Jeremiah Spears

CSCE 2100

12/5/2018

Project 3 Design Document

Design Process:

As I begun the design process for my program, I started by carefully reading through the project document making sure I knew exactly what would be required. As I read it, I realized that I could use some of the same elements as I had used in project 2. I knew that I would be able to store the image in the same way that I stored the grid of agents in the cellular automata program. I then decided to break down my program into the required functions that I would need before I actually coded them. This made it so that I would be able to keep track of what functionality I still needed to implement in my code.

The main steps I created to design my program (in order):

- Reading the input file
- Creating a struct to hold the pixel data
- Creating a 2d vector to store pixels
- The first pass of finding objects
- The second pass of finding objects
- Printing a properly formatted grid to show each object
- Counting the number of objects in the grid

When I first started out on the coding process, I underestimated the difficulty of the program. The longer I worked on it the more time I realized it would take. Thankfully I started the project with plenty of time to spare. In the past I have not given myself enough time to work on my projects so for project 3 I made sure to start early. This was a tremendous help in the design of my program as it gave me plenty of time to think out a solution to each problem I encountered.

Data Structures:

For my program implementation I used a struct to hold the data for each pixel. I went with a structure because I knew it would be the simplest way to keep track of each pixel, what object it belonged to, and it's relationship with other objects. The struct is called nPixel and it holds two integers: background and status. The background integer is used to keep track of the initial state of the pixels and is changed in the first pass of the image processing. The status integer is used to transfer the data during the second pass from the background integer. This allowed me to just use one 2d vector instead of copying the data to a temporary vector to perform the image processing.

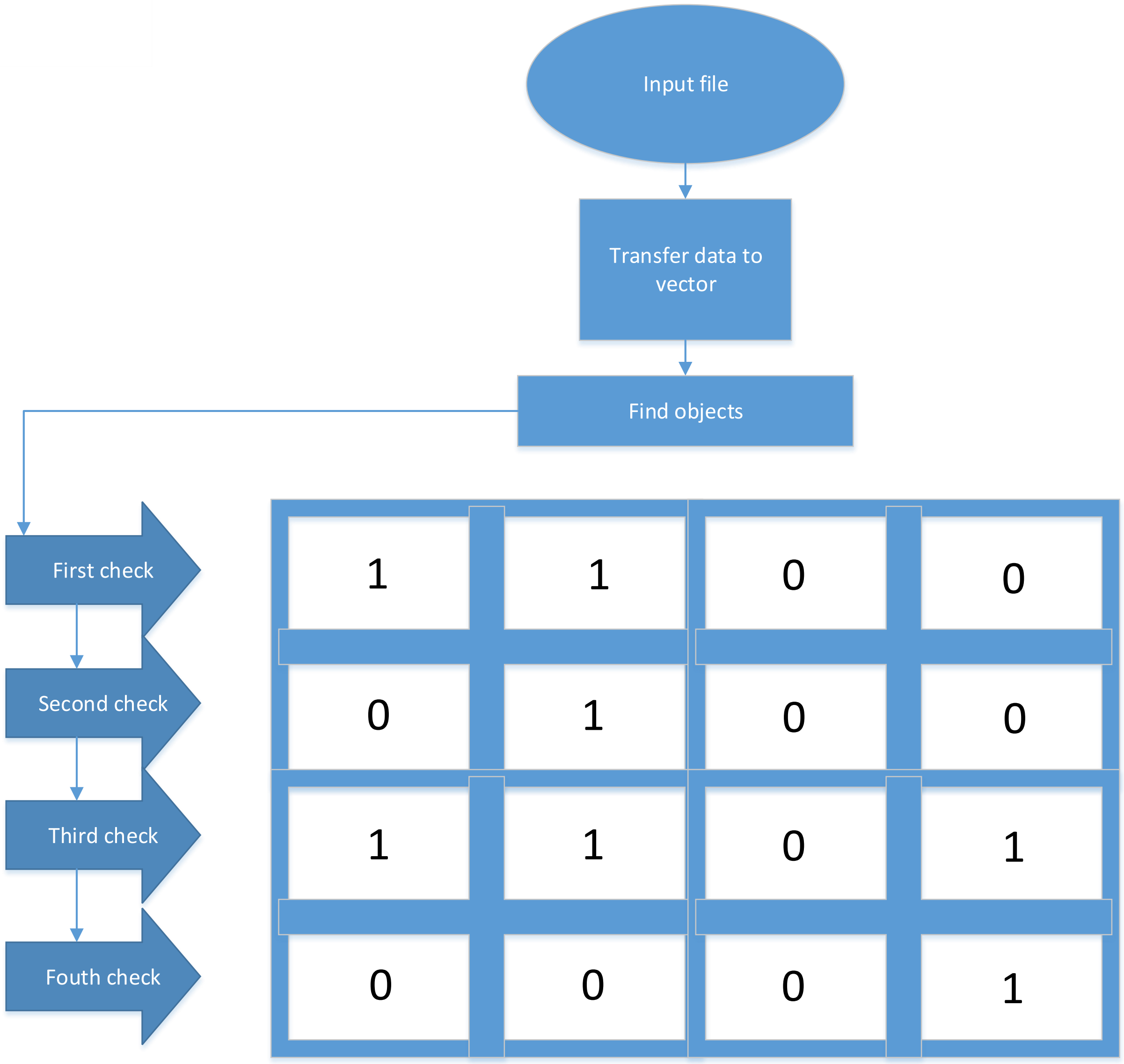
The program does the majority of the storage using one 2d vector. Using a 2d vector instead of a 2d array allows the program to easily work with any size input file. The 2d vector I used is a 2d vector of struct nPixels. This allows the program to keep track of the data for each pixel while also keeping it in the correct grid format.

The final part of the program uses a set to ultimately count each pixel status that has been stored in the grid. I decided to utilize a standard set because I could efficiently store each value in it, but could grid rid of any duplicate values easily and without error. This allowed me to simply output the size of the set to count the number of objects in the image.

Functionality:

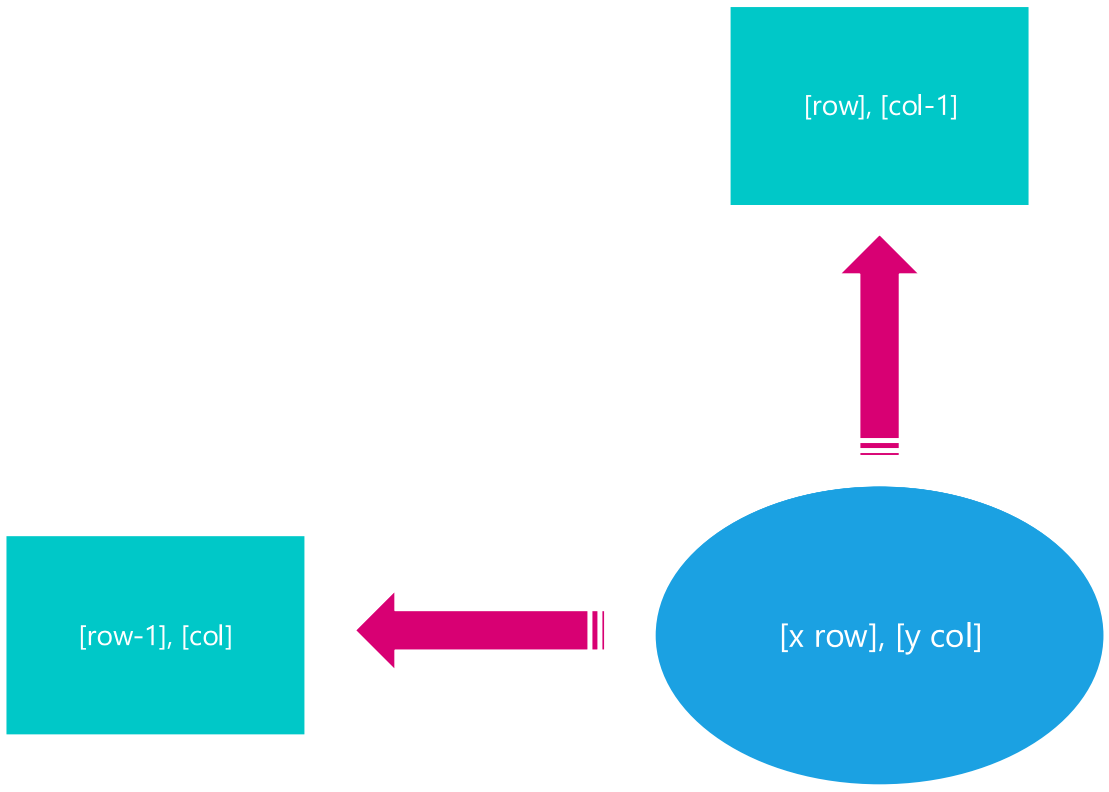
The program can take any user selected input file (with the correct formatting) and then store the data in a vector. The program will also get the size of the grid from the input file. Once the data is stored in a temporary vector of structs (nPixels) the 2d vector of nPixels will then be resized to the correct dimensions for the grid. After that the program will populate the 2d vector with the data from the temporary vector.

Once all the data is correctly stored in the 2d vector the program will then start the first pass of the image processing. The objects are found in the image by searching to see if any given pixel has an adjacent pixel either above it or to the left. This is accomplished using the calc() function in my program. This function uses two for loops to search through the entire image. It will first search through the first row and will go down a column once it reaches the end of the row. It will continue this process until it has reached the final row and column in the grid.



The program will only check the pixels for their adjacent neighbors if they are not in the top right corner as the pixel in the top right of the grid can not have a top or left pixel adjacent to it. The last procedure for the first pass is for the program to set the status of whether or not any objects have a relationship with each other. If the objects are placed adjacent to each other the program will assign them the same status which can then be used in the second pass.

How the calc function finds the adjacent pixels to form each object within the grid



In the second pass the program will go through again and search for any objects that are adjacent to each other. The process is executed in the next function called relationship(). This function uses a modified version of the algorithm from the first pass. Instead of finding the objects using their background values it finds the objects using their status value. If a pixel has the same status value as one next to it, but a different background value, the background value will be converted to the lowest of the two. The function will finish by converting any matching pixels from the converted pixel's old object to match with the new fully complete object.

Once both passes are complete the program will output the grid to the user and will use the count() function to count the number of unique objects in the image. It will then output the number of objects to the user which will end the program.

Group work share

I worked by myself on the full assignment and created both the code and design document for the project.