CSCE2110

Section 213

Group 6

October 21st, 2022

SimCity Project Design Document

Trevonne Bridges, Chandler Garret, Nathan Jodoin, and Salma M. Omary

**Diagram

Description automatically generatedPart 1 – Functionality and Diagrams**

Above is the UML Class diagram designating the relationships between each of our classes, including logical data requirements. The map struct is an implementation of zone child objects in a two-dimensional array, bounded by an X and Y size.

The zone class has 7 children, including 3 functional classes and 4 simple children representing non-functional data. The three functional zones have populations which are modified by a series of global functions. For growth in industrial and commercial zones, workers are assigned from residential zones. As population increases for commercial zones, goods are transferred from industrial to commercial zones.

**Process Flow DiagramsDiagram

Description automatically generatedDiagram

Description automatically generated**

**Diagram

Description automatically generated**

Diagram

Description automatically generated

Diagram

Description automatically generated**Level 1 – Analysis Process Flow Diagram**

**Part 2 – Data Storage**

At runtime, the program will read in a series of characters from a provided file. Each character will be translated into a zone object child, and then stored in a two-dimensional vector. The two-dimensional vector will be the only full copy of each zone object, and all other data structures will retain pointers to the 2D vector addresses for each object.

To calculate remote adjacencies, it is most efficient for each node to retain a list of direct adjacencies, otherwise known as local adjacencies. Each zone object will have a vector data member of size 8 populated with zone pointers. Index 0 represents the top left corner adjacency, and each subsequent index represents a clockwise rotation around the node.

Each cycle, commercial and industrial nodes need to probe surrounding commercial and industrial zones to draw on workers and goods. Calculating adjacencies at each update is computationally expensive. A series of adjacency lists for each node, calculated once at runtime, will allow for the simulation to run more efficiently. As workers and goods may travel via road, a local adjacency list will not suffice. Each industrial and commercial node requires a list of distantly adjacent residential nodes which can provide workers. Closer workers are preferred, so the list should be sorted by distance along the available path or paths. This may be accomplished naturally with breadth first search. Similarly, as commercial nodes also receive goods locally and by road, an additional list of industrial adjacencies is required.

To avoid instances of multiple state, our program will access all of our data from the top level through a series of ordered function calls. To facilitate this functional approach, there will be a minimum of three linked lists, one for each type of populated zone. During to update and display loop of the simulation, the lists will be the primary means by which our zone objects are accessed. For example, pollution, which affects populated zones, may be update through a function which iterates through the list of each industrial zone, and then for each zone updates the adjacent nodes, moving outward until the appropriate depth has been reached. The functional list sorting will be by population from highest to lowest. Each functional working list will be populated with zone object pointers, pointing at each object in the original two-dimensional vector.

**Part 3 – File Organization**

Our files will be structured on an as needed basis. As we are using a procedural and functional approach, we will make limited use of header files only when they are deemed necessary during the coding of the project.

The following are the existing files in our project as planned thus far:

* simcity/
* main.cpp
  + the main function
* analysis.cpp
  + included in main, the analysis process
* commercial.cpp
  + commercial class definition
* display.cpp
  + map display logic
* file\_io.cpp
  + file IO and initialization logic
* industrial.cpp
  + industrial zone class definition
* residential.cpp
  + residential zone class definition
* unpop\_zones.cpp
  + unpopulated zone simple class definitions
* update\_map.cpp
  + population and pollution update logic
* zone.cpp
  + zone superclass definition