CMP2090M

OBJECT-ORIENTED PROGRAMMING ASSIGNMENT REPORT

Joseph Doody, [Student ID: 16603020]

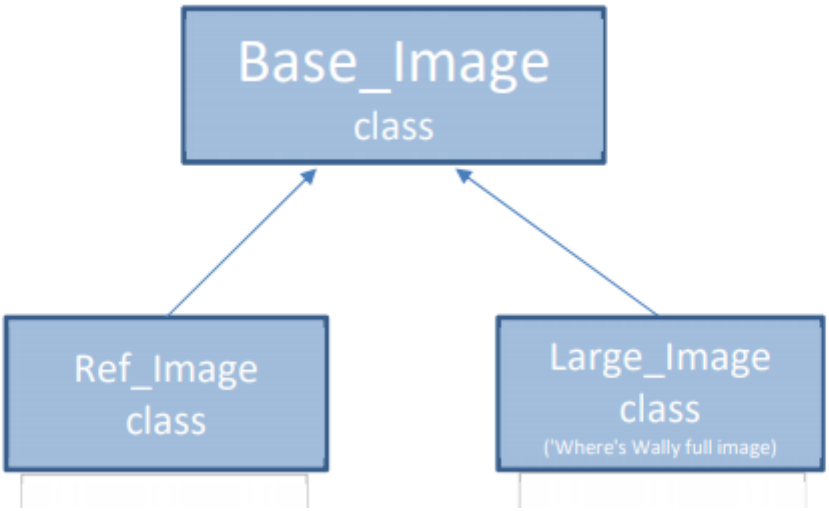
1. INTRODUCTION

For this assignment I was to use Object-oriented programming techniques to solve a problem using a search known as “Nearest-Neighbour Search”. The coding language I would be using is C++ because I believe this is the easiest way to demonstrate Object-oriented programming with the use of headers and classes.
The task was based on the game, “Where’s wally”, where you will have an image that will be populated with multiple characters and you must find Wally within the crowd. I have been supplied with two .txt files which hold values which illustrate pixel values for both images. The image will both be in a grayscale.
As stated earlier I will be using the Nearest-Neighbour Search algorithm to find Wally within the crowded picture. *Nearest neighbour search, as a form of proximity search, is the optimisation problem of finding the point in a given set that is closest (or most similar) to a given point. [1]*

2. PROGRAMME STRUCTURE

To solve this problem, I first created a matrix class, the reason to why I made this class was because images work in matrices, you have your rows and columns illustrating where the pixels are on the image. The way I did was, is I would create a two-dimensional array to store data. I would then create functions which would allow me to manipulate data within the matrix, print values within the matrix, and reference certain values within the matrix. I chose to make these functions because I believe these would help with what I need to do with my problem. This was not going to be a class I would use within my program, it was a template class which I would use to manipulate different images.
For my main program I would have three different classes all based around the matrix class, the first class would be the crowded image, second image would be the Wally image and the third class would be a reference image. I would read all the data within the crowded image .txt file and then store all the values into the matrix assigned to that images class (Base_Image class). I would then do the same for the Wally image and assign it to its image class (Large_Image class). This would then leave me with the last class (Reference Image class), I would also set the crowded image to this class however the matrix for this image was not going to show the whole image, it would show a small area, a similar size to what the Wally image.

The referencing image would be the main class I would be using that will help me to find where wally is in the crowded picture. It would follow the inheritance hierarchy show below.



I would then use the NNS algorithm (Explained in the next section) to find a match to where Wally is within the crowded picture and once he has been found I would illustrate where he is in the big image by shading the area black where he is and write the final image (Crowded image) named as "found.pgm".
I have also made use of overloads within my program, I call the constructors at the start of my program and then calling the destructor when I have finished with the class.
Memory management has also been used that when I'm finished with the variables which are used for reading the images, they are then deleted.

3. NNS ALGORITHM

The type of NNS algorithm I used was a linear kind. This kind of method for NNS where it computes the distance from the query point to every other point in the matrix, keeping track of the "best so far" [2].
So, I would use the reference class, this would create a matrix based on the crowded image and create it in a size similar of the Wally image. I would then compare both matrixes by comparing pixels at same index points in the matrix. The reference that had the most pixel matches would be set as the "best so far". I would use a "checker" that would tell me what has the best match to then set to another "checkermax" which would store the best match. The way I would analyse each point in the matrix I would have a value which would look at an index point in the crowded image and then create the image that way into the matrix. To then increment that further I would have a point variable which will store the next index point in the crowded image which I can go back to start the next iteration of the search.

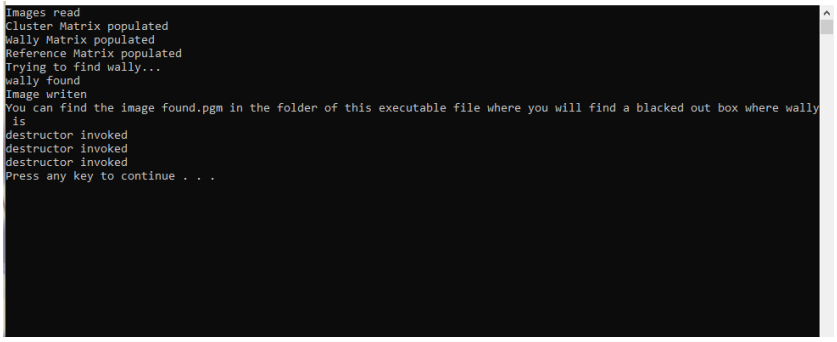
Pseudocode:

```
int I = 0
int point = 1;

While (point < pixel amount in image)
        Int row = 0;
```

```
Int maximumrows = 0;
Int columns = 0;
Int maximumcolumns = 0;
Int checker = 0;
Int checkermax = 0;
For (row; row < maximumrows; row++)
        For (columns; columns < maximumcolumns; columns++)
                Referenceclass.set pixel[row, column];
                If referencepixel == wallypixel
                        Checker ++;
                I++;
        I += nextrowstart;

If checker > checkermax
        Checkermax = checker;
```

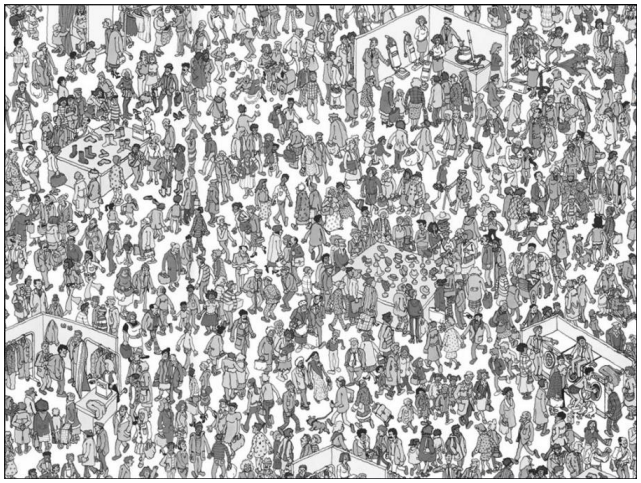
4. RESULTS

In the end the program was a success because I was able to find where wally is within the crowded image. I had outputs within the console so that the user knows where the program is.

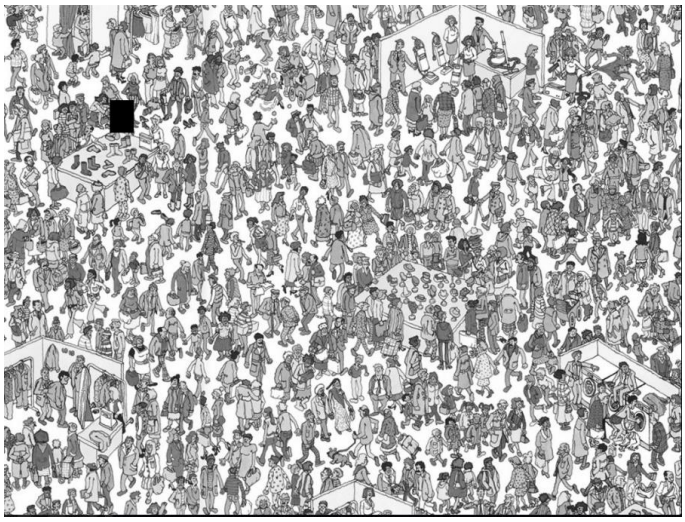


4.1 Best matching

The original image:



Wally Found:



The images may not clear within the word document however if you open the images in the folder after running the program you'll be able to view the found image and notice the area that has been shaded out black, that's where Wally is.

5. Discussion & Conclusion

The main thing for my program is that it was a success because I was able to find where Wally is within the image by using a Nearest Neighbour search algorithm. I made sure I used Object-Oriented programming techniques from using classes and using them within my main program. C++ techniques such as arrays and memory management were used well within my program.
If I was to look over this task again, a major improvement I could make to the program is the efficiency in which it runs, currently the program runs well with no errors however from start to end it takes two minutes for it to compile by reading the images, going through the NNS algorithm to find Wally then write the image. Maybe using a more efficient algorithm instead of a linear algorithm would be better. Something I could use in the future for better efficiency could be a K-d tree which is a used to organise some number of points in a space with k dimensions. It is a binary search tree which is very useful for nearest neighbour searches [3]

References

1. Nearest-Neighbour Search Definition: https://en.wikipedia.org/wiki/Nearest_neighbor_search
2. Nearest-Neighbour Search Linear: Section 2.1.1 https://en.wikipedia.org/wiki/Nearest_neighbor_search
3. K-d tree: http://www.pointclouds.org/documentation/tutorials/kdtree_search.php