Centralized Emergency Response

AiS 2002-2003 Final Report

Team Number 012
Members:
Jim Adolf
Josh Langsfeld
Matt Strange
Philip Coleman

Team Sponsor:
Jim Mims

Table of Contents

Executive Summary

Emergency Response systems, such as fire departments, have reaped the benefits of modern disaster-fighting technology, but have yet to employ efficient traffic-controlling systems to maximize their speed. Our project designed a new and original program that takes a city and a fire and then builds a complete fire-fighting response mechanism in the most efficient manner possible. In addition, we built a simulator for the city, providing a realistic means of modeling the chaos that occurs during rescue operations and providing a difficult problem that our program would have to defeat. The simulator, which controls a large body of people and cars, reports all data back to the distribution and direction algorithm and gives dynamic traffic patterns for our city. The city itself is a collection of nodes and paths, a system that gave us the dynamic city-building capabilities we would need to model real cities, such as Rome. The final part of our program was the graphical display system that read in the results of a test and gave a real-time simulation to model the city and our response.

Testing our program against the 'grid system' that current fire stations employ, we proved that our program is a useful aid that works well in both distributing fire-fighting equipment and directing operations in an intelligent manner.

Problem Statement

Centralized administrative systems almost always increase the speed and efficiency of an organization. Intuitively, it makes sense to have the "big boss" in the middle running everything, from a viewpoint of the big picture. This method also ensures a systematic breakdown of any task that would have to be completed, etc. By delivering all the information collected at the low-levels of the system to the high levels, a plan can be formed, assessed, broken up into manageable chunks, and sent back down to the lower levels for execution. This view of the "big picture" is imperative when trying to minimize costs or action for that system. For example, the United States Army uses this system to construct battle plans and tactics against whomever it may be fighting. If the lowly lieutenants and captains were allowed to do what they could on their own, there would be mass chaos, and the army would be defeated. But when the generals get the appropriate information, they can assess it and send it back as orders to lower personnel, who can then participate in a joint effort with others to complete an operation.

For our project this year, we decided to apply this centralized system to emergency response systems. Our goal was to simulate a city and then destroy it, through virtual fires. We would also create a fire department for that city, whose job would be to respond to the fires in the best possible manner. There were several problems that had to be addressed. If there is a lack of fire trucks, which fires should be put out first? How can we find the shortest route to a fire that tries to minimize contact with civilian people and cars? What is the optimum

number of fire trucks we should send to a fire to maintain a balance between putting out the fire in the shortest possible time, and conserving costs on resources? What is the best way to simulate a full, working city?

Finally, another large part of our project, unrelated to the idea of centralized emergency response, was the graphics. We needed a way to display the results of the simulation in a way that would be easy to understand and could give us all the information we needed simply by watching it. This was the challenge laid out for us at the beginning of the year.

Method

Our program hinges around an effective way to distribute and direct fire trucks. Each task uses separate algorithms that interact with each other to create a dynamically changing view of the city and the fastest possible response to all of the fires.

Our distribution algorithm is a type of greedy algorithm. It finds the fastest possible path from any fire truck to any fire at a given time, and causes that truck to go to that fire. It then repeats the process, omitting all fires and fire trucks already assigned. This creates what we feel is the most efficient way of distributing fire fighting material throughout the city, because we feel that rather than creating paths roughly equal to each other to all fires, we ensure that as many fires are put out as fast as possible, allowing fire trucks which have completed their tasks to help other fire trucks with their fires, up to a limit, which we felt was wise, because after a certain number of fire trucks, any more would just cause congestion in the disaster area and decrease efficiency, rather than increasing it.

Our direction algorithm was originally a genetic algorithm, which worked by taking information from randomly created paths to create better paths, and then, using these paths as a base, repeating the process. We had some success with this originally, but we encountered two problems with it in the later stages of our project. We found that it was much more difficult to create valid random paths and ensure validity of subsequent generations' paths from one point to another in a node based city than in a square-gridded city layout. Also,

the genetic algorithm was not fast enough for our purposes, and did not ensure the fastest path from one point to another. The genetic algorithm did ensure a relatively fast path, compared to a randomly generated path, but it was not necessarily the absolute fastest. Also, when we were using the genetic algorithm in the distribution algorithm, it did not always come up with the same path from one point to another on multiple trials, so calculating the time taken to get from one point to another was inconsistent, causing the greedy algorithm to not always end with the absolute correct data.

Our solution was to use Djikstra's algorithm, a method for finding the least cost path in a network. The algorithm was mentioned in passing by Jim Adolf's brother as a possible solution to our problem. He was able to explain the basics of the algorithm, but before he was able to provide any more help, he had to go back to college. We used the basic idea of Djikstra's Algorithm to create our own algorithm. We know it is possible to create the algorithm using O (n*log (n)), but at our present level of skill, we had to use an algorithm of O (n^2 log (n)) (This was because we could not create the sparse matrix required for the fastest possible operation). Also, the original Djikstra's Algorithm used complex matrix arithmetic that we could not fully understand. Instead, we used the basic idea and used quicksort() to simplify the algorithm. Djikstra's Algorithm uses the following steps: First, it creates a blank copy of the city, and adds on all data directly connected to the start node. Then, it finds the fastest possible path from the starting node to all nodes not already part of the newly created city. In the beginning, these are the starting node's neighbors. Then, it finds the fastest path

of any of these. Lastly, it adds the end node of this fastest path to the newly created city. It repeats steps two through four until all nodes have been added to the city. After the first iteration of the algorithm, the new city will contain the starting node and its closest neighbor. The algorithm guarantees that it will be able to find the fastest path from the starting node to all nodes in the city, because it only adds a path to the list after it ensures that no other paths could possibly be shorter than it. This new approach has many benefits. First, it always produces the same data given the same conditions, something that we had problems with in the genetic algorithm. Secondly, it produces all the paths simultaneously, which means determining the fastest path from a fire station to any number of fires takes the same number of steps, something that our genetic algorithm could not do. Lastly, and most importantly for the project, it could guarantee the fastest path from any point to another, which could, if our program was enacted, save lives.

In addition to our ideal distribution algorithm, we created a distribution algorithm that mimics the way fire trucks are distributed now. In this algorithm, each fire truck has a given territory that it is responsible for, and any and all fires in that district are the responsibility of that fire station. We feel that our distribution algorithm is superior, because, especially in times of unusual traffic, the fires are responded to in the quickest manner. Also, it ensures that no fires are left unanswered due to lack of fire trucks at a single station. We admit that if the districts were drawn carefully, the responsible fire station would most likely be

the speediest to respond, but we feel that our method has all the benefits of the previous plan in addition to the added flexibility that is its strength.

Our model of the city uses stochastic modeling, in which the people and cars move randomly through the city. We felt that this was the best way to model their movement, because we felt that it was impossible to accurately predict where humans would go and how they would react to a given circumstance. Our city calculates the traffic along a given road by using the following equation:

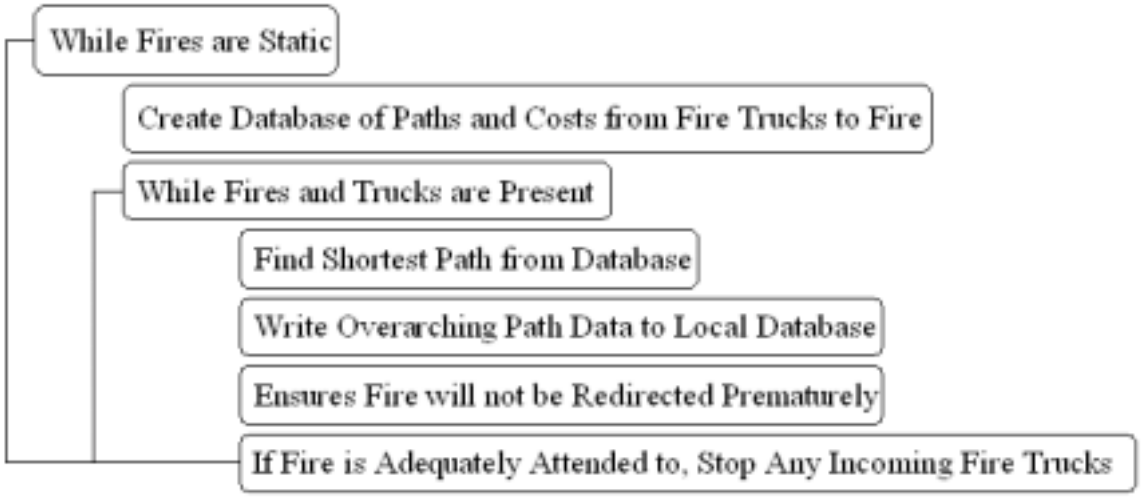
traffic=((speed limit)/5)*(#of people on road+5*#of cars on road+1) +1.

People move at a constant speed, so their speeds are proportional only to the traffic, whereas cars move proportionally fast to both the traffic and the speed limit.

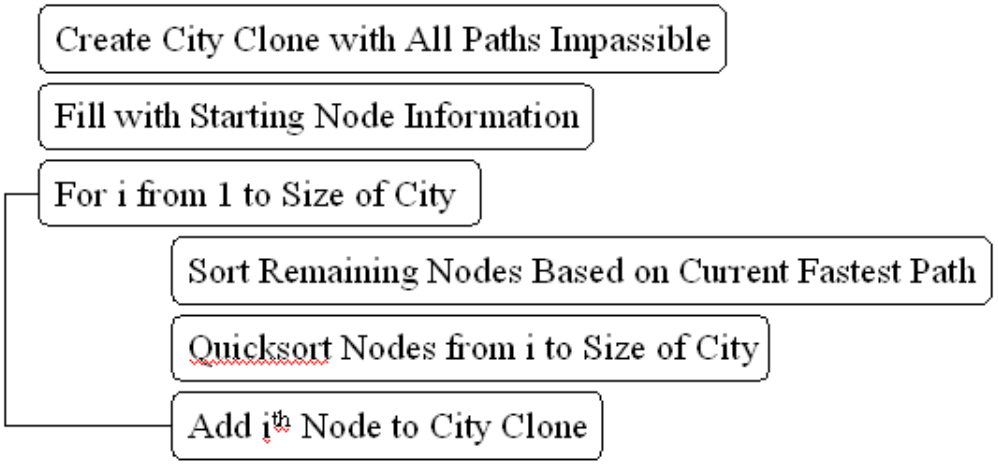
Lastly, our program interfaces with graphics, so that the efficacy of our program can be clearly seen visually. Our graphics are animations created by original code, which does use two .dll files, which can be found in the acknowledgements. The graphics interface with the rest of the program through the use of a .cer-formatted file. The distribution and directing algorithms write to the file using predetermined keywords, and the graphics use a parsing algorithm to read the file in, and output the information in an animated format. On of the advantages of having the graphics separate from the rest of the code is that each individual result of the program can be viewed an unlimited number of times, so that the run can be analyzed carefully, and the efficacy clearly seen.

Math Model

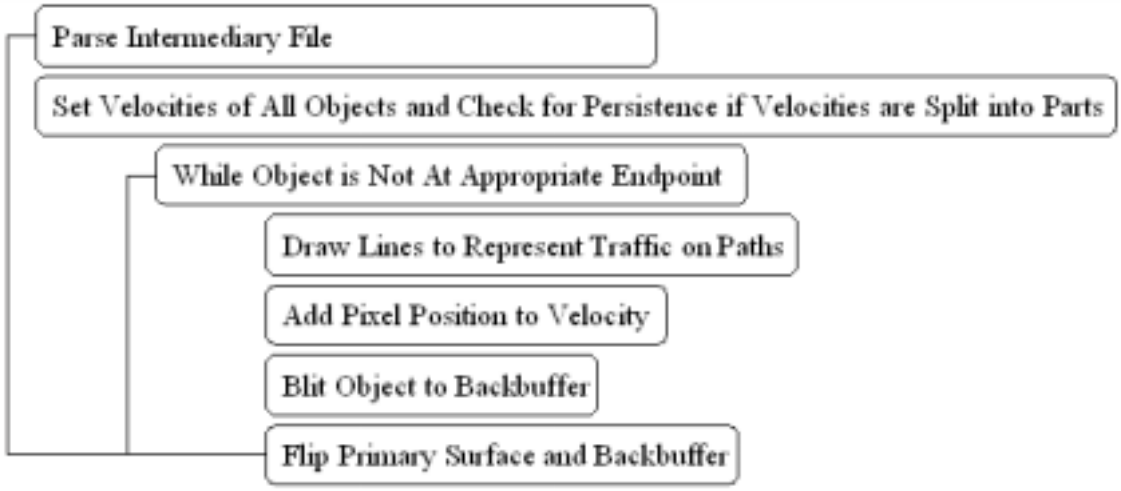
Distribution Algorithm



Djikstra's Algorithm



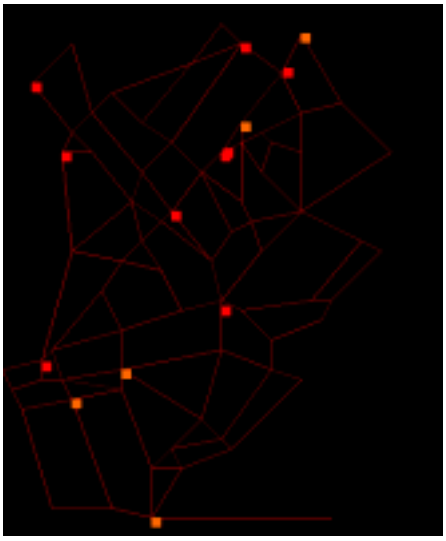
Graphics



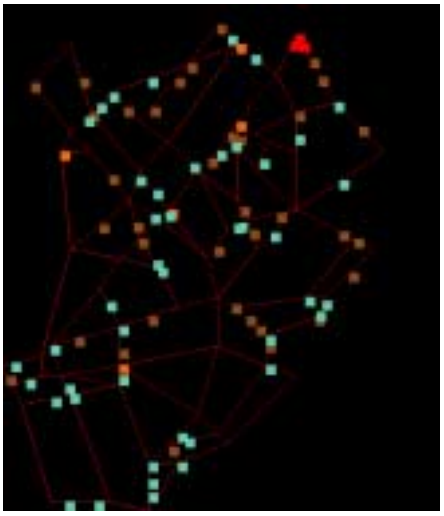
Results

We found that our algorithm dispersed the fire trucks accurately and speedily, and it did follow the fastest path. Our distribution algorithm was often very comparable to the present system, if we drew the districts wisely, and only was significantly better when unusually large concentrations of civilians congregated in a path. Of course, if we drew the district boundaries poorly, our algorithm was far superior to the present system. We also ran a single test using our distribution algorithm, but comparing the Djikstra Algorithm to a path that we picked out by hand, using only the raw city map as reference. The Djikstra algorithm clearly out performed it, because the civilians in the way of the hand picked path slowed the fire truck down significantly more than those in the way of the path picked by the Djikstra Algorithm.

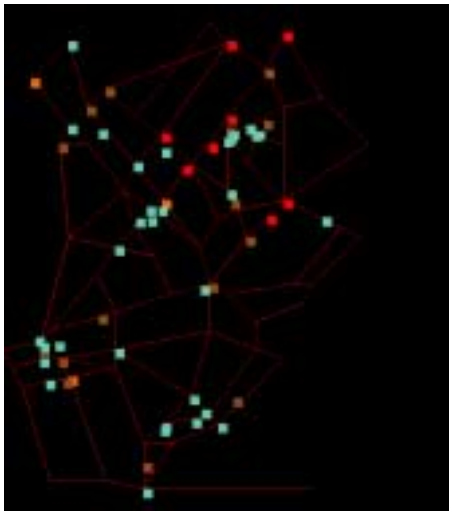
Here are some screen shots of our graphics:



The fire trucks proceed to extinguish all the fires in our first fire simulation. These trucks have begun at the top-leftmost point and spread downward, using the low-traffic path and reaching the rest of the city's fires. The red blocks are fire trucks, and the orange blocks are fires.



Our more advanced simulation lets the fire trucks start in a new location. Note the different colors of lines to indicate the traffic on the path. A brighter red indicates a more heavily trafficked path, as far as the AI is concerned. The brown blocks are people, and the turquoise blocks are cars.



This further point in the simulation shows the fire trucks accessing median routes to avoid people and cars from the city. While still moving towards the fires, the AI has recalculated the paths to keep traffic low at all times. This process continues until all fires are successfully extinguished (note some remaining fires in the middle, top and bottom left of the map).

Conclusions

We concluded from the testing of our simulator that our project offers a truly superior method of directing emergency response vehicles by considering the complete traffic patterns of the city. We believe that our implementation of Djikstra's Algorithm provided an efficient manner of directing fire truck traffic, and that the city simulations built onto it added a new degree of fidelity and accuracy. We concluded that the current 'grid-based' system used by fire departments is inferior since it:

1. Calculates based on the position of the fire, and does not take city traffic patterns into account,

2. Does not coordinate the combined power of its vast fleet of fire trucks effectively, and

3. Provides no real-time data as the fire moves and spreads.

Bearing these facts in mind, our program is not especially useful in an area where it cannot receive live data feeds, or in a city that is so small that the road system would not require an advanced algorithm to find paths. Our program is also limited to sphere of fire response for now. We intended to coordinate all disaster efforts, but had to limit ourselves to fires in the end. Our failure with the Genetic Algorithm also proved that our algorithms had to be streamlined and simplified before they would prove as useful as we had hoped.

Our project provides optimized implementations of all the above qualities and proved that it would successfully outpace a grid system in testing. Using

calculated traffic equations and a intelligent ‘response level’ approach, the fire trucks acted in a manner that, from our simulations, appears to be quite intelligent and methodical. In summation, we concluded that a Djikstra AI implantation would actually provide improved fire response capabilities over the system that is currently used today, both in theory and in practice.

Recommendations

We feel that right now the only weak link in our code is the fact that the general population of the city is not governed by equations. We are still trying to find statistics on the area of Rome that we chose as our basis. As soon as we find this information, we intend to add it in.

Original Achievement

We had two achievements that we were very proud of in our project. The first was the implementation of Djikstra's algorithm into our simulation of the fire and the city. After a very long period of coding and debugging, it was very satisfying to watch all the fire trucks spread out and neutralize all the fires in the city.

The second achievement was our graphics. This part of the project took a tremendous amount of effort and was definitely well worth it. The graphics turned out to be a big success, showing everything we expected to see, and more, such as small pieces of info such as the current command being executed or the step number the simulation was on.

Both of these creations took a long time to produce, and we were all very satisfied with the results that we obtained.

Acknowledgements

The AI used the Djikstra algorithm. Robert Adolf (Jim Adolf's brother) provided an explanation of the algorithm, but all code for the distribution and direction of fire trucks is originally written by Jim Adolf. The people and car movement algorithms were originally written by Josh Langsfeld.

The Graphics utilized two open-source packages: the Simple DirectMedia Layer (SDL) 1.2.4 by Sam Lantinga, which allowed the graphics to have buffering and blitting capabilities and the SDL_Draw Primitives Drawing Library by Mario Palomo, José de la Huerga and Pepe González, which allowed the drawing of

lines and circles. All parsing, interpretation, and drawing code is originally written by Matt Strange.

Code

Cermain.h

//This is the main header file for the entire program. It contains all the data of all of our classes not used in the graphics.

```
#ifndef CERMAIN_H
#define CERMAIN_H


#include <iostream.h>
#include "apvector.h"
#include <fstream.h>
#include "apqueue.h"
#include "apmatrix.h"


#define FIRESTATIONS 1
#define TPS 10
#define TPF 1
#define SIZE 73

struct pair {
     int start, end;
};

struct node_path
{
     int id;
     double cost;
     int pos;
     int path[SIZE];
};

typedef double city_t[SIZE][SIZE];
```

//This is the connection matrix of the city. If the member is 10000, it is considered impassable. Otherwise, smaller numbers mean faster speeds between nodes. Every single connection and possible connection is mapped on this matrix.

```
const double city[SIZE][SIZE] = {
{
  0, 10000, 10000, 10000, 10000,   8, 10000, 10000,   5,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000,   0, 10000,   6,   6, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000,   0, 10000, 10000, 10000,   5, 10000, 10000,
10000, 10000,   7, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000,   6, 10000,   0,   6, 10000,   6, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   1, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000,   6, 10000,   6,   0, 10000, 10000,   5, 10000,   8,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
  8, 10000, 10000, 10000, 10000,   0, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   5, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000,   5,   6, 10000, 10000,   0, 10000, 10000,
10000,   3, 10000,   9, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000,   5, 10000, 10000,   0,   7,
5, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
  5, 10000, 10000, 10000, 10000, 10000, 10000,   7,   0,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   9, 10000,   3,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000,   8, 10000, 10000,   5, 10000,
0, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,  10, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000,   3, 10000, 10000,
10000,   0,   5, 10000, 10000, 10000,   9, 10000,   6,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
```

```
10000, 10000,   7, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   5,   0, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   4, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000,   9, 10000, 10000,
10000, 10000, 10000,   0,   6,   9, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   6,   0,  10, 10000, 10000,
10000, 10000, 10000,   6, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   9,  10,   0,   8, 10000,
10000,   6,   3, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   9, 10000, 10000, 10000,   8,   0,  11,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,  11,   0,
8,   7, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   6, 10000, 10000, 10000, 10000, 10000,   8,
0, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   7, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   6, 10000,   7,
10000,   0, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   7, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   3, 10000, 10000,
10000, 10000,   0,   3, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   7, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   6, 10000, 10000, 10000,
10000, 10000,   3,   0,   8, 10000, 10000, 10000, 10000,
10000,   3, 10000, 10000, 10000,   6, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000,   1, 10000, 10000, 10000, 10000, 10000,
10, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   8,   0, 10000, 10000, 10000,   7, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   0,   7,   7,
10000,   4, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000,   5, 10000, 10000,   9,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   7,   0,   5, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   7,   5,   0,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   1, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   3,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   7, 10000, 10000, 10000,   0,
11,   5, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   4, 10000, 10000,
11,   0, 10000, 10000, 10000, 10000, 10000,  10,   6,   4,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   3, 10000, 10000, 10000, 10000,
5, 10000,   0, 10000, 10000, 10000, 10000,   9, 10000,
10000, 10000, 10000,   3, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   4, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   0,   4, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   7,   7, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   4,   0,   7, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   6,   8, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   7, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   7,   0,   6, 10000, 10000,
10000, 10000, 10000, 10000,   6, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   6, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   6,   0, 10000, 10000,
10000, 10000, 10000,   6, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,  10,   9, 10000, 10000, 10000, 10000,   0,
```

```
6, 10000, 10000, 10000,   4, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   6, 10000, 10000, 10000, 10000, 10000,   6,
0, 10000,   6,   8, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   1,
10000,   4, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
0,   3, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   1,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   6,   3,   0,   3, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   5,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   8, 10000,   3,   0, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   2, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   3, 10000, 10000, 10000,   6,   4,
10000, 10000, 10000, 10000,   0, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,  10, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   6,   6, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   0, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   1, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   8, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   0,   8,   1,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   8,   0,
10000,   5, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   1, 10000,
0,   8, 10000, 10000,   7, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   5,
8,   0,  10, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,  10,    0,   3, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   3,   0,   7, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   8, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   7, 10000, 10000,   7,   0,   9, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,  10,   1, 10000, 10000,
10000, 10000, 10000, 10000,   9,   0,   9, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   4, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   2, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   9,   0,   4, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   4,   0,
7, 10000, 10000,   7, 10000, 10000, 10000,   7, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   5, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   7,   0,
10000, 10000,   4, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   0,   8, 10000,   7, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   1, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,   8,   0,   9, 10000,   6, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   7,
4, 10000,   9,   0, 10000, 10000,  10, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   7, 10000, 10000,   0,   4, 10000, 10000,
```

```
10000, 10000, 10000,   6, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   6, 10000,   4,   0,   7, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,  10, 10000,   7,   0,   6,
10000, 10000, 10000, 10000,   4,  11, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   7,
10000, 10000, 10000, 10000, 10000, 10000,   6,   0,   4,
10000, 10000, 10000, 10000,   6,   5, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   4, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   4,   0,
4, 10000, 10000, 10000, 10000,   6, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   8, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   4,
0,   9, 10000, 10000, 10000, 10000,   6, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   9,   0, 10000, 10000, 10000, 10000, 10000,
10000,   3, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   6, 10000, 10000, 10000,
10000, 10000, 10000,   0,   6, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   2, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   4, 10000,
10000, 10000, 10000,   6,   0,   4, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   2, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,  11,   6,
10000, 10000, 10000, 10000,   4,   0, 10000, 10000, 10000,
10000,   4, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,   5,
6, 10000, 10000, 10000, 10000, 10000,   0,   9,   7, 10000,
10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000,   6, 10000, 10000, 10000, 10000,   9,   0,
7,   6, 10000, 10000, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   7,   7,
0, 10000,  10,  11, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   3, 10000, 10000, 10000, 10000,   6,
10000,   0, 10000,   7, 10000, 10000, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000,   4, 10000, 10000,
10, 10000,   0,   3, 10000, 10000,   7, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000,  11,   7,   3,   0, 10000, 10000,   4, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
```

```
10000, 10000, 10000, 10000,   2, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   0,   8, 10000, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000,   2, 10000, 10000, 10000,
10000, 10000, 10000, 10000,   8,   0,   3, 10000},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000,   7,   4, 10000,   3,   0,   2},
{
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000,   2,   0}
};
```

//This is a matrix that maps nodes numbers to x, y, coordinates.

```
const int map[SIZE][2]={
{ 72, 79 },
{ 224, 54 },
{ 309, 69 },
{ 248, 78 },
{ 192, 96 },
{ 34, 114 },
{ 281, 105 },
{ 115, 125 },
{ 92, 140 },
{ 146, 152 },
{ 306, 149 },
{ 342, 134 },
{ 247, 155 },
{ 221, 184 },
{ 247, 174 },
{ 267, 166 },
```

```
{ 279, 176 },
{ 301, 189 },
{ 263, 201 },
{ 243, 230 },
{ 207, 202 },
{ 175, 175 },
{ 93, 183 },
{ 74, 161 },
{ 60, 189 },
{ 149, 208 },
{ 133, 233 },
{ 177, 240 },
{ 397, 182 },
{ 301, 245 },
{ 251, 257 },
{ 230, 263 },
{ 161, 259 },
{ 149, 274 },
{ 77, 280 },
{ 125, 298 },
{ 162, 308 },
{ 218, 292 },
{ 265, 289 },
{ 364, 271 },
{ 380, 286 },
{ 313, 334 },
{ 338, 332 },
{ 327, 350 },
{ 279, 373 },
{ 247, 348 },
{ 224, 335 },
{ 180, 348 },
{ 121, 366 },
{ 103, 325 },
{ 9, 400 },
{ 41, 390 },
{ 57, 385 },
{ 17, 423 },
{ 49, 416 },
{ 66, 413 },
{ 123, 403 },
{ 224, 388 },
{ 278, 404 },
{ 300, 411 },
{ 23, 446 },
{ 73, 435 },
{ 120, 424 },
{ 205, 458 },
{ 223, 475 },
{ 179, 487 },
{ 239, 488 },
{ 156, 503 },
{ 183, 505 },
{ 56, 543 },
```

```
{ 111, 545 },
{ 158, 559 },
{ 333, 552 }};
```

//This class takes care of the direction of fire trucks through the city

```
class AI
{
public:

     AI();

     apvector<node_path> find_routes(int start);
     apvector<node_path> output(){return nodes;}
private:
     void Dijkstra( int start );
     void QuickSort(int first, int last);
     int partition(int left, int right);
     void QuickSort2(int first, int last);
     int partition2(int left, int right);
     void swap(node_path & a, node_path & b);
     apvector<node_path> nodes;
};
```

//This class takes care of the distribution of fire trucks through the city

```
class finder
{
public:
     apmatrix<node_path> distribute_trucks();
private:
     void findfire(apqueue<int> & firequeue);
     void find_min(node_path paths [FIRESTATIONS][SIZE],
pair & current_target);
};
```

//This class takes care of all functions relating to the people in the city.

```
class person
{
public:
     person();
     void Move(ofstream &, apmatrix <int> &, apmatrix <int>
&);
     void setid(int ID) {id = ID;}
     int getid() {return id;}
     int loc() {return itsLoc;}
     void setloc(int loc) {itsLoc = loc;}
private:
     int dest;
     int itsLoc;
```

```
     int id;
     bool onMove;
     int stepstogo;
};
```

//This class takes care of all functions relating to the cars in the city.

```
class vehicle
{
public:
     vehicle();
     void Move(ofstream &, apmatrix <int> &, apmatrix <int>
&);
     void setid(int ID) {id = ID;}
     int getid() {return id;}
     int loc() {return itsLoc;}
     void setloc(int loc) {itsLoc = loc;}
private:
     int dest;
     int id;
     int itsLoc;
     bool onMove;
     int stepstogo;
};

#endif
```

driver.cpp

//This file contains the int main() function and is responsible for calling all of the other functions and classes.

```
#include "cermain.h"
void setpaths(ofstream & outfile);

int fires[10]={2,12,5,27,13,24,3,56,61,71};

int main()
{
     int i, j, k, l, r, p, c, s;
     ofstream outfile;
     char file[50];
     apmatrix <int> ptraffic(SIZE, SIZE, 0);
     apmatrix <int> ctraffic(SIZE, SIZE, 0);
     //pt = &ptraffic;
     //ct = &ctraffic;
     cout << "Enter number of 10-pixel blocks: ";
     cin >> r;
     cout << "Enter number of people: ";
     cin >> p;
     cout << "Enter number of cars: ";
```

```
     cin >> c;
     cout << "Enter number of steps: ";
     cin >> s;
     cout << "Enter .cer file name: ";
     cin >> file;

     apvector <person> population(p);
     apvector <vehicle> cars(c);

     outfile.open(file);
     outfile << "worldsize " << r << " " << r << endl;
     outfile << "arrayperson " << p << endl;
     outfile << "arraycar " << c << endl;
     outfile << "arrayfiretruck " <<FIRESTATIONS*TPS <<endl;
     outfile << "arrayfire 10" <<endl;

     for ( i=0; i<FIRESTATIONS*TPS;i++)  //Intialize objects
     {
          outfile <<"setfiretruck " << i << " " <<
(map[0][0])/10 << " " << (map[0][1])/10 <<endl;
     }
     for (i=0; i < p; i++)
     {
          j = rand() % SIZE;
          population[i].setid(i);
          population[i].setloc(j);
          outfile << "setperson " << i << " " << map[j][0]
<< " " << map[j][1] << endl;
     }
     for (i=0; i < c; i++)
     {
          j = rand() % SIZE;
          cars[i].setid(i);
          cars[i].setloc(j);
          outfile << "setcar " << i << " " << map[j][0] << "
" << map[j][1] << endl;
     }


     outfile <<"STEP" <<endl;
     for (i=0; i<10; i++)  //Initialize fires
          outfile <<"setfire " << i << " " <<
(map[fires[i]][0])/10 << " " << (map[fires[i]][1])/10 <<
endl;
     outfile <<"STEP" <<endl;

     AI hal;
     finder spazzo;
     hal.find_routes(0);
     apvector<node_path> nodes(hal.output());
     apmatrix<node_path> targets
(spazzo.distribute_trucks());
     l=0;
     int dest[FIRESTATIONS*TPS];
```

```
     int check[FIRESTATIONS*TPS];
     int put_out[FIRESTATIONS*TPS];
     for ( i=0; i<targets.numrows(); i++)
          for ( j=0; j<targets.numcols(); j++) {
               dest[l]=targets[i][j].id;
               l++;
          }
          for (i=0; i<FIRESTATIONS*TPS; i++) {
               check[i]=1;
               put_out[i]=1;
          }
     l=0;
     j=0;
     while (l==0){
          for (i=0; i<FIRESTATIONS*TPS; i++) {
               if(j<nodes[dest[i]].pos){
                    outfile <<"movefiretruck " << i << " "
<< (map[nodes[dest[i]].path[j]][0])/10 << " " <<
(map[nodes[dest[i]].path[j]][1])/10 << " 1"<<endl;
               }
               else {
                    if (put_out[i]==1)
                         for (k=0; k<10; k++)
                              if (dest[i]==fires[k])
                                   outfile <<"killfire " <<
k <<endl;
                    check[i]=0;
               }
          }
          l=1;
          for (i=0; i<FIRESTATIONS*TPS; i++)
               if (check[i] != 0)
                    l=0;
          j++;
          if (l==0){
               setpaths(outfile);
               outfile <<"STEP" <<endl;
          }
     }

     return 0;
}
void setpaths(ofstream & outfile)
{
     int i, j;
     for (i=0; i<SIZE; i++)
          for (j=0; j<SIZE; j++)
               if (city[i][j] != 10000 && city[i][j]!=0)
                    outfile <<"setpath " << (map[i][0])/10
<< " " << (map[i][1])/10 << " " << (map[j][0])/10 << " " <<
(map[j][1])/10 << " " <<100-(city[i][j]*9) <<endl;
}
```

Djikstra.cpp

//This file is responsible for directing the firetrucks along their paths, and also providing the distribution algorithm with the path cost data.

```
#include "cermain.h"

AI::AI()
{
     nodes.resize(SIZE);
}
apvector<node_path> AI::find_routes(int start)
{
     int i, j;
     for (i=0; i<SIZE; i++) {
          nodes[i].id=i;
          nodes[i].cost=10000;
          nodes[i].pos = 0;
          for( j=0; j<SIZE; j++ )
               nodes[i].path[j]=10000;
     }
     nodes[start].cost=0;
     Dijkstra( start );
     QuickSort2(0, SIZE-1);
     return nodes;
}

/*************************************************************
******************/

void AI::Dijkstra( int start )
{
     int i, j, k;
     QuickSort(0,SIZE-1);
     // make a new city
     city_t current_city;
     for ( i=0; i<SIZE; i++)
          for ( j=0; j<SIZE; j++)
               current_city[i][j]=10000;
     // fill in with self info
     for( i=0; i<SIZE; i++ )
          current_city[start][i] = city[start][i];
     for( i=0; i<SIZE; i++)
          if( current_city[start][nodes[i].id] < 10000) {
               nodes[i].cost = city[start][nodes[i].id];
               nodes[i].path[nodes[i].pos] = nodes[i].id;
               (nodes[i].pos)++;
          }
     QuickSort(0, SIZE-1);
     // Main Loop
     for(i=1; i<SIZE; i++) {
          // Sort the remaining nodes
          QuickSort(i, SIZE-1);
```

```
          // We know that the ith entry in nodes is the next
cheapest path
          // Now we update the costs of all of the nodes
adjacent to the ith entry
          for( j=0; j<SIZE; j++ )

     current_city[nodes[i].id][j]=city[nodes[i].id][j];
          for(j=i; j<SIZE; j++) {
               // if adjacent
               if( current_city[nodes[i].id][nodes[j].id] !=
10000) {
                    // Relax
                    if(
current_city[nodes[i].id][nodes[j].id]+nodes[i].cost <
nodes[j].cost) {
                         // the new path is cheaper
                         nodes[j].cost =
current_city[nodes[i].id][nodes[j].id] + nodes[i].cost;
                         for( k=0; k<SIZE; k++ )
                              nodes[j].path[k] =
nodes[i].path[k];
                         nodes[j].pos = nodes[i].pos;
                         nodes[j].path[nodes[j].pos] =
nodes[j].id;
                         (nodes[j].pos)++;
                    }
               }
          }
     }
}

void AI::QuickSort(int first, int last )
{
     int currentLocation;
     if ( first >= last )
          return;
     currentLocation = partition(first, last);
     QuickSort(first, currentLocation - 1);
     QuickSort(currentLocation + 1, last);
}

int AI::partition(int left, int right)
{
     int position = left;
     while ( true )
     {
          while ( nodes[position].cost <= nodes[right].cost
&& position != right )
               --right;
          if ( position == right )
               return  position;
          if ( nodes[position].cost > nodes[right].cost)
          {
               swap(nodes[position], nodes[right] );
```

```
                position =  right;
          }
          while ( nodes[left].cost <= nodes[position].cost
&& left != position )
                ++left;
          if ( position == left )
                return  position;
          if ( nodes[left].cost > nodes[position].cost )
          {
                swap(nodes[position], nodes[left] );
                position = left;
          }
     }
}


void AI::QuickSort2(int first, int last )
{
     int currentLocation;
     if ( first >= last )
          return;
     currentLocation = partition2(first, last);
     QuickSort2(first, currentLocation - 1);
     QuickSort2(currentLocation + 1, last);
}

int AI::partition2(int left, int right)
{
     int position = left;
     while ( true )
     {
          while ( nodes[position].id <= nodes[right].id &&
position != right )
                --right;
          if ( position == right )
                return  position;
          if ( nodes[position].id > nodes[right].id)
          {
                swap(nodes[position], nodes[right] );
                position =  right;
          }
          while ( nodes[left].id <= nodes[position].id &&
left != position )
                ++left;
          if ( position == left )
                return  position;
          if ( nodes[left].id > nodes[position].id )
          {
                swap(nodes[position], nodes[left] );
                position = left;
          }
     }
}
```

```
void AI::swap(node_path & a, node_path & b)
{
     int i;
     node_path temp;
     temp.id=a.id;
     temp.cost=a.cost;
     for( i=0; i<SIZE; i++ )
          temp.path[i]=a.path[i];
     temp.pos=a.pos;
     a.id=b.id;
     a.cost=b.cost;
     for( i=0; i<SIZE; i++ )
          a.path[i]=b.path[i];
     a.pos=b.pos;
     b.id=temp.id;
     b.cost=temp.cost;
     for( i=0; i<SIZE; i++ )
          b.path[i]=temp.path[i];
     b.pos=temp.pos;
}
```

Finder.cpp

//This file is responsible for distributing the fire trucks to the various fires throughout the city.

```
#include "cermain.h"

apmatrix<node_path> finder::distribute_trucks()
{
     int i, j, k, l, num_fires, num_trucks;
     AI hal;
     apvector<int> truck_dist (FIRESTATIONS);
     apmatrix<node_path> targets (FIRESTATIONS, TPS);
     for ( i=0; i<FIRESTATIONS; i++)
          truck_dist[i]=(TPS-1);
     apqueue<int> firequeue;
     int firestations[FIRESTATIONS] = {0};
     findfire(firequeue);
     node_path paths [FIRESTATIONS][SIZE];
     for ( i=0; i<FIRESTATIONS; i++)
          for ( j=0; j<SIZE; j++) {
               paths[i][j].id=10000;
               paths[i][j].cost=10000;
               for (k=0; k<SIZE; k++)
                    paths[i][j].path[k]=10000;
               paths[i][j].pos=10000;
          }
     apvector<int> fires(firequeue.length());
     for ( i=0; i<fires.length(); i++) {
          firequeue.dequeue(j);
```

```
          fires[i]=j;
     }
     for ( i=0; i<FIRESTATIONS; i++)
          for ( j=0; j<fires.length(); j++)
               paths[i][j].cost=10000;
     for ( i=0; i<FIRESTATIONS; i++) {
          apvector<node_path> fire_paths
(hal.find_routes(firestations[i]));
          for ( j=0; j<SIZE; j++)
               for ( k=0; k<fires.length(); k++) {
                    if( fires[k]==j ) {
                         paths[i][j].id=j;
                         double temp=fire_paths[j].cost;

     paths[i][j].cost=fire_paths[j].cost;
                         temp=paths[i][j].cost;
                         for( l=0; l<SIZE; l++ )

     paths[i][j].path[l]=fire_paths[j].path[l];
                         paths[i][j].pos=i;
                    }
               }
     }
     int fire_dist [SIZE];
     for (i=0; i<SIZE;i++)
          fire_dist[i]=-1;
     for ( i=0; i<SIZE; i++)
          for (j=0; j<fires.length(); j++)
               if ( fires[j]==i)
                    fire_dist[i]=(TPF-1);

     num_trucks=FIRESTATIONS*TPS;
     num_fires=fires.length()*TPF;
     while (num_trucks > 0) {
          if (num_fires==0)
               num_trucks=0;
          while (num_fires > 0) {
               if(num_trucks > 0) {
                    pair current_target;
                    find_min(paths, current_target);
                    if (truck_dist[current_target.start] > -
1) {
                         if (fire_dist[current_target.end] >
-1) {

     targets[current_target.start][truck_dist[current_target
.start]].id=current_target.end;


     targets[current_target.start][truck_dist[current_target
.start]].cost=paths[current_target.start][current_target.end
].cost;
                              for( i=0; i<SIZE; i++ )
```

```
     targets[current_target.start][truck_dist[current_target
.start]].path[i]=paths[current_target.start][current_target.
end].path[i];

     truck_dist[current_target.start]--;
                              fire_dist[current_target.end]-
-;
                              num_fires=0;
                              for ( i=0; i<SIZE; i++)

     num_fires=num_fires+fire_dist[i]+1;
                              num_trucks=0;
                              for ( i=0;
i<truck_dist.length(); i++)

     num_trucks=num_trucks+truck_dist[i]+1;
                         }
                         else
                              for ( i=0; i<FIRESTATIONS;
i++)

     paths[i][current_target.end].cost=10000;
                    }
                    else
                         for ( i=0; i<SIZE; i++)

     paths[current_target.start][i].cost=10000;
               }
               else
                    num_fires=0;
          }
     }
     return targets;
}


void finder::findfire(apqueue<int> & firequeue)
{
     int i;
     for (i=0; i<SIZE; i++)
//        if (tile.isBurning)
          if (i==2 || i==12 || i==5 || i==27 || i==13 ||
i==24 || i==3 || i==56 || i==61 || i==71)
               firequeue.enqueue(i);
}
void finder::find_min(node_path paths [FIRESTATIONS][SIZE],
pair & current_target)
{
     int i, j, k;
     node_path current_max;
     current_max.cost=10000;
     for ( i=0; i<FIRESTATIONS; i++)
          for ( j=0; j<SIZE; j++){
```

```
                if( paths[i][j].cost < current_max.cost) {
                     current_max.id=paths[i][j].id;
                     current_max.cost=paths[i][j].cost;
                     for( k=0; k<SIZE; k++ )

     current_max.path[k]=paths[i][j].path[k];
                     current_max.pos=paths[i][j].pos;
                }
           }
     current_target.start=current_max.pos;
     current_target.end=current_max.id;

}
```

person.cpp

//This file is responsible for all functions regarding people.

```
#include "cermain.h"

person::person():dest(-1)
{
     onMove = false;
}

void person::Move(ofstream &outfile, apmatrix <int> &pt,
apmatrix <int> &ct)
{
     int i, traf, sl, p, c;
     if (! onMove)
     {
          apvector <int> moves;
          for (i = 0; i < SIZE; i++)
          {
               if (city[itsLoc][i] != 10000 && itsLoc != i)
               {
                    moves.resize(moves.length() + 1);
                    moves[moves.length() - 1] = i;
               }
          }
          i = rand() % (moves.length() + 1) - 1;

          if (i != -1)
          {
               p = pt[itsLoc][moves[i]];
               c = ct[itsLoc][moves[i]];
               sl = city[itsLoc][moves[i]]*5;
               traf = ((sl)/5) * (p + 5 * c + 1) +1;
               outfile << "moveperson " << id << " " <<
map[moves[i]][0] << " " << map[moves[i]][1] << " " << traf
<< endl;
               onMove = true;
```

```
                stepstogo = traf;
                dest = moves[i];
                pt[itsLoc][moves[i]]++;
                pt[moves[i]][itsLoc]++;
           }
     }
     else
     {
           if (stepstogo == 0)
           {
                onMove = false;
                pt[itsLoc][dest]--;
                pt[dest][itsLoc]--;
                itsLoc = dest;
                dest = -1;
           }
           else
                stepstogo--;
     }
}
```

vehicle.cpp

//This file is responsible for all functions regarding non-emergency response vehicles.

```
//#include "vehicle.h"
#include "cermain.h"

vehicle::vehicle():dest(-1)
{
     onMove = false;
}

void vehicle::Move(ofstream &outfile, apmatrix <int> &pt,
apmatrix <int> &ct)
{
     int i, sl, cs, p, c, stps;
     if (! onMove)
     {
           cerr << "car move\n";
           apvector <int> moves;
           for (i = 0; i < SIZE; i++)
           {
                if (city[itsLoc][i] != 10000 && itsLoc != i)
                {
                     moves.resize(moves.length() + 1);
                     moves[moves.length() - 1] = i;
                }
           }
```

```
          i = rand() % (moves.length() + 1) - 1;

          if (i != -1)
          {
               p = pt[itsLoc][moves[i]];
               c = ct[itsLoc][moves[i]];
               sl = city[itsLoc][moves[i]]*5;
               cs = city[itsLoc][moves[i]];
               stps = ((sl) / 5) * (p + 5 * c + 1) / cs +1;
               outfile << "movecar " << id << " " <<
map[moves[i]][0] << " " << map[moves[i]][1] << " " << stps
<< endl;
               onMove = true;
               stepstogo = stps;
               dest = moves[i];
               ct[itsLoc][moves[i]]++;
               ct[moves[i]][itsLoc]++;
          }
     }
     else
     {
          cerr << "car !move\n";
          if (stepstogo == 0)
          {
               onMove = false;
               ct[itsLoc][dest]--;
               ct[dest][itsLoc]--;
               itsLoc = dest;
               dest = -1;
          }
          else
               stepstogo--;
     }
}
```

Graphics

```
//main.cpp
//contains main() function that controls all program
direction
//Matt Strange
#include "cgmain.h"
//externals////////////////////////////////////
extern SDL_Surface *primary; //primary surface ptr
extern char cerfilename[_MAX_PATH];    //the filename of the
CER file

int main(int argc, char *argv[])
{
    //initalize SDL and open file etc.
    initSDL();
    //useful variables
    SDL_Event event;
```

```
    //this counter will hold the numerical data that feeds
into the
    //bitmap filenames
    unsigned int counter = 0;
    //do the drawing
    for(int iter=0; iter<(cgnumSTEP()-1); iter++)
    {
        //where is the next STEP?
        int start,end;    //the line numbers
        cgSTEP(iter,start,end);
        cggetv(start, end);
        bool finished = 0;
        Uint32 ticks;   //keep track of time
        while(finished == 0)        //move things
        {
            ticks = SDL_GetTicks();
            SDL_PollEvent( &event );    //any window events
            if(event.type == SDL_QUIT)  //close when they
click the X
                exit(0);
            cgclearbb();    //fill the BB w/ black
            cgtext("cerGUI v3.1",500,0);
               string stepdisp = "Step Number: " +
cgitos(iter);
               cgtext((char*)(stepdisp.c_str()),500,15);
               cgtext("---------------------------",500,30);
               //draw everything
               finished = cgwriteBB(start,end);
               SDL_Flip(primary);
               //SDL_Delay(5);
               while(SDL_GetTicks() <= (ticks + 100));
        }
    }
    exit(0);
    return 0;
}
```

```
//cggfx.h
//Graphics Header File
//All Function Prototypes and Two Important Classes
#ifndef _CGGFX_H
#define _CGGFX_H
//includes//////////////////
#include <iostream>
#include <stdlib.h>
#include <vector>
#include <string>
#include <sstream>
#include <fstream>
#include "math.h"
#include "SDL.h"
#include "SDL_ttf.h"
#include "SDL_draw.h"
#include "windows.h"
#include "windowsx.h"
using namespace std;
//typedef///////////////////
typedef int (*FuncType)(void);
#define screen_height 480
#define screen_width 640
#define SCREEN_HEIGHT 480
#define SCREEN_WIDTH 640
//CLASSES///////////////////////////////////////////
class cgcoord {
public:
    cgcoord(){x=0; y=0;}
    cgcoord(int a, int b){x=a;y=b;}
    void set(int a, int b){x=a;y=b;}
    int retx(void){return x;}
    int rety(void){return y;}
private:
    int x;
    int y;
};
//////////////
class cgrect {
public:
    bool visible, persist, finished;
    float mapx, mapy;
    float vx, vy; //velocities
    float x,y;    //screenpos
    float myoldmapx, myoldmapy;
    float mynextstep, numsteps;
    float distx, disty;
    void setv(float qx, float qy)
    {
        vx = qx;
        vy = qy;
    }
    void move()
```

```
    {
        x += vx;
        y += vy;
    }
    void print()
    {
        cerr << "p:" << persist
             << " f:" << finished
             << " x:" << x
             << " y:" << y
             << " vx:" << vx
             << " vy:" << vy
             << " mynextstep:" << mynextstep
             << " numsteps" << numsteps << endl;
    }
};
//prototypes/////////////////
void initSDL();             //initalize SDL
void lockSDLprimary();      //lock the primary buffer
void unlockSDLprimary();    //unlock the primary buffer
void cgquit();
//when locked, draw a pixel
void putpixel(SDL_Surface *surface, int x, int y, Uint32
pixel);
BOOL cgopenfile(char *filename, int length);    //open a
file
void etest(int& rvalue, string& estring);    //test for
errors
void cgclearbb();
string cgitos(int num);
//annoying file parser functions
void cgloaddata(string filename);
int cgnumSTEP(void);
void cgSTEP(int &iter,int &startpos,int &endpos);
void zeroset(vector<cgrect> setme);
void cgtext(char* stext, int x, int y);
bool cgwriteBB(int &startpos,int &endpos);
string whatsatline(int line);
int cgstoi(string& s);
float sizebox(void);
void cggetv(int& startpos, int& endpos);
void storemaps(cgrect& r);
bool testpos(cgrect& r);
bool moverect(cgrect& r);
void checkzero(int r);
void setdistance(cgrect& r,float x, float y);
#endif
```

//cggfx.cpp
//Graphics Driver File
//Contains Implementation of All Graphics Functions

```
#include "cgmain.h"
#include <SDL_version.h>
#include <SDL_syswm.h>
```

```
//globals/////////////////
SDL_Surface *primary; //primary surface ptr
HWND main_window_handle;    //handle to the window, as per
SDL
char cerfilename[_MAX_PATH];    //the filename of the CER
file
SDL_Surface* text;  //a text surface
cgcoord worldsize;              //how big is our map
vector<int> steppos;            //linenums of STEPS
vector<cgrect> person;             //all the BOBs on the map
vector<cgrect> car;
vector<cgrect> policecar;
vector<cgrect> firetruck;
vector<cgrect> ambulance;
vector<cgrect> building;
vector<cgrect> fire;
char *home; //path of home directory
//open this ttf font in a window$ os environment
TTF_Font* font;
//colors
SDL_Color WHITE = { 0xFF, 0xFF, 0xFF, 0 };
//testing
bool zeroreported;
//functions///////////////

//initSDL/////////////////////////////////////////////////
//////////
void initSDL()
{
    //when it closes destroy SDL
    atexit(cgquit);
    //initialize the SDL video interface
    if ( SDL_Init(SDL_INIT_VIDEO) < 0 )
    {
        cerr << "Unable to initialize SDL:" <<
SDL_GetError() << endl;
        exit(1);
    }

    //initalize the SDL screens
    //set up primary to 640x480x16 windowed mode
    primary = SDL_SetVideoMode(640, 480, 16, SDL_SWSURFACE |
SDL_RESIZABLE | SDL_DOUBLEBUF);
    if ( primary == NULL ) //error
    {
        cerr << "Unable to set 640x480 video: " <<
SDL_GetError() << endl;
        exit(1);
    }
    //initalize the title
    SDL_WM_SetCaption("Centralized Emergency Response GUI
3.0", "C:\\Dev-Cpp497\\Icons\\software.ico");
    //start up SDL_TTF
```

```
    TTF_Init();
    //init our font
    font = TTF_OpenFont("c:\\windows\\fonts\\arial.ttf",12);
    //get a window handle
    SDL_SysWMinfo info;
    SDL_VERSION(&info.version);
    if ( SDL_GetWMInfo(&info) < 0 )
        exit(1);
    main_window_handle = info.window;
    //try to open a file
    cerfilename[0] = (char)NULL;   //set the first entry to
NULL
    if(!cgopenfile(cerfilename, sizeof(cerfilename)))
    {
        cerr << "Can't Open File";
        exit(1);
    }
    cgloaddata((cerfilename));  //initialize the data

    Draw_Init(); //initialize SDL_Draw
}
//void
lockSDLprimary//////////////////////////////////////////
///
void lockSDLprimary()
{
    /* Lock the screen for direct access to the pixels */
    if ( SDL_MUSTLOCK(primary))
    {
        if ( SDL_LockSurface(primary) < 0 )
        {
            cerr << "Can't lock screen:" << SDL_GetError()
<< endl;
            exit(1);
        }
    }
}
//void
unlockSDLprimary//////////////////////////////////////////
///
void unlockSDLprimary()
{
    if ( SDL_MUSTLOCK(primary) )
    {
        SDL_UnlockSurface(primary);
    }
}

//putpixel//////////////////////////////////////////////////
//////////
void putpixel(SDL_Surface *surface, int x, int y, Uint32
pixel)
{
    int bpp = surface->format->BytesPerPixel;
```

```
    /* Here p is the address to the pixel we want to set */
    Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch
+ x * bpp;

    switch(bpp) {
    case 1:
        *p = pixel;
        break;

    case 2:
        *(Uint16 *)p = pixel;
        break;

    case 3:
        if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
            p[0] = (pixel >> 16) & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = pixel & 0xff;
        } else {
            p[0] = pixel & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = (pixel >> 16) & 0xff;
        }
        break;

    case 4:
        *(Uint32 *)p = pixel;
        break;
    }
}
////////////////////////////////////////////////////////////
////
BOOL cgopenfile(char *filename, int length)
{
    OPENFILENAME ofn;   //used to hold dlgbox data
    ZeroMemory(&ofn, sizeof(OPENFILENAME));     //clear out
any leftover
                                                //stack
memory
    ofn.lStructSize = sizeof(OPENFILENAME);     //set size
flag
    ofn.hwndOwner = main_window_handle;
    //the type of file filter
    ofn.lpstrFilter = ("CER Files (*.cer)\0*.cer\0All Files
(*.*)\0*.*\0\0");
    ofn.lpstrFile = filename;   //the filename
    ofn.nMaxFile = length;      //sizeof the buffer
    ofn.lpstrTitle = ("Open CER File...");
    ofn.lpstrInitialDir = "..\\";  //align to current
directory
    ofn.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
    BOOL rvalue = GetOpenFileName(&ofn);
    return rvalue;
}
```

```
////////////////////////////////////////////////////////
//
void cgquit()
{
    SDL_FreeSurface(text);
     SDL_FreeSurface(primary);
     TTF_CloseFont(font);
     TTF_Quit();
     SDL_Quit();
}
////////////////////////////////////////////////////////
///
void etest(int& rvalue, string& estring) //test for errors,
as usual
{
    if(rvalue != 0) //error
    {
        cerr << "Error in function " << estring << endl;
        cerr << SDL_GetError();
        exit(1);    //call the prefab error return function
    }
}
////////////////////////////////////////////////////////
/////
void cgloaddata(string filename)
{
    ifstream in;
    in.open(filename.c_str());     //the cer file
    in.seekg(0);                   //put the pointer @ zero
    in >> ws;                      //skip inital whitespace
if any
    string indata = "";            //holds input data
    //the loop of input
    while(in.peek() != EOF)
    {
        if(in.peek() == ';')           //comment line
        {
          in.ignore(INT_MAX, '\n');//ignore it
     }
     else                                    //there is
something to set
     {
          in >> indata;
          }
          if(indata == "worldsize")     //set the world
          {
          int x,y;
          in >> x >> y;
          worldsize.set(x,y);
          }
          if(indata == "arrayperson")        //set the
number of people
          {
               int x;
```

```
                in >> x;
                person.resize(x);
                zeroset(person);
          }
          if(indata == "arraycar")      //set the number of
cars
          {
                int x;
                in >> x;
                car.resize(x);
                zeroset(car);
          }
          if(indata == "arraypolicecar")//set policecar
vector
          {
                int x;
                in >> x;
                policecar.resize(x);
                zeroset(policecar);
          }
          if(indata == "arrayfiretruck")  //set number of
firetrucks
          {
            int x;
            in >> x;
            firetruck.resize(x);
            zeroset(firetruck);
        }
        if(indata == "arrayambulance")  //set number of
ambulences
        {
            int x;
            in >> x;
            ambulance.resize(x);
            zeroset(ambulance);
        }
          if(indata == "arrayfire")
          {
                int x;
                in >> x;
                fire.resize(x);
                zeroset(fire);
          }
        if(indata == "setperson")           //set mappos of
person BOB
        {
            int iter, x, y;
            in >> iter >> x >> y;
            person[iter].mapx = x;
            person[iter].mapy = y;
            person[iter].x = x*sizebox();
            person[iter].y = y*sizebox();
        }
```

```
        if(indata == "setfiretruck")           //set mappos
of firetruck BOB
        {
            int iter, x, y;
            in >> iter >> x >> y;
            firetruck[iter].mapx = x;
            firetruck[iter].mapy = y;
            firetruck[iter].x = x*sizebox();
            firetruck[iter].y = y*sizebox();
        }

        if(indata == "setcar")           //set mappos of car
BOB
        {
            int iter, x, y;
            in >> iter >> x >> y;
            car[iter].mapx = x;
            car[iter].mapy = y;
            car[iter].x = x*sizebox();
            car[iter].y = y*sizebox();
        }

        if(indata == "setpolicecar")           //set mappos
of policecar BOB
        {
            int iter, x, y;
            in >> iter >> x >> y;
            policecar[iter].mapx = x;
            policecar[iter].mapy = y;
            policecar[iter].x = x*sizebox();
            policecar[iter].y = y*sizebox();
        }

        if(indata == "setambulance")           //set mappos
of ambulance BOB
        {
            int iter, x, y;
            in >> iter >> x >> y;
            ambulance[iter].mapx = x;
            ambulance[iter].mapy = y;
            ambulance[iter].x = x*sizebox();
            ambulance[iter].y = y*sizebox();
        }

     indata = "";                         //reset indata
          in >> ws;                            //skip any
leftover whitespace
     } //finished reading file
} //we be initialized
////////////////////////////////////////////////
void zeroset(vector<cgrect> setme)
{
    for(int i=0; i<setme.size(); i++)
    {
```

```
        setme[i].mapx = 0;
        setme[i].mapy = 0;
        setme[i].x = 0;
        setme[i].y = 0;
        setme[i].visible = 1;
    }

}
//////////////////////////////////////////////////////
int cgnumSTEP(void)
{
    ifstream in(cerfilename);
    char buffer[100];
    int numlines = 0;
    int numsteps = 0;
    string indata;
    while(in.peek() != EOF)
    {
        in.getline(buffer, 100);
        indata = buffer;
        if(indata == "STEP")
        {
            numsteps++;
            steppos.resize(numsteps);
            steppos[numsteps-1]=numlines;
        }
        numlines++;
    }
    return numsteps;
}
//////////////////////////////////////////////////////
void cgSTEP(int &iter,int &startpos,int &endpos)
{
    //we get the 'id number' of the step and use send back
STEP+1 and (STEP+1)-1
    startpos = steppos[iter];
    endpos = steppos[iter+1];
    return; //all done...
}
//////////////////////////////////////////////////////
void cgtext(char* stext, int x, int y)   //writes to BB
{
    //uses some global varibles
    text = TTF_RenderText_Blended(font,stext, WHITE);
    SDL_Rect dstrect;
    dstrect.x = x;
    dstrect.y = y;
    dstrect.w = text->w;
    dstrect.h = text->h;
    //destroy the stuff behind this
    Uint32 black = SDL_MapRGB(primary->format, 0, 0, 0);
    SDL_FillRect(primary, &dstrect, black);
    //blit
    SDL_BlitSurface(text, NULL, primary, &dstrect);
```

```
}
//////////////////////////////////////////////////
void cgclearbb()
{
    //clear buffer
    Uint32 black = SDL_MapRGB(primary->format, 0, 0, 0);
    SDL_Rect buff = {0,0,primary->w,primary->h};
    SDL_FillRect(primary, &buff, black);
}
//////////////////////////////////////////////////
string cgitos(int num)
{
    stringstream ss;
    ss << num;
    return ss.str();
}
//////////////////////////////////////////////////
bool cgwriteBB(int &startpos,int &endpos)
{
    //use up lines before...
    ifstream in(cerfilename);
    char buffer[100];
    for(int i=0; i<=startpos; i++)
        in.getline(buffer, 100);

    //now get the important stuff
    string enddata = whatsatline((endpos-2));
    string indata = "__##__#...";
    //what's important here is to change the
velocities/vectors of the
    //rects.  they are all drawn afterwards

    //set done == 0 so that all progs must report
    //done == 1 to continue to next step
//    vector<int> done(10);
//    for(int i=0; i<done.size(); i++)
//        done[i] = -1;
//   cerr << done.size() << endl;
    zeroreported = false;

     int outputpos = 0;
    while(indata != enddata)
    {
       istringstream ss;
       in.getline(buffer, 100);
       indata = buffer;
       ss.str(indata);
       //we'll assume that this is a cool move func. w/ 5
parts
       //command, id, x, y, numsteps
       string command, sid, sx, sy, snumsteps;
       ss >> command >> sid >> sx >> sy >> snumsteps;
       int id = cgstoi(sid),
             x = cgstoi(sx),
```

```
             y = cgstoi(sy),
             numsteps = cgstoi(snumsteps);

       firetruck[0].print();

       cgtext((char *)command.c_str(),500,45);
//        cerr << firetruck[0].persist << ' ' <<
firetruck[0].vx
//             << firetruck[0].vy << endl;
       //process this move
          if((command == "moveperson") &&
(person[id].persist == 0))
             checkzero(moverect(person[id]));
          if((command == "movecar") && (car[id].persist ==
0))
            checkzero(moverect(car[id]));
          if((command == "movepolicecar") &&
(policecar[id].persist == 0))
            checkzero(moverect(policecar[id]));
          if((command == "movefiretruck") &&
(firetruck[id].persist == 0))
            checkzero(moverect(firetruck[id]));
          if((command == "moveambulance") &&
(ambulance[id].persist == 0))
            checkzero(moverect(ambulance[id]));

          if(command == "setpath")
          {
            //we need to get the data out of the ss again
since
            //it's a huge member line
            ss.str(indata);
            string ta;
            int x1,y1,x2,y2,traffic,speedlimit;
            ss >> ta >> x1 >> y1 >> x2 >> y2 >> traffic;
            //draw a line that will represent the traffic
constant
            Uint32 t = SDL_MapRGB(primary-
>format,(int)(100+traffic*(155/100)),0,0);
            Draw_Line(primary, sizebox()*x1, sizebox() * y1,
                      sizebox()*x2, sizebox()*y2, t);
          }
     }
    //process persistant rects
    for(int i=0; i<person.size(); i++)
    {
        if(person[i].persist)
            checkzero(moverect(person[i]));
        //cerr << moverect(person[0]) << endl;
    }
    for(int i=0; i<car.size(); i++)
    {
        if(car[i].persist)
            checkzero(moverect(car[i]));
```

```
    }
    for(int i=0; i<policecar.size(); i++)
    {
        if(policecar[i].persist)
            checkzero(moverect(policecar[i]));
    }
    for(int i=0; i<firetruck.size(); i++)
    {
        if(firetruck[i].persist)
            checkzero(moverect(firetruck[i]));
    }
    for(int i=0; i<ambulance.size(); i++)
    {
        if(ambulance[i].persist)
            checkzero(moverect(ambulance[i]));
    }//end persistants

     //person[0].print();

     //vectors/velocities done. let's draw
     for(int i=0; i<fire.size(); i++)
     {
        Uint32 orange = SDL_MapRGB(primary->format, 255,
100, 0);
        SDL_Rect rect = {fire[i].x,fire[i].y, sizebox(),
sizebox()};
        if(fire[i].visible)
            SDL_FillRect(primary, &rect ,orange);
     }
    for(int i=0; i<person.size(); i++)
    {
        Uint32 brown = SDL_MapRGB(primary->format,
139,69,19);
        SDL_Rect rect = {person[i].x,person[i].y, sizebox(),
sizebox()};
        //if(person[i].visible)
        {
            SDL_FillRect(primary, &rect ,brown);
        }
    }
    for(int i=0; i<car.size(); i++)
    {
        Uint32 green = SDL_MapRGB(primary->format,
102,205,170);
        SDL_Rect rect = {car[i].x,car[i].y, sizebox(),
sizebox()};
        //if(car[i].visible)
            SDL_FillRect(primary, &rect ,green);
    }
    for(int i=0; i<policecar.size(); i++)
    {
        Uint32 blue = SDL_MapRGB(primary->format,0,0,225);
        SDL_Rect rect = {policecar[i].x,policecar[i].y,
sizebox(), sizebox()};
```

```
        //if(policecar[i].visible)
            SDL_FillRect(primary, &rect , blue);
    }
    for(int i=0; i<firetruck.size(); i++)
    {
        Uint32 red = SDL_MapRGB(primary->format,255,0,0);
        SDL_Rect rect = {firetruck[i].x,firetruck[i].y,
sizebox(), sizebox()};
        //if(firetruck[i].visible)
            SDL_FillRect(primary, &rect , red);
    }
    for(int i=0; i<ambulance.size(); i++)
    {
        Uint32 white = SDL_MapRGB(primary-
>format,255,255,255);
        SDL_Rect rect =
{ambulance[i].x,sizebox()*ambulance[i].y, sizebox(),
sizebox()};
        //if(ambulance[i].visible)
            SDL_FillRect(primary, &rect , white);
    }
//    cerr << zeroreported << endl;
    //see if we can finish now
//    for(int l=0; l < done.size(); l++)
//    {
//
//        //if there is ANY zero, we can't go on
//        //might be too harsh on bad code?
//        if(done[l] == 0)
//        {
//            cerr << "zero from: " << l << endl;
//            return 0;
//        }
//    }
    if(zeroreported == true)
        return 0;
    if(zeroreported == false)
        return 1;
}
////////////////////////////////////////////////////
string whatsatline(int line)
{
    //use up lines before...
    ifstream in(cerfilename);
    char buffer[100];
    for(int i=0; i<=line; i++)
        in.getline(buffer, 100);
    //now get the important stuff...
    string goodstuff;
    in.getline(buffer, 100);
    goodstuff = buffer;
    return goodstuff;
}
////////////////////////////////////////////////////
```

```
int cgstoi(string& s)
{
     stringstream ss(s);
     int r;
     ss >> r;
     return r;
}
////////////////////////////////////////////////////
float sizebox(void)
{
    int a = (SCREEN_HEIGHT/worldsize.retx());
    int b = (SCREEN_WIDTH/worldsize.rety());
    int rval = (a<b)?a:b;
    return rval;
}
////////////////////////////////////////////////////
void cggetv(int& startpos, int& endpos)
{
    //use up lines before...
    ifstream in(cerfilename);
    char buffer[100];
    for(int i=0; i<=startpos; i++)
        in.getline(buffer, 100);

    //now get the important stuff
    string enddata = whatsatline((endpos-2));
    string indata = "__##__#...";
    //what's important here is to change the
velocities/vectors of the
    //rects.  they are all drawn afterwards

    //set done == 0 so that all progs must report
    //done == 1 to continue to next step
    int done = 0;
     int outputpos = 0;
    while(indata != enddata)
    {
       istringstream ss("");
       in.getline(buffer, 100);
       indata = "";
       indata = buffer;
       ss.str(indata);
       //we'll assume that this is a cool move func. w/ 5
parts
       //command, id, x, y, numsteps
       string command, sid, sx, sy, snumsteps;
       ss >> command >> sid >> sx >> sy >> snumsteps;
       int id = cgstoi(sid),
             x = cgstoi(sx),
             y = cgstoi(sy);
       float numsteps = cgstoi(snumsteps);

        //process this move
          if(command == "moveperson")
```

```
     {
         //dump into myoldmap... values
         person[id].numsteps = numsteps;
            storemaps(person[id]);

               //where does the rect need to go
               float dirx = (x-person[id].mapx)/numsteps;
               float diry = (y-person[id].mapy)/numsteps;
               person[id].visible = 1;

//             cerr << "New V:" << dirx << ' ' << diry << "
id " << id << endl;
//             cerr << "@ startpos = " << startpos << endl;
//             cerr << "from:"  << person[id].mapx << ' ' <<
person[id].mapy
//                     << " to " << x << ' ' << y << endl;

            //set velocities
               person[id].setv(dirx,diry);
               setdistance(person[id],x,y);
          }
          if(command == "movecar")
     {
              //dump into myoldmap... values
            car[id].numsteps = numsteps;
            storemaps(car[id]);
               //where does the bob need to go
               float dirx = (x-car[id].mapx)/numsteps;
               float diry = (y-car[id].mapy)/numsteps;
               car[id].visible = 1;

            //set velocities
               car[id].setv(dirx,diry);
               setdistance(car[id],x,y);
          }
          if(command == "movepolicecar")
     {
         //dump into myoldmap... values
         policecar[id].numsteps = numsteps;
            storemaps(policecar[id]);
               //where does the bob need to go
               float dirx = (x-policecar[id].mapx)/numsteps;
               float diry = (y-policecar[id].mapy)/numsteps;
               policecar[id].visible = 1;

               policecar[id].setv(dirx,diry);
               setdistance(policecar[id],x,y);
          }
          if(command == "movefiretruck")
     {
         //dump into myoldmap... values
         firetruck[id].numsteps = numsteps;
            storemaps(firetruck[id]);
               //where does the bob need to go
```

```
               float dirx = (x-firetruck[id].mapx)/numsteps;
               float diry = (y-firetruck[id].mapy)/numsteps;

//             cerr << "j " << dirx << ' ' << diry << endl;

               firetruck[id].visible = 1;

            firetruck[id].setv(dirx,diry);
               setdistance(firetruck[id],x,y);
//            cerr << "k " << firetruck[id].vx << ' ' <<
firetruck[id].vy << endl;
          }
          if(command == "moveambulance")
     {
         //dump into myoldmap... values
         ambulance[id].numsteps = numsteps;
            storemaps(ambulance[id]);
               //where does the bob need to go
               float dirx = (x-ambulance[id].mapx)/numsteps;
               float diry = (y-ambulance[id].mapy)/numsteps;
               ambulance[id].visible = 1;

            ambulance[id].setv(dirx,diry);
               setdistance(person[id],x,y);
        }
          if(command == "setfire")
          {
               fire[id].mapx = x;
               fire[id].mapy = y;
            fire[id].x = fire[id].mapx * sizebox();
            fire[id].y = fire[id].mapy * sizebox();
               fire[id].visible = 1;
          }
          if(command == "killfire")
            fire[id].visible = 0;

        //process persistant rects
        for(int i=0; i<person.size(); i++)
        {
            if(person[i].persist)
                person[i].finished = 0;
        }
        for(int i=0; i<car.size(); i++)
        {
            if(car[i].persist)
                car[i].finished =0;
        }
        for(int i=0; i<policecar.size(); i++)
        {
            if(policecar[i].persist)
                policecar[i].finished =0;
        }
        for(int i=0; i<firetruck.size(); i++)
        {
```

```
            if(firetruck[i].persist)
                firetruck[i].finished = 0;
        }
        for(int i=0; i<ambulance.size(); i++)
        {
            if(ambulance[i].persist)
                ambulance[i].finished = 0;
        }//end persistants

     }
}
////////////////////////////////////////////////////////
void storemaps(cgrect& r)
{
    //dump the current map values into the myoldmap...
values
    r.myoldmapx = r.mapx;
    r.myoldmapy = r.mapy;
    //now reset the mynextstep varible to 1
    r.mynextstep = 1;
    //turn on persistance if more than one step
    if(r.numsteps>1)
        r.persist = 1;
    else
        r.persist = 0;
    //we aren't finished
    r.finished = 0;
}
////////////////////////////////////////////////////////
////
bool testpos(cgrect& r)
{
    float rmx,rmy;
    rmx = (r.x / sizebox());
    rmy = (r.y / sizebox());

    float addx=0, addy=0;
    //this holds the distance between the start and
    //endpoints, since all algorithms are standard for

    if(r.vx != 0.0)
    {
        if(r.vx>0.0)
            addx = r.distx*(r.mynextstep / r.numsteps);
        if(r.vx<0.0)
            addx = 0 - r.distx*(r.mynextstep / r.numsteps);
    }
    if(r.vy != 0.0)
    {
        if(r.vy>0.0)
            addy = r.disty*(r.mynextstep / r.numsteps);
        if(r.vy<0.0)
            addy = 0 - r.disty*(r.mynextstep / r.numsteps);
    }
```

```
    cerr << rmx << ' ' << rmy << ' ' << endl
         << addx << ' ' << addy << endl
         << r.myoldmapx << ' ' << r.myoldmapy << endl
         << r.distx << ' ' << r.disty << endl;

    if((rmx == (r.myoldmapx + addx)) &&
       (rmy == (r.myoldmapy + addy)))
    {
        //nextstep!
        r.mynextstep++;
        if(r.mynextstep == (r.numsteps + 1))//not zero based
        {
            //shut off persistance... rect is finished
            r.persist = 0;
            //reset velocities
            r.setv(0,0);
        }
        r.finished = 1;     //don't keep moving in this step
        return 1;
    }
    if(r.finished)
        return 1;
    //all checks failed
    return 0;
}
////////////////////////////////////////////////////////////
//
bool moverect(cgrect& r)
{
    //move
    if(r.finished == 0)
        r.move();
    //Update the MapN values as per screen position

    r.mapx = ((float)(r.x)/(float)(sizebox()));
    r.mapy = ((float)(r.y)/(float)(sizebox()));
    //If already there then stop and quit
    if(testpos(r))
        return 1;
    else
     return 0;
}
////////////////////////////////////////////////////////////
//
void checkzero(int r)
{
    if(r == 0)
        zeroreported = true;
}
////////////////////////////////////////////////////////////
///
void resetfinished(cgrect& r)
{
```

```
    if(r.mynextstep <= (r.numsteps + 1))
    {
        //will be moving again in this step
        r.finished = 0;
    }
}
////////////////////////////////////////////////////////////
////
void setdistance(cgrect& r,float x, float y)
{
    //r.dist = sqrt(pow((x-r.mapx),2) + pow((y-r.mapy),2));
    r.distx = abs(x-r.mapx);
    r.disty = abs(y-r.mapy);
}
////////////////////////////////////////////////////////////
//////
```