

CMP2090M

OBJECT-ORIENTED PROGRAMMING ASSIGNMENT REPORT

Kiran Thomasson, [Student ID: 15625218]

1. INTRODUCTION

We were tasked to build a C++ solution which makes use of a Nearest Neighbour Search (NNS) algorithm to find an item from a reference image inside of a larger image. In this case, we were given a base image of wally and a scene in which to find him. NNS has many different ways of implementing and many different methods which can produce differing results. I decided to make use of the Sum of Squared Differences algorithm as it is best suited for dealing with numbers such as in a matrix for an image. I also decided to use the moving window approach in which the algorithm works through the image giving each window a SSD value and stores the lowest result. This is the closest match to the reference image, meaning locating the subject's nearest match is a simple task.

2. PROGRAMME STRUCTURE

The structure of my program makes use of many different OOP features including classes, objects, attributes and functions. My program makes use of 5 classes. Ref_Image and Large_Image which are child classes of Base_Image, are used to store data such as the rows and columns of the matrices needed. The matrix class which is used to create filenames and pointers to allow for the saving of the matrices. Lastly, the NNS class which contains my SSD function as well as the 'tempmatrix' vector. All of these classes except for the NNS class contain constructors and virtual destructor functions which are used to set up attributes and also to delete objects. I made use of objects of my classes in order to pass data into the main function. This allows my code to be efficient with memory usage as it allows objects to be deleted from memory when they are no longer in use. I also made use of functions in my program. One such function performs the sum of squared differences for each of the windows. It takes 2 matrices, the number of rows & columns and returns the result as a double value. When the program is started, the 3 objects (wallyobj, baseobj and matrixobj) are created. The program then creates 4 variables to store the rows and columns of the 2 matrices. Next, the program reads the 2 text files and writes the 2 pgm files using the ReadWrite.cpp file. After this, 2 vectors are created to store the numbers that have been read from the text file. I chose to make these unsigned char vectors as this speeds up the iteration of the values as the vector does not need to store the zeros after the decimal point. The code then performs 2 sets of 2 nested 'for' loops which import the data from the 1D array into the vector. These loops rely on the $k = i * n + j$ equation to place the values in the correct slot so that they stay in the correct form for a 2d array. After this, the program deletes the matrixobj to free memory, as it is no longer needed. It calls a loading animation on a new thread and creates a new object called Searcherobj, which is an object of the NNS class. Now the searching algorithm begins. It is comprised of 4 nested 'for' loops which create a moving window the size of wally, which will

iterate through the base image, giving each section an SSD value. This works by moving the box by one column to the right on each iteration, creating a temporary matrix from the base image and then calling the SSD algorithm and passing it the two matrices. After the NNS algorithm has finished, the main function then takes the value returned and checks if it is lower than the lowest value recorded. If it is lower than the number along with the I and J integer values to record the location of the window when the SSD was lowest, thus this is a nearer match for Wally. After the loops have finished the result will be the array containing the values of the lowest SSD and the location of the window in the form of I and J. This array is then used in a final iteration. In which, the program re-writes to a new matrix, it uses the data from the base image, until it reaches the area where Wally is located, in which it draws a black grid over the area. The matrix is then passed to the writepgm function, which writes a new image called "highlightedscene.pgm" which can be seen below in section 4 (Results). The code now completes and displays a message to say that Wally has been found. It then deletes the remaining objects to free memory and ends.

3. NNS ALGORITHM

Below is the pseudocode for my NNS algorithm. I also provide a line by line description of my algorithm on the next page. I chose to use a Moving-Window Sum of Squared Differences (SSD) algorithm to solve the problem given. I chose to use this algorithm because of its ability to take into account a window of a certain size and compare it to a reference image. It then provides a number based on the similarity of the 2 matrices, the lower the number is, the closer the 2 matrices, the more similar they are. With a result of 0 being an exact match. My algorithm uses a window the same size as Wally's reference image and iterates through the base image, giving each area a SSD. It then passes these back to the main function which will record which positioning of the moving window gave the n nearest fit for to the reference of Wally.

```

1 NearestNeighbourSearch(matrix1, matrix2, rows, columns)
2 {
3     var SumOfSquaredDifferences;
4     var Difference;
5     var k;
6
7     for (i=0; i<=rows - 1; i++){
8         for (j=0; j<=columns - 1; j++){
9             k = i * columns + j;
10            if (matrix1[k] == 255) {
11                matrix1[k] = matrix2[k];
12            }
13            difference = matrix1[k] - matrix2[k];
14            SumOfSquaredDifferences += difference * difference;
15        }
16    }
17    return SumOfSquaredDifferences;
18 };

```

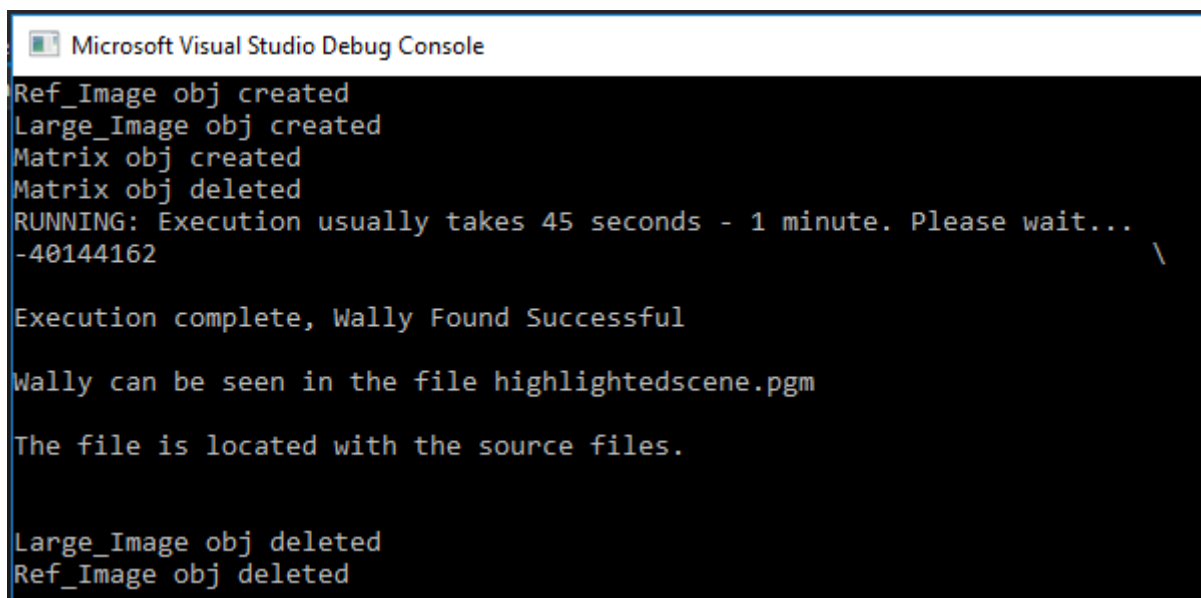
```

1 //Beginning of function which takes 2 vectors and 2 integers
2
3 //Variable to store the sum of squared differences
4 //Variable to store the difference between the 2 numbers
5 //Variable to store the location of the numbers based on  $k = i * N + J$ 
6
7 //Beginning of first loop to iterate the rows
8 //Beginning of second loop to iterate the columns
9 //Assign K based on I and J
10 //If statement to search for white spaces in image
11 //White space is replaced with pixels from base image
12
13 //Calculate difference between the 2 pixels
14 //Square the difference and add it to the Sum
15
16
17 //Function ends and returns the Sum Of Squared Differences to main
18 //The lower the SSD, the closest match the selected window is with the base image

```

4. RESULTS

My program outputs a PGM file with the title “highlightedscene.pgm” It contains the entire base image except the area where the algorithm found the best match for wally is shaded in black. My program consistently outputs the correct area for wally and I have checked this on multiple runs. Below is a screenshot of the output of the console window from my program as well as the highlightedscene.pgm image which contains wally as a shaded rectangle, found in section 4.1.



```

Microsoft Visual Studio Debug Console
Ref_Image obj created
Large_Image obj created
Matrix obj created
Matrix obj deleted
RUNNING: Execution usually takes 45 seconds - 1 minute. Please wait...
-40144162

Execution complete, Wally Found Successful

Wally can be seen in the file highlightedscene.pgm

The file is located with the source files.

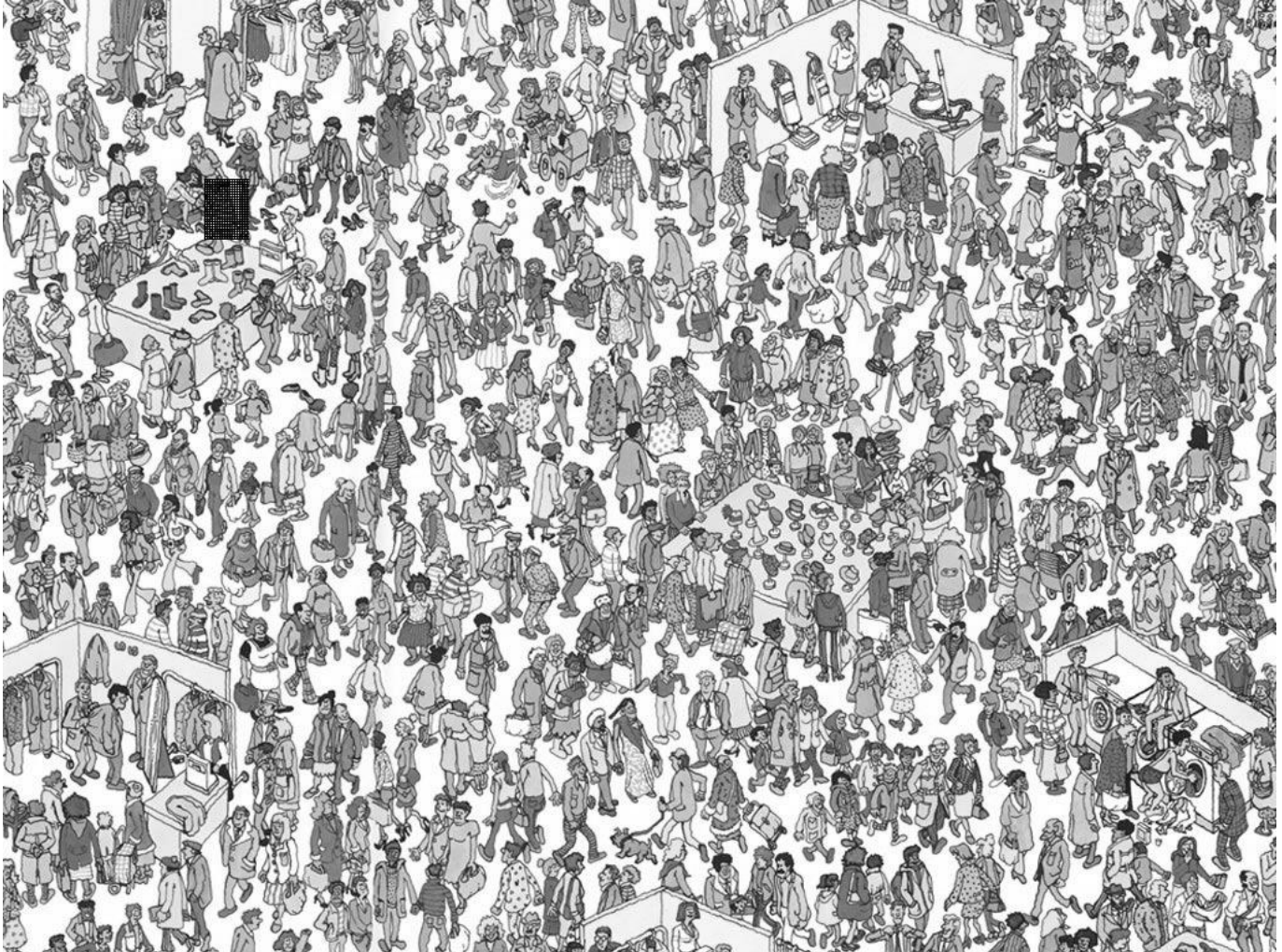
Large_Image obj deleted
Ref_Image obj deleted

```

In the console window, you can see when objects are being created and deleted. The program then outputs the result array when the NNS has finished. This number can be broken down into 40,144,162. Where 40 is the lowest result of the SSD, 144 is I and 162 is J, being the location of the best match for wally.

4.1 Best matching

The best match for wally is located where $I = 144$ & $J = 162$. This is the location on the base image for where the top left corner of the best match can be found. This can be seen below in the highlightedscene.pgm file. Wally can be seen in the top left of the image, within the shaded area.



5. DISCUSSION & CONCLUSION

The results of my program prove that the algorithm works well and does successfully find wally from the provided template. However, there are some improvements that could be made to my program. One such improvement would be with memory management and reducing recursion. The size of the vectors and arrays is large which does use a large amount of memory, and the recursive nature of the nested for loops does cause a large amount of the stack to be used. These are minor details that could be fixed with some optimisation and use of more efficient structures, however the program still functions as intended and finds wally in the correct location, thus solving the problem given.

REFERENCES

MICC (2013) A33Matching. Available from http://www.micc.unifi.it/delbimbo/wp-content/uploads/2013/11/A33_Matching.pdf [Accessed 27 November 2018]
(Used to assist in my understanding of the NNS SSD Algorithm)