**Group Name - 2100Project**

**Group Members**

**Kevin Pham**

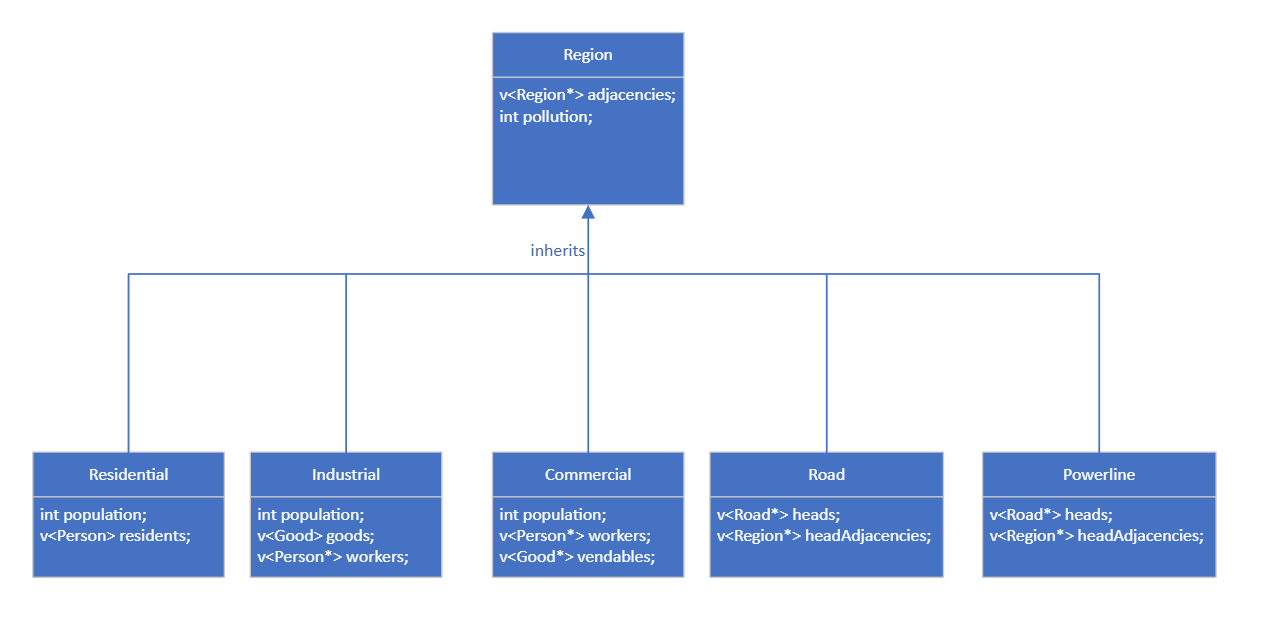
**Divyam Jain**

**Suraj Varne Sheela**

General Overview

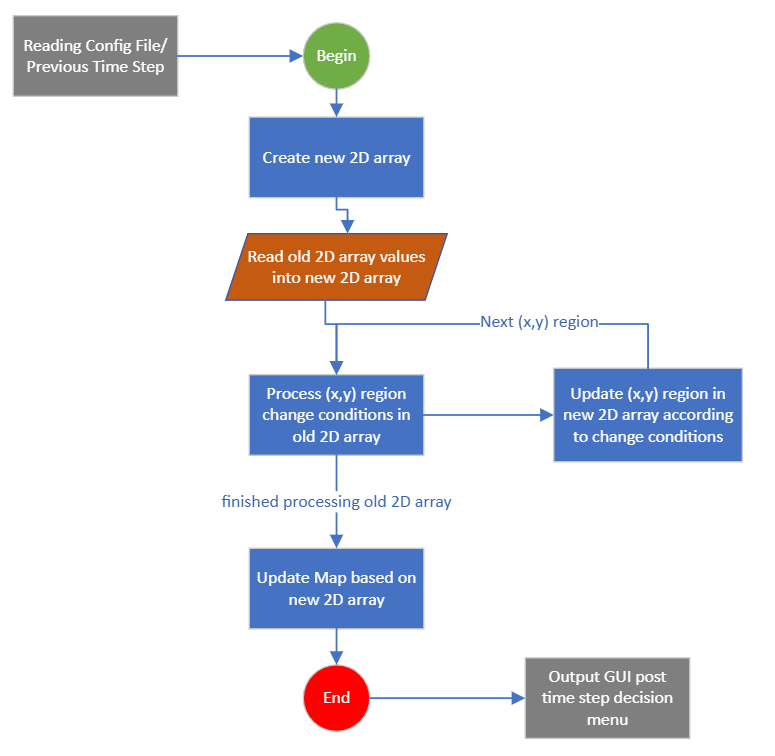
Regions:

There are a handful of different regions (Residential, Commercial, Road, Powerline, etc.), all of which have adapting game states based on their surroundings. Conditions for such changes will vary, however, many attributes such as population will be contained in almost all of them. A region class/struct will be created to normalize the needs of every different region into a single datatype, 2D vector array.



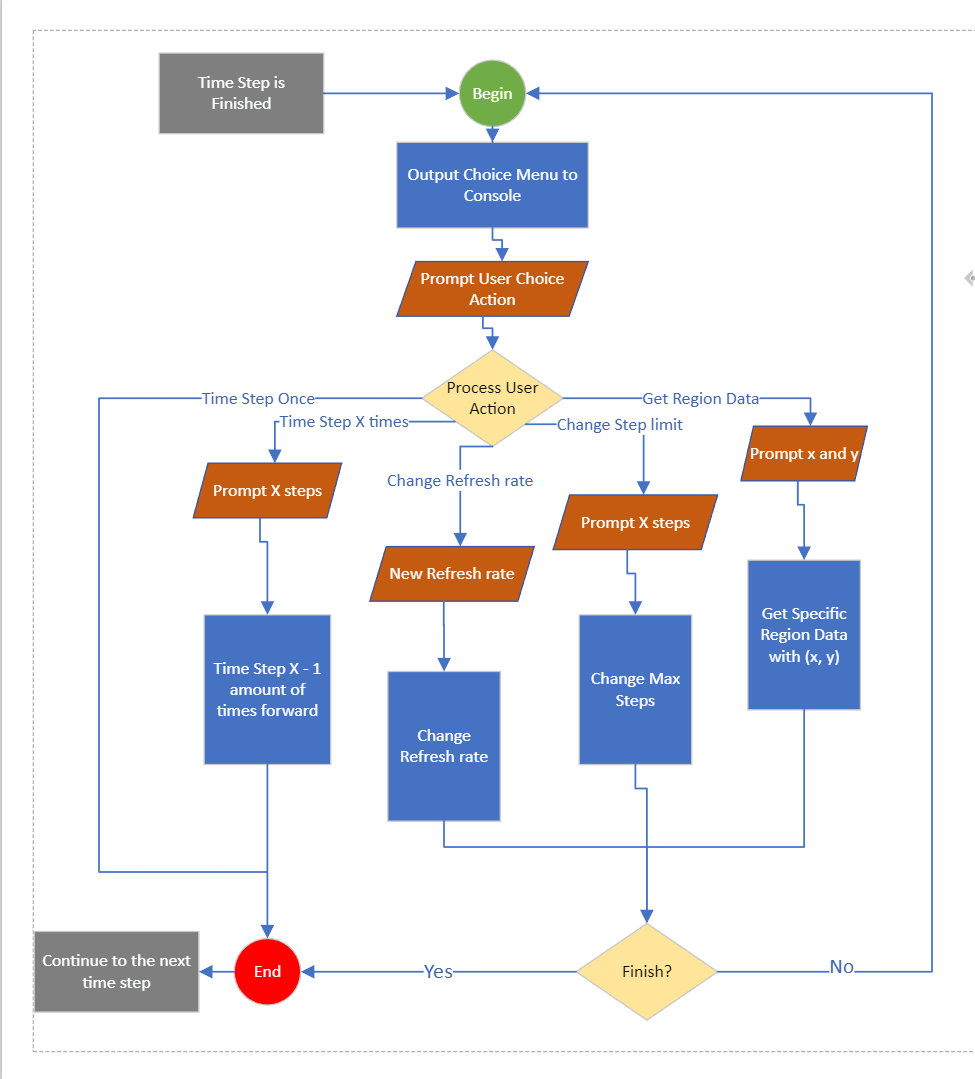
Time Step Process:

Each time step in the simulation requires iterating through every element in the array and updating regions based on their growth conditions. We will use a variable sizeable 2D vector array to store the state of every region, as well as call an update function to each region. As we do not want to update a region based on partially updated surroundings (as we are using an iterative approach), a copy of the old and new region must be kept in two separate 2D arrays.



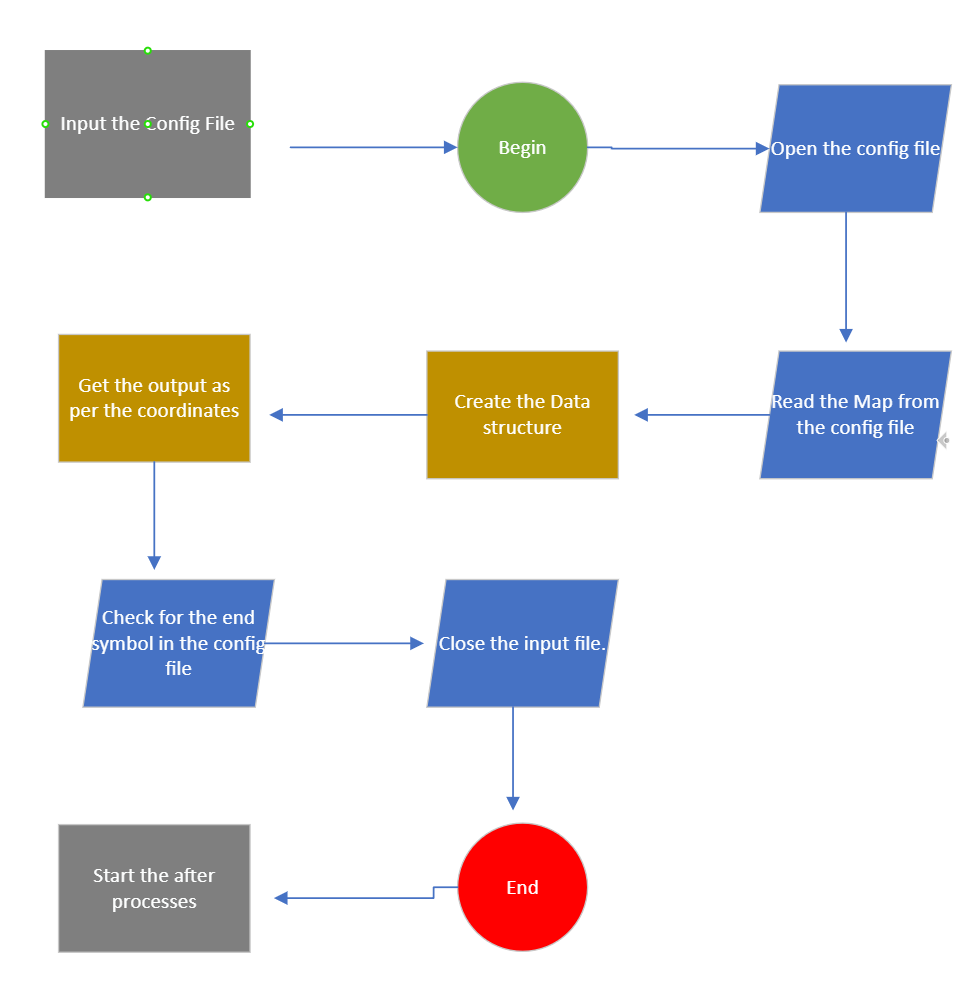
Decision GUI:

The city simulation must allow a user to analyze and evaluate each detail in every time step of the process. For each process, a GUI will present the user with numerous options such as getting population for a single region, stepping forward in time, and returning the specifics of the working populous. After processing each time step, the user will be shown a menu in the console noting possible analytical functions. The user will then be prompted to enter a single character, in which the GUI function will store and process accordingly.



Reading the Config File/Creating the 2D Array:

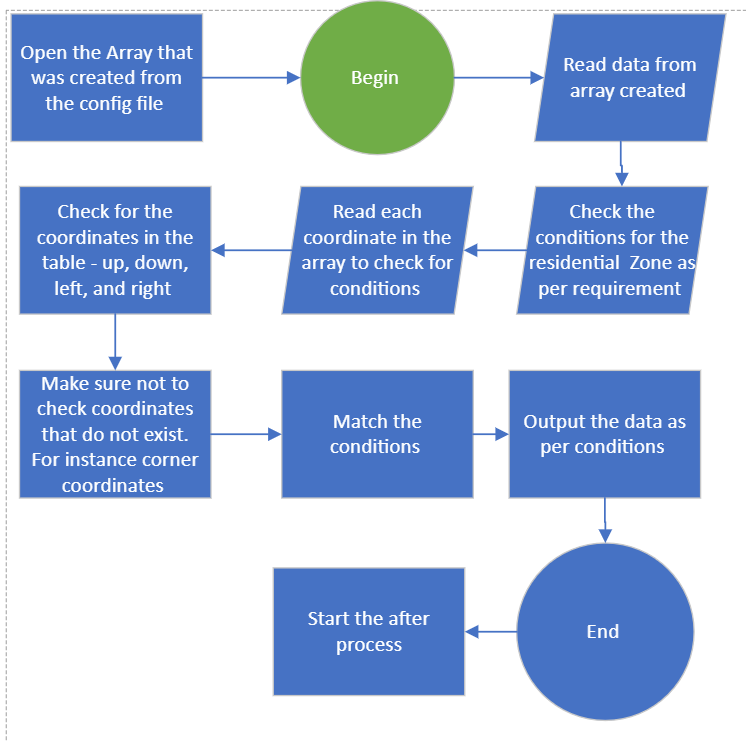
The goal is to process data with each element of the array as each element will be having its own unique (x, y) coordinates (x is the row and y is the column). After creating the array, it is going to fill each element same as given in the config file. The objective is to open the config file using the fstream library in the C++ compiler and then read the grip/map from the config file, hence giving the total no. of rows and columns.



Residential, Industrial, and Commercial Zones:

This is where the main process to check for the conditions starts in the program. The main task here is to access the array that was created in the previous step and read the data. At the same time, we will also have to keep in mind that corner values will not have up or down values, so we need to read and match this data carefully. After this step, we can output the result of the conditions and pass it on to the next step.

(Shown below is the design for the Residential Zone)



Rules for Residential, Industrial and Commercial Zones:

1. Residential Zone

Diagram

Description automatically generated

1. Industrial Zone

Diagram

Description automatically generated

1. Commercial Zone

Diagram

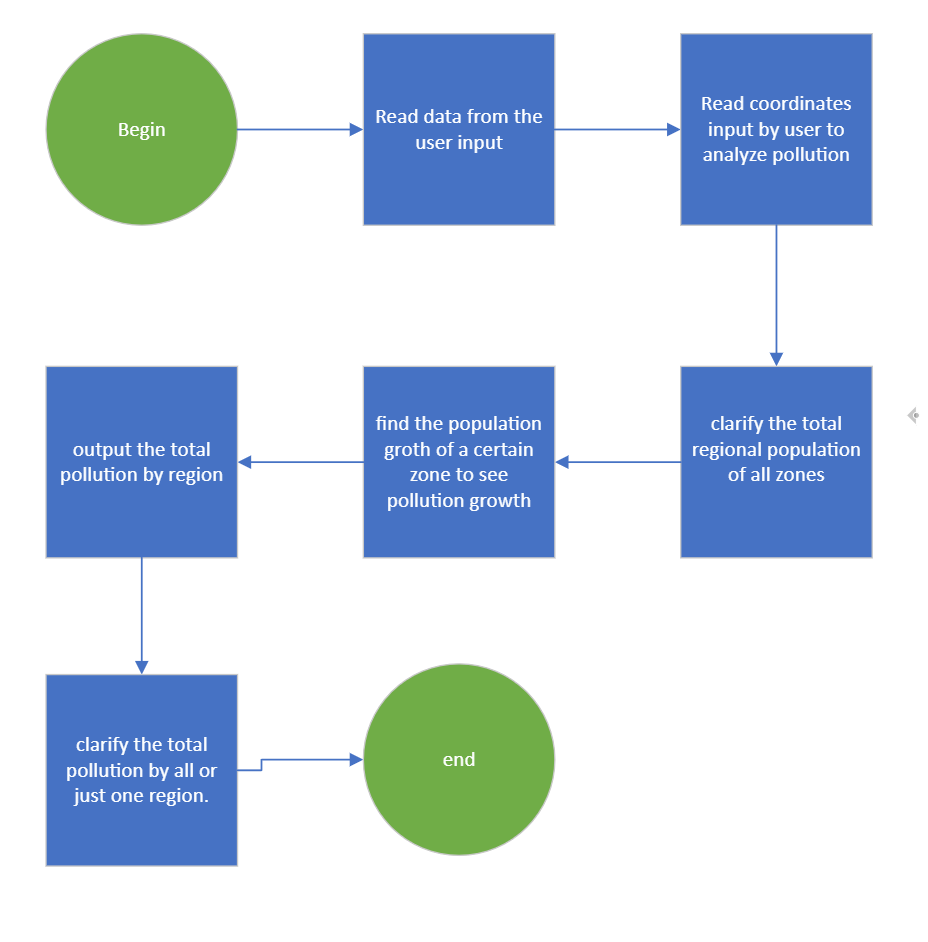
Description automatically generated

These diagrams shows that how the population for each zone is increasing. The classes created for each zone will be called and then it is going to check the conditions and at the end it is going to update each class.

Pollution:

Pollution is currently produced only by the Industrial zone; therefore, the industrial region class will contain the implementation for pollution functionality. After each time step, the process of pollution will be handled by the industrial zone “spreading” pollution to its neighbors. Depending on the region location, the pollution may be higher or lower as the population may increase or decrease by one for an industrial zone. If the user chooses to analyze regions for pollution in the decision menu, the console must properly output the correct data for such.

(shown below is the design for outputting the pollution data)



Rules for Pollution:

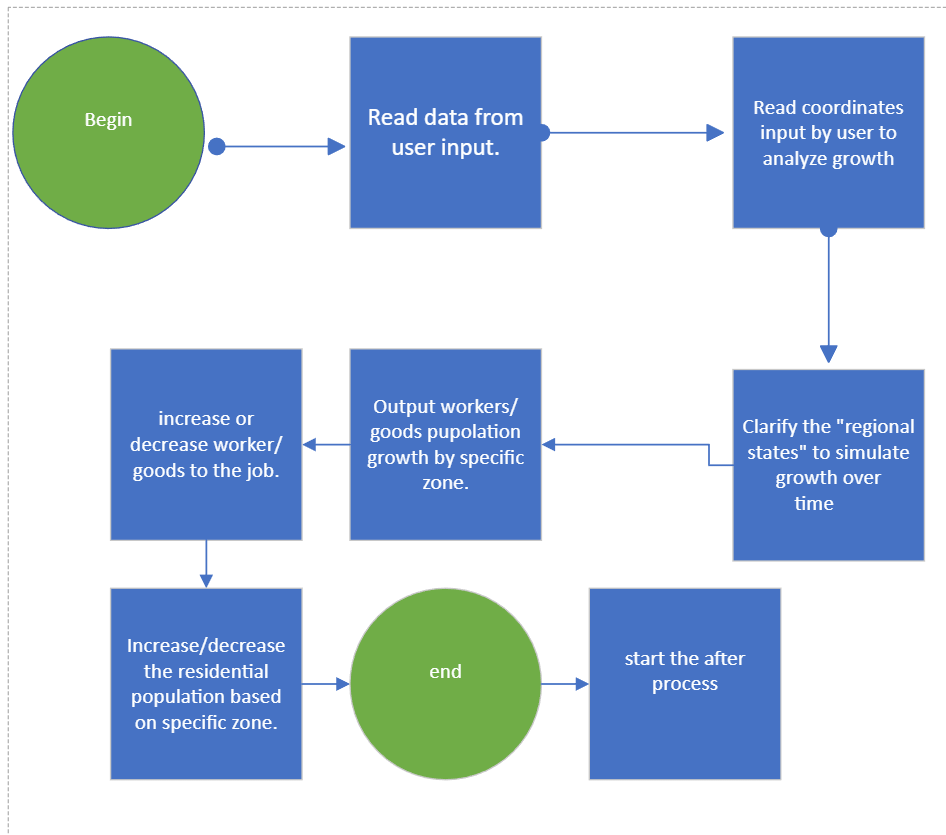
Diagram

Description automatically generated

Worker and Goods :

Available people and available goods are required to have an origin of production as well as an assignment if applicable. There will be a Person and Goods class/struct to fit that need, implemented with pointers to indicate origin and assignment. All such states must be indicated during inspection of the simulation so that during analysis the user understands where a worker lives and works/where a good is produced and is sold.

(shown below is the design for outputting worker and goods data)



Rules for workers and goods as per zones:

1. Residential Class

Diagram

Description automatically generated

1. Industrial Class

Diagram

Description automatically generated

1. Commercial Class

Diagram

Description automatically generated