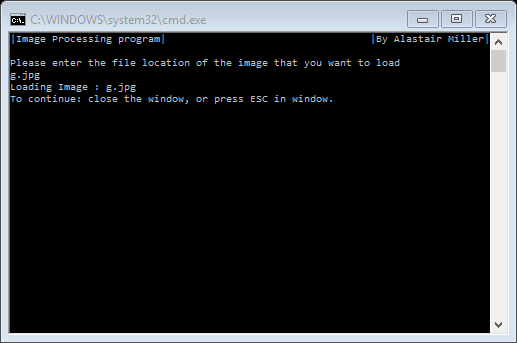


Figure 2.2b:

Fail(As expected)

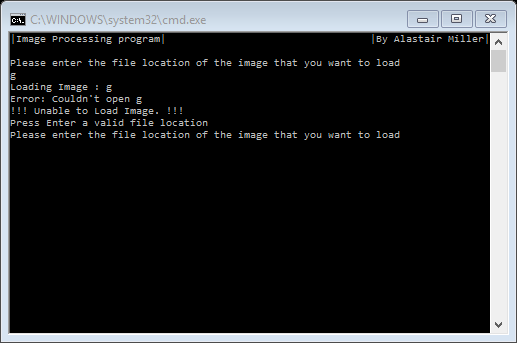


Figure 2.2c

Success (As expected)

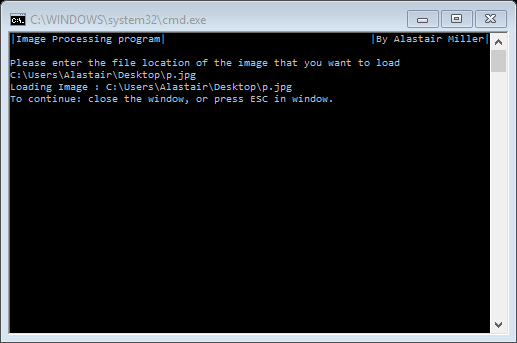


Figure 2.2d

Fail(As expected)

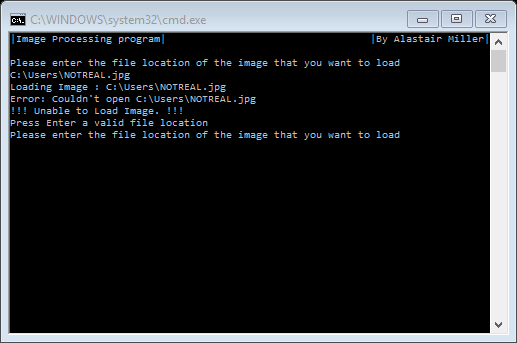


Figure 2.3a

Success(Unexpected)

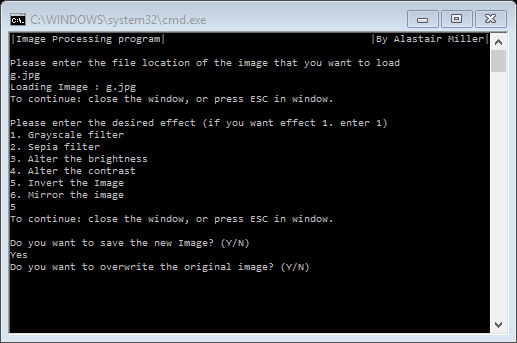


Figure 2.3b

Success (expected)

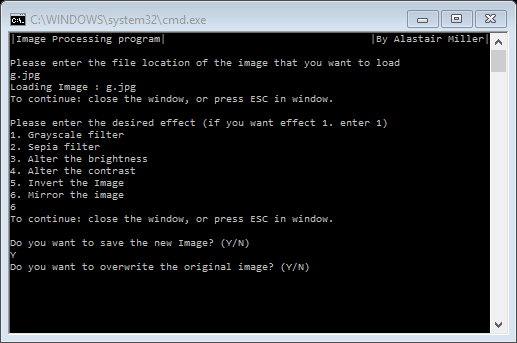


Figure 2.3c

Success (expected)

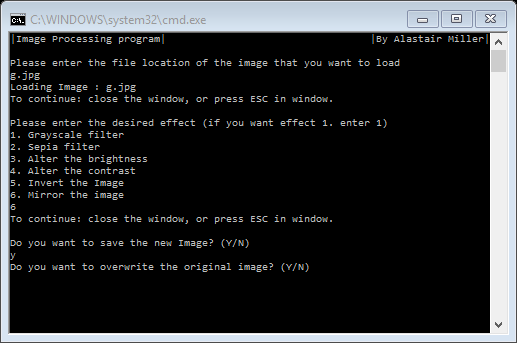


Figure 2.3d

Success (as expected)

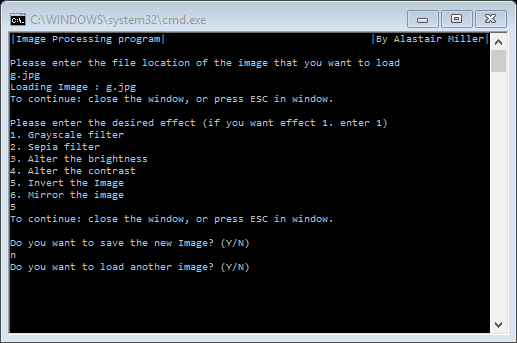


Figure 2.3e

Failure (as expected)

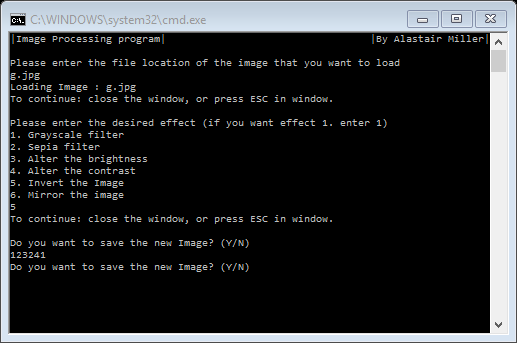


Figure 3.1a

Success (As expected)

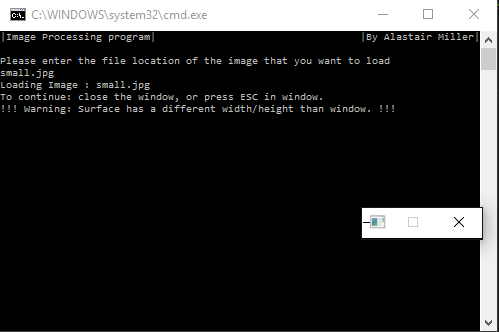


Figure 3.1b

Success(As expected)

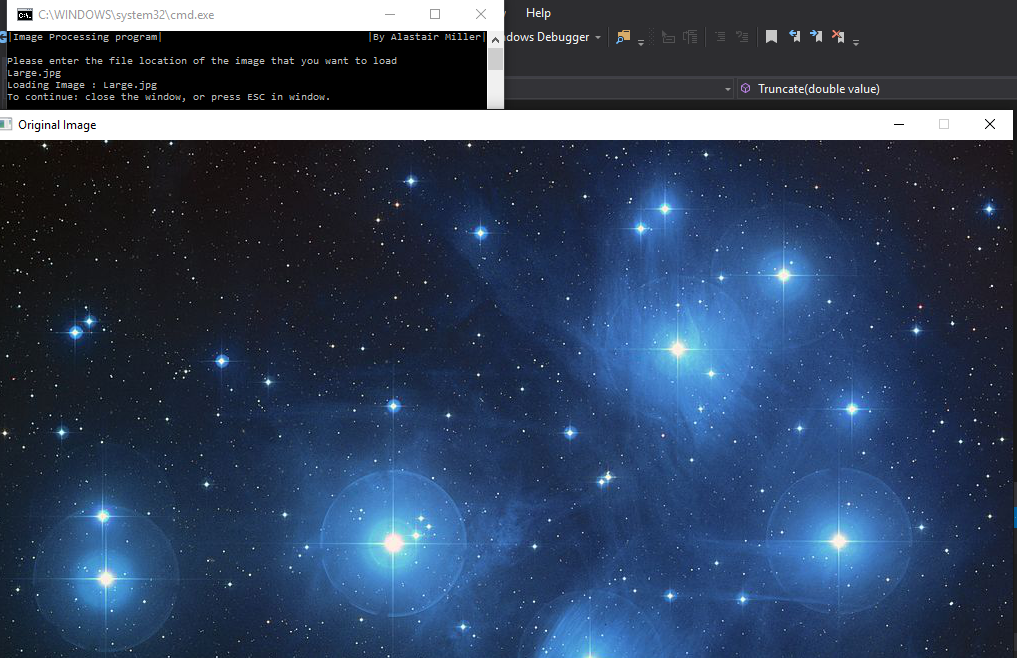


Figure 3.1c

Success(As expected)

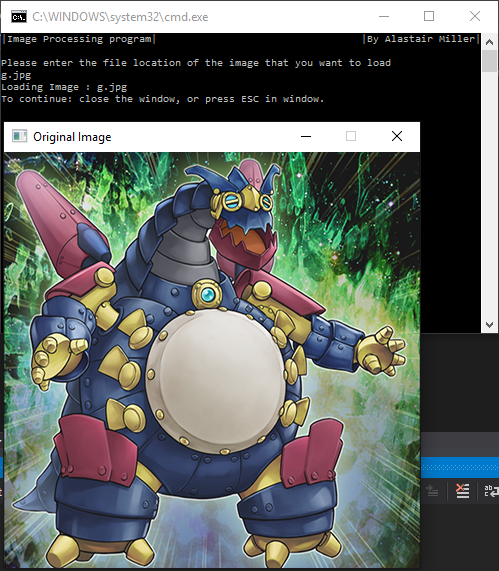


Figure 3.2a

Success(As expected)

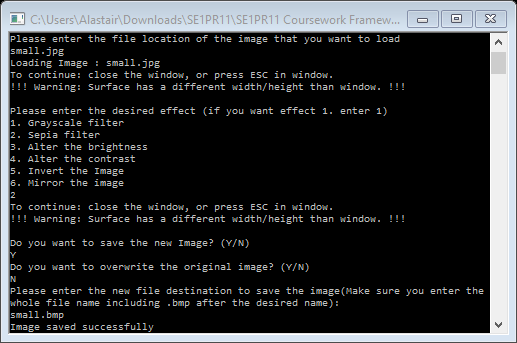


Figure 3.2b

Success(As expected)

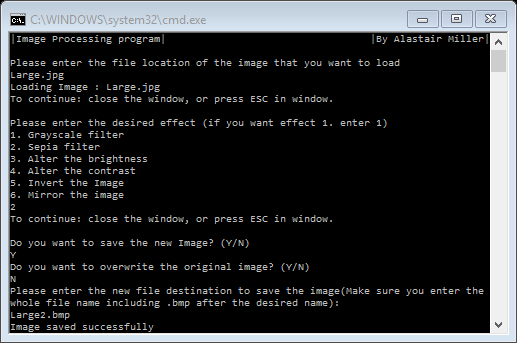


Figure 3.2c

Success(As expected)

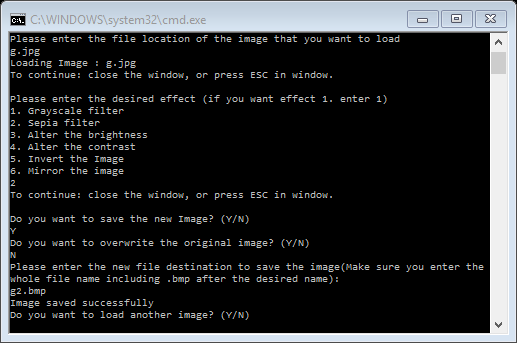


Figure 5.1a

Success(As expected)

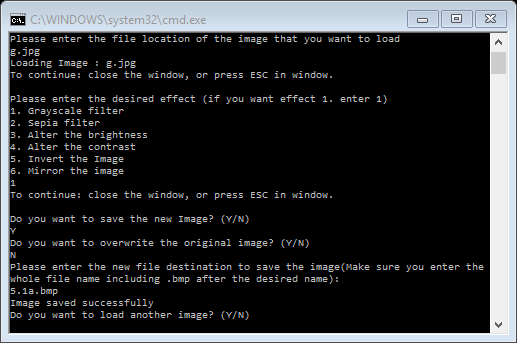


Figure 5.1b

Success(As expected)

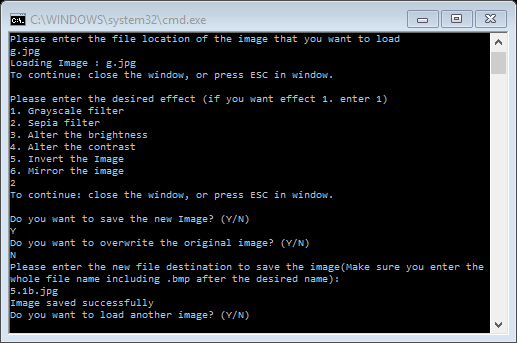


Figure 5.1c

Success(As expected)

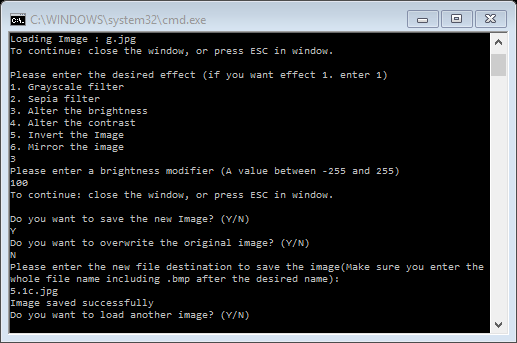


Figure 5.1d

Success(As expected)



Figure 5.1e

Success(As expected)

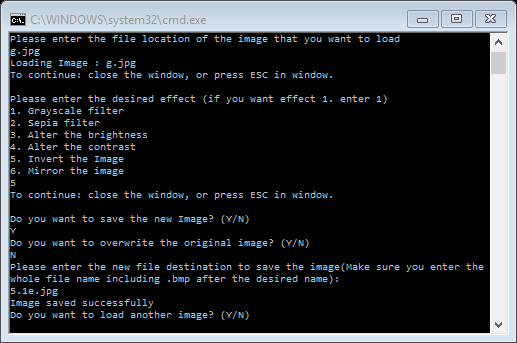


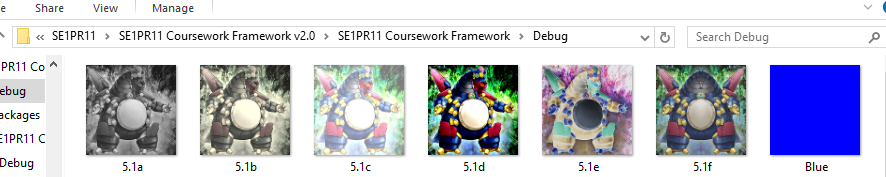
Figure 5.1f

Success(As expected)



Figure 5.1g

Success(As expected)



All images saved correctly.

Results from testing

All the tests were a success apart from one 2.3a, it has read only the first character which is a Y hence why it passed and is on to the next question. The reason this has occurred is because the data that is inputted is stored in a char as only one character is stored. It has been decided to keep this bug as if someone entered the whole word they would still get the desired selection. Overall no changes were made to the program.

Conclusion

I have improved many skills over the course of this project, these range from the obvious skills such as C++ programming to other skills such as improve to my document writing skills and how to test a system more comprehensively. I found that it easy to create the basic flow of the program; opening and saving the image before adding effects as sub-routines. By breaking the problem into smaller sections it became easier to solve and this was success so I will apply this method to all future projects I develop. The most obvious difficulty was that the inexperience with SDL and its functions. I solved this by reading up in its uses and how it can be implemented. I also sketched how some of the filters would be implemented.

If I encountered errors I would first dry-run the code, to try and find why the program had encountered the error. This method usually worked but if I could not identify the error I would a ask one of my friends to review the section of code for me. In this project all the problems I encountered where identified with one of these methods. I used techniques from the Software Engineering module to optimally test the program to ensure that undetected bugs are kept to a minimum.

Review

All in all, I believe this project was successful and I have learnt different skills then the other programming tasks as there has been much more documentation that has needed to be produced. I have discovered two different ways of effectively finding bugs in programs. In a future version I would implement a feature to edit a whole directory of images instead of just one at a time, this would save the user loading the image in each time. I would also like to have the user have more control over the effects like instead of having to enter a number they could move a slider bar and preview the effect before they proceed. Looking back I would have changed the program to open file explorer to make it easier to find the desired image instead of entering the exact file name.