Bobby Kim  
Nick tindle  
CSCE 2100 – Project 3

# Picture Parsing - Design Document

Design Process

The design process began with short burst of trial and error functions to familiarize myself with certain aspects of this project. This was simple tests of populating, editing, and displaying the data inside a vector. This resulted in the first version of our project, and later became the code/program that we submitted for milestone one.

The milestone grade showed the unfamiliarity with vectors and returning data apparently. Nick explained that the two-vector approach was correct, but for some reason the data was not being returned properly. It was decided to use the same idea from the milestone submission, but to further branch from it by implementing the input being read into a grid instead of line by line. This fixed many issues we were having with the original code. It was also discovered that integer vectors were more manageable during parsing.

Data Structures

This program involves the usage of vectors as “grids” (pretty much just a formatted list of elements), and object-oriented programming.

There is an extensive use of for loops throughout the entire program. These loops are for parsing of the input, the population of the initial grid, and for populating and creating a grid of objects. If statements are used to check current data sets inside each vector. The program does not use any binary trees, graphs, BFS or DFS because we did not feel that it was applicable to the program.

Functionality

This program will begin by reading a provided input file to determine the initial count of everything. If an input file is not provided, the program will exit and throw an error describing the issue. The file input is read while the program is already running, instead of as a command line argument. Using the command line arguments may have caused some memory issues that we encountered in the milestone.

The program consists of several functions. The initial function is to read the input provided by the user into a “grid” vector. The second function then converts the initial input grid from a string/char vector into an int vector. We found it easier to parse integers instead of strings. The third function will then count the number of elements and label the objects accordingly. The fourth function will return the grid, while the fifth and six functions simply return the count of the number of objects inside the last object vector.

The first function reads in the input file using the ‘getline’ command and will remove any comma delimiters from input so it's easier for us to work with the data later. In the second function, the input elements are converted using ‘stoi’ [tempGrid.push\_back(stoi(val))] before being pushed into an int vector. This int vector is later parsed and separated into objects depending on the rules provided in the assignment. The conversion of objects checks the neighbors of the current element but with certain constraints. If the element is the first element in the vector, we do not check for a left neighbor, because it does not exist and will give us a segmentation fault. If any elements are on the first line, they will not have neighbors above them, so we do not check for them. The above neighbors are checked by finding the difference between the current element and its place in the vector. An example of this would be an element at location [2][5]. The above neighbor would be located at [1][5]. So we check for the location at [i-1][j] and compare the two. The same logic is applied when searching for neighbors to the left of the current element. We also kept track of each line that was currently being parsed. This was to prevent the accidentally comparison of a “left neighbor” that was actually just on a different line.

The grid is displayed by using a for loop assigned to the length of the vector. The data will be continually returned until the length of the vector has been reached. We also count the number of objects touching each other by applying the same “neighbor search” logic to the “checkGridForConnectingObjects” function. The search runs the entire length of the vector, and returns the results by incrementing the count of objects. We also searched the same object vector for the largest object number and saved it as a variable to be returned later.

Share of Work

This project (the design, coding, and debugging) was completed by Nick Tindle and Bobby Kim. I mainly worked with the original design for the milestone, and then Nick blew the original build to this amazing program now. The design report was typed by me, and Nick proofread it for any mistakes and edits.