Zombie Apocalypse Simulation

by Garrett Daviscourt, Zach Marble, and Jacob Ojennus

Design

This zombie apocalypse takes place in the fictional city of Simville. It is comprised of 4 districts containing 2000 denizens in total. There are 4 denizen types; Ignorant, Alarmed, Zombie, and Doctor. The four districts are the Suburbs, Downtown, Medical Hill, and University District. Our first zombie appears in University District, and 9 doctor are initialized in Medical Hill. The simulation will run until all 2000 denizens have become zombified! On average this process takes anywhere from 4.5 days to a little over a week given our probabilities. After everyday passes, the statistics will be stored in an output file.

Denizen Class

The denizen class is used to manage the different types of denizens in our simulation. It utilizes polymorphism and inheritance. Each denizen has a name and an infection probability. They also have a polymorphic interact and move functions. The first child class of Denizen is the Ignorant class. They have a 20% chance of being infected by a zombie and a 2% chance of movement. They cannot change another denizen's type when interacting. The next child class is the Alarmed class. The alarmed denizens have a 10% chance of infection, and a 10% chance of moving to a new district. When an alarmed denizen interacts with an ignorant person, there is a 25% chance of turning them into an alarmed denizen. They cannot change they type of any other denizens. The Zombie class has a 0% chance of infection and a 5% chance of moving. They can change they type of any denizen they interact with, excluding themselves. Finally, the Doctor class has an 8% chance of being infected, and a 90% chance of moving districts. They also have a 90% chance of healing a zombie. When this happens, the zombie is converted into an alarmed denizen. These classes heavily rely on the <cstdlib> library, specifically the rand() function.

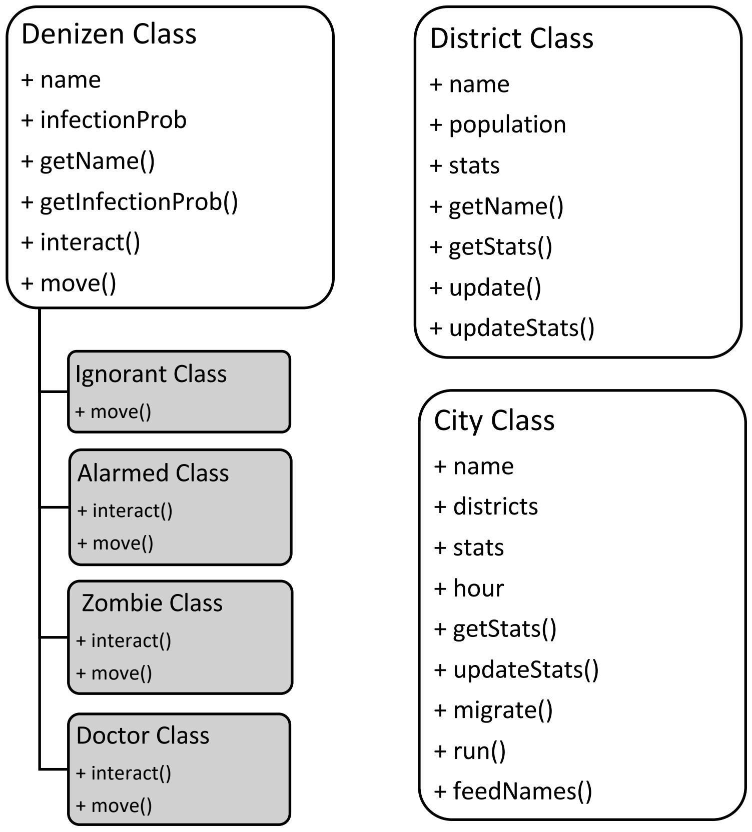
District Class

Our district class is how we manage interactions between denizens in each area. It is comprised of a vector of pointers to denizens called population. We also have map with the key being a string and the value being an integer that manages the current statistics in a district. The string represents the type of each denizen, and the value is the amount of each denizen. The most notable function in this class is our update() function. It is a void function that does not take in any parameters, but it has everybody in the district interact with a denizen at a random index. We initially had this be a two-way interaction, where the person at index i interacts with the random index and vice versa. We found this accelerated the spread of the zombies too much, so we settled with a one-way interaction.

City Class

The city class is where we conjoin all our classes and simulate the apocalypse. It utilizes a linked list to store all our districts, along with its own map similar in structure to the district map that stores all the city stats. It also has an integer that keeps track of the current hour. Our migrate function was by far one of the most difficult parts of this project. The goal is to call the move function in the denizen class and move the denizen to the neighboring districts. Our final solution involved using null pointers as to not disrupt the iteration through the entire population. We learned that we could utilize the vector erase function to remove every instance of the nullptr after the entire vector had been iterated through. Our run function is the heart of this project. It generates an output file and runs every crucial process until every denizen has become zombified. We also have a feed names function that uses exception handling.

UML



Reflection

In the end, we didn’t accomplish what we originally set out to do. We really wanted to build a project modeled after a fantasy world, but the number of classes and interactions that would have had to be done complicated things too much. We tried, but we had to pivot to something that we could complete in the allotted time and really polish. If we were to continue, we would love to build out the project to the original spec and find a way to output a graph. Overall, this project was a great learning experience, and we cannot wait to apply what we have learned to our future endeavors. Thank you, Dr. Bell, for a wonderful semester!