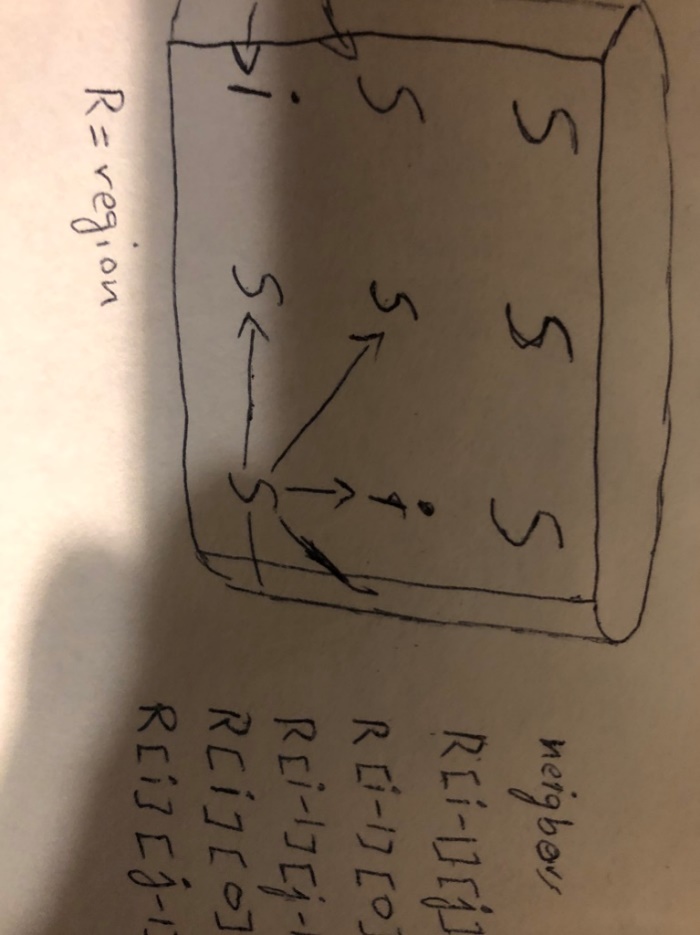
**Design Document**

**Design Process**

For the assignment we created several drawings simulating the region to come up with an approach and gather our intuition. We concluded that some sort of 2D array type of structure would be necessary to simulate the region. We decided to work individually on the coding and to either combine the best part of our projects into one or to just use the project that works the most efficiently. Team members came up with several creative approaches, using structs and other methods but we decided on a simple 2D vector approach and built on that. Once we settled on an approach we worked collaboratively distributing a single file on canvas. We ran into many issues using vectors there were index out of bounds errors, and segmentation violations in runtime we had to debug and fix. We then had to decide on what approach to use for transitioning functions, iterative or recursive were the two options. Upon consultation we decided on an indirect recursive approach using two similar transition functions.



**Data Structure**

The data structures we used in this code are two-dimensional vectors. Since vectors are dynamically sized, it gives us an advantage over using arrays. We used a two dimensional vector of strings to hold all the “S”-“I”-“R”-“V” state variables in the region. We used a two-dimensional vector of int to simulate the infectious period for all infected in the region. The integer vector is initialized at 0 for every variable in the primary String vector. We had to use several temporary one-dimensional vectors to create the 2D vectors.

**System Functionality**

First we created a readfile function that is used to retrieve the input file. When the file is opened, it reads the first three line into string variables that consume threshold, infectious period and display rate. We used for loops to loop through the string to find the integers and store them as integer values into integer reference variables. The reference variables store the integer values into a corresponding threshold, infectious period, and display rate function in main for modularity.

The readfile reads the matrix using a while loop, the while loop; code block breaks the row at delimiter “,” and reads the row into a temp vector. The temp vector is appended to the 2D vector and the temp vector is cleared to read the next row. After reading in the region the file is closed.

We created a mile stone function for the mile stone. Void function mile stone holds vector string as its argument and contains int counts S,V,I, and R which are all initialized at zero. A for loop is initiated that uses conditionals of string comparisons and increments the count of S,V,I and R for every instance of “s”, “i”, “r”, and “v” in the vector.

The second and third static void function are used to transition between S => I => R => V states and simulate period in “days”. The functions are identical recursive they switch back and forth every other “day”. We used if statements to detect the states of SIRV model.

The first if statements finds instances of “s”: susceptible units in the region. We used case based if statements for each case where “s” could be and used nested if to define the conditions for neighbors in each case. The if statements look “i”: infected in the neighborhood of “s”. We count infected neighbors of “s” and if the count exceeds threshold “s” switches to infected.

The second if statements detects “i” and checks and at every instance of “i” increments the total count of “i” and increments the corresponding variable in the integer vector that tracks infected period for each “i”. It uses a nested of to check if the infected period exceeds the infectious period the “i” switches to recovered “r”. Count of “i” is used in the termination condition for the recursion. If the count of “i” is greater than 0 the recursion continues.

The last condition is used to display the final state of the region, if it is not already displayed within the bounds of the display rate. The last void function is used to display the final state of the region and displays final conditions.

**Team Work**

Uzair Akram formulated the SIRV transition algorithm and wrote the code for the transition function for SIRV modeling using a recursive approach. Created 2D vectors for simulating the infected period of the infected states. Wrote details for the system functionality and the Data structure portion of the design document. Tested and debugged code to ensure functionality on different systems and CSE machine.

Rory Spralls created the input function to read in the file and created the milestone function. Tested and debugged the code and made contributions to transition functions. Worked on the design document.

Reevs Cota worked on the display of the output on the CSE machine, debugged and tested the code and drafted the design document. Contributed to the design of transition function and corrected errors in the code..