Laurence Bradshaw

DISEASE MODELING  
 FRAMEWORK

AQA A-Level Computer Science NEA Project

Table of Contents

[Analysis 4](#_Toc100130499)

[Description of the Problem 4](#_Toc100130500)

[The Current System and Interview 4](#_Toc100130501)

[User Requirements 5](#_Toc100130502)

[Acceptable Limitations 5](#_Toc100130503)

[Background Research 6](#_Toc100130504)

[Different types of models 6](#_Toc100130505)

[Effect of an environment 7](#_Toc100130506)

[Disease transmission rate 7](#_Toc100130507)

[Project Objectives 8](#_Toc100130508)

[Proposed Solution 8](#_Toc100130509)

[Initial Design 9](#_Toc100130510)

[Processes 9](#_Toc100130511)

[Flow Chart 10](#_Toc100130512)

[Documented Design 11](#_Toc100130513)

[Overall System Design 11](#_Toc100130514)

[Data Dictionary 11](#_Toc100130515)

[Entity Relationship Diagram 14](#_Toc100130516)

[Data Flow Diagram 16](#_Toc100130517)

[Framework Class Diagrams 16](#_Toc100130518)

[Algorithm Examples 17](#_Toc100130519)

[Algorithm for progressing time in the simulation 17](#_Toc100130520)

[Algorithm for calculating hosts moving between cities 18](#_Toc100130521)

[Algorithm for calculating new infections in an environment 19](#_Toc100130522)

[SQL Examples 20](#_Toc100130523)

[Query for getting required data about all the cities 20](#_Toc100130524)

[Query for getting required data about the environments in the city 21](#_Toc100130525)

[Query for inputting data for a day 21](#_Toc100130526)

[Query for getting the disease data for the simulation 22](#_Toc100130527)

[Query for getting the number of the next iteration 22](#_Toc100130528)

[Query for getting the number of hosts in a specified city 22](#_Toc100130529)

[Query for getting the amount of time the simulation will run for 22](#_Toc100130530)

[Query for getting the start date 23](#_Toc100130531)

[Technical Solution 24](#_Toc100130532)

[Framework Files 24](#_Toc100130533)

[main.py 24](#_Toc100130534)

[model.py 24](#_Toc100130535)

[host.py 26](#_Toc100130536)

[disease.py 28](#_Toc100130537)

[environment.py 30](#_Toc100130538)

[container.py 31](#_Toc100130539)

[preprocessor.py 34](#_Toc100130540)

[plotter.py 35](#_Toc100130541)

[databasehandler.py 35](#_Toc100130542)

[validation.py 36](#_Toc100130543)

[Country Level Implementation Files 38](#_Toc100130544)

[person.py 38](#_Toc100130545)

[disease.py 39](#_Toc100130546)

[place.py 40](#_Toc100130547)

[databasehandler\_country\_level\_implementation.py 48](#_Toc100130548)

[plotter\_country\_level\_implementation.py 52](#_Toc100130549)

[functionLib.py 53](#_Toc100130550)

[preprocessor\_country\_level\_implementation.py 56](#_Toc100130551)

[Unit Test Files 57](#_Toc100130552)

[city\_tests.py 57](#_Toc100130553)

[country\_tests.py 59](#_Toc100130554)

[functionLib\_tests.py 59](#_Toc100130555)

[validation\_tests.py 62](#_Toc100130556)

[Map Plotting Script 70](#_Toc100130557)

[Map\_plotter.py 70](#_Toc100130558)

[Testing 74](#_Toc100130559)

[Tables Of Tests 74](#_Toc100130560)

[Validation tests 74](#_Toc100130561)

[Validation unit tests passing 80](#_Toc100130562)

[Function library tests 80](#_Toc100130563)

[Function library unit tests passing 81](#_Toc100130564)

[Database handler tests 82](#_Toc100130565)

[Database handler pictures 83](#_Toc100130566)

[Testing of graphs produced 84](#_Toc100130567)

[Graph pictures 85](#_Toc100130568)

[Evaluation 91](#_Toc100130569)

[Evaluation against project objectives 91](#_Toc100130570)

[Client Feedback 92](#_Toc100130571)

[List of Improvements 93](#_Toc100130572)

# Analysis

## Description of the Problem

The client for the project is Dr. Catherine Bradshaw, my mother and a climate scientist at the Met Office­­. Dr Bradshaw works in a team that focuses on the role of climate in food security and has recently expanded her research into crop pests and diseases.

Dr. Bradshaw has made a model for simulating the spread of wheat stem rust to determine if it is possible for stem rust to spread from Kenya to Iraq in a year. Since expanding her research, she is investigating other diseases, but it would require significant time and effort to rework the existing system into a model for a different disease. She would therefore like a framework for creating a system which can be used across the Met Office to model any disease affecting human and ecosystem health.

## The Current System and Interview

I conducted an interview with Dr. Bradshaw in order to obtain information about the current system and document user requirements for my project.

*Q. What’s the current system for your model?*

A. I have written some code in python to simulate the spread of wheat stem rust across locations in eastern Africa and the Middle East. It uses output from an atmospheric dispersion model to simulate the spread of wheat rust spores from infected locations and combines it with output from an environmental suitability model that simulates the infection chance.

*Q. What’s the issue with the current system?*

A. The current system has been designed with simulating the spread of wheat stem rust in mind so wheat stem rust is the only disease the model can simulate. This means the system would require significant time and effort to rework into a system for modelling a different disease.

*Q. What are the inputs and outputs for the current system?*

A. I preprocess the atmospheric dispersion model data and the environmental suitability model data into CSV files. Preprocessing takes a bit of time, but it means the model itself runs faster so I can run it many times with different scenarios being tested. The inputs I provide to each simulation are essentially lists of potential locations that could be infected with parameters about where they are, and the time of year that they are susceptible, when to start and stop the simulation and a title for the simulation and I also provide the maximum number of fungal spores that are going to be used because that gets used in the infection chance calculation. The system outputs a PNG file showing a map of all the locations and their infection status each timestep.

*Q. How do you use this output data?*

A. I generally combine the PNG files together into an animated GIF or an MP4 file to visualize how the disease has spread. The specific question I am trying to answer with this model is whether it is possible for wheat stem rust to spread from Kenya to Iraq in one year. I am writing a paper to summarize the findings and I look at the maps to determine the date which different locations have been infected and write about it.

*Q. So you look through each picture to find the date that the location is infected? It sounds like a* *database to store this information would be useful.*

A. Yes, I think that would be very useful.

*Q. What are some other features that you would like to have that the current system does not?*

A. It would be useful to have a graph that shows how many of all the locations are infected at each timestep and it would be useful if the model was more reusable. The current system is a bespoke stand-alone implementation for wheat stem rust meaning that it can’t be easily used to simulate a different disease without substantial rewriting.

*Q. How long does the current system take to run and do you run it on your personal computer or* *one of the Met Office super computers?*

A. It can run on either but it’s faster on the supercomputer and I can run more than two simulations at a time, but I have also run it on my personal computer because the queue for the supercomputer is too long for me to wait. It takes around two and a half hours for me to run one simulation of two years on the supercomputer and I have not timed it on my personal computer.

*Q. Did you use object-oriented programming?*

A. Kind of, a little but I am not a software engineer, and I am sure that my model could be improved from that point of view to make it more reusable and modular.

## User Requirements

* Tool must be in python.
* Be able to operate at different spatial scales.
* Populate initial conditions as entered by the user for a real-world scenario.
* Store disease transmission statistics through time and space in a database.
* Plot a graph showing disease transmission statistics through time and space.

## Acceptable Limitations

* The end model used at the Met Office will incorporate real weather data to change the transmission rate of the disease, but this data comes from a different model so the framework developed in this project will not incorporate the weather.
* Disease vector location data is computed by a different model which uses weather data and therefore doesn’t need to be computed as part of my framework.

## Background Research

The spread of a disease has multiple components to it: a host (the entity that gets infected), a vector (the thing that transfers the disease to the host), and an environment (the place where the hosts reside).

### Different types of models

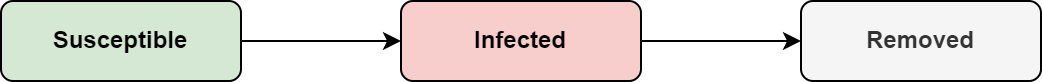
There are several different types of models that vary based on the disease, host, and vector.

The first model is for diseases which do not lead to immunity, it is called an S.I.S model because hosts only have two states: susceptible and infected. This means that the hosts return to the susceptible state after infection. This model would be used on diseases like the common cold, one which you can get many times. This model assumes that hosts and vectors are the same entity.

Shape

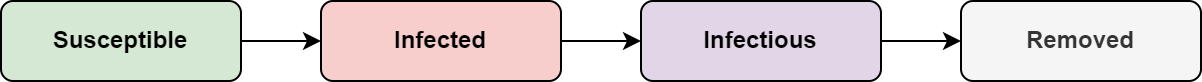
Description automatically generated with medium confidence

The second model is an S.I.R model where a host has three states: susceptible, infected, or removed. It is like the SIS model but once a host has been infected it has a removed state where it can no longer be infected by that disease again. This model type would be used to model some disease like chicken pox, one which you only get once in your life. Again this model assumes that hosts and vectors are the same entity.

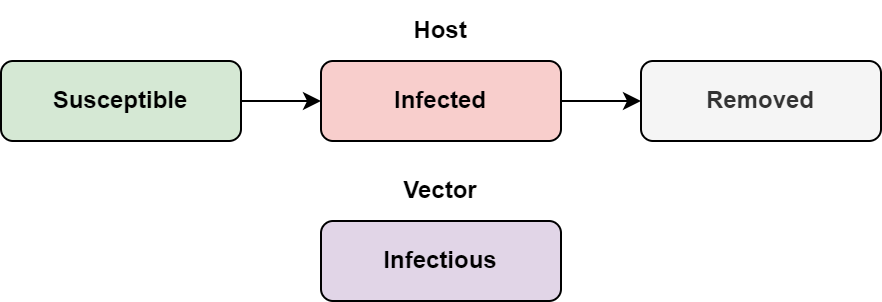


The third is an S.I.I.R model where the vector has an infected but not infectious state, where the disease is incubating; this becomes a little more complex if the vector and the host are the not the same entity.

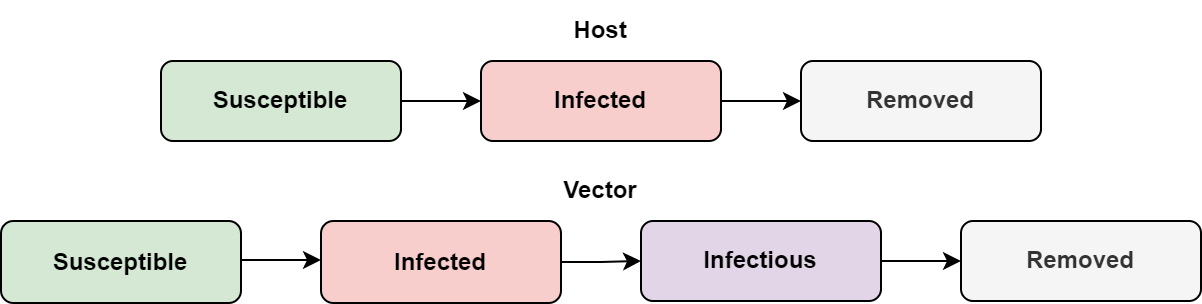
If the host and the vector are the same entity:



If the host and the vector are not the same entity and the vector is always infectious:



If the host and the vector are not the same entity and the vectors can infect each other: (essentially two hosts)



### Effect of an environment

An environment is the place where the hosts and vectors mix to transfer the disease. Environments also vary in the number of hosts that they can contain. An environment with a large population will have a higher chance of spreading the disease to others in that environment. The type of environment will affect the rate of infection. For example, a family infected with the common cold could enter both school and office environments, but there will be a higher rate of transmission of the cold in the school where children have a lot of contact with each other, than in the office where contact is much lower.

Environments would also have certain times at which it can infect the hosts. For example, a crop field would only be able to spread the infection if there was crop in the field or flowers/spores are open, likewise in an office, it would only be able to spread the disease if it has people in it, so not a weekend.

### Disease transmission rate

The measure for how transferable a disease is, the basic reproduction number (R0), is the average number of new infections that a single infection will cause over their infectious period:

It + y = It \* R0

Where y is the duration a host has the disease for. If R0 ­> 1, the disease will spread because each infected host on average infects more than one other host; if R0 ­< 1, then the disease will die out because each infected host infects less than one host. If R0 ­= 1 then the disease is said to be endemic, i.e. it is so widespread amongst a population that the population no longer requires infection from an external population to sustain it. R0 ­ assumes every host is susceptible and because this quickly becomes untrue so you can multiply R0 ­by the proportion of the population that is susceptible (S) you get the reproduction number for the susceptible population (R or Rt). To eliminate a disease from a population, R needs to be less than 1.

Herd immunity occurs when a significant proportion of the population is immune to the disease through vaccination or from already having contracted the disease. The herd immunity threshold is the proportion of a population that must be immune for R to equal 1:

HIT = 1 – 1/R0

## Project Objectives

* Simulate the spread of a disease through a population where the hosts are also the vectors.
* Create base classes for inheritance for the framework.
* Read simulation data to and from a database.
* Control what environments are run for a given simulation with the database.
* Control the properties of an environment with the database.
* Plot a graph to show infections and immunities at the end of the model run.
* Accommodate for scenarios where the spatial axis is different (simulate a whole country vs simulate just a city)
* EXTRA TIME: Plot and animate a choropleth map to show infections and their locations at the end of the model run
* EXTRA TIME: Accommodate for scenarios where the host and the vectors are different entities

## Proposed Solution

The solution to Dr. Bradshaw’s problem that I have decided on will be to create a framework of abstract classes and a main model script that will only be dependent on those abstract classes in order to create a model. This would mean that the user, if they so desire, can create their own classes for the model to run, and can mix and match components such as different plotting scripts or preprocessors to set the environments up in a different way. This also mean the user is able to specify the spatial and temporal scale as they need along with the type of model (SIS, SIR, etc.)

The model I make from the abstract classes will be able to model a disease with different parameters and allow processing to take place at different spatial scales. The timestep for the model will be daily. The preprocessing step will involve the system retrieving data from a database and creating a level 1 spatial scale environment class, which then contains level 2 spatial environment scale classes for the specified simulation. Each level 2 environment class, when made, will either hold a higher level of spatial scale environment or host objects. This means that the user can create the spatial scale to their specification when defining the environments.

After preprocessing, the model will begin to simulate the disease spread, it will tell the level 1 environment class to simulate a day, this will tell the level 2 environments to simulate a day and they will tell the level 3 environments to simulate a day and so on until the environment with the hosts has been reached. After the level 1 environment has simulated a day, it will output the statistics for that day to the database. Once this process has been repeated for the specified amount of time, it will plot graphs of infections and immunities against time for each of the level 2 environments to show the spread of the disease across time.

For my example implementation of the framework, I will demonstrate an application of simulating the spread of covid in England where the level 1 environment class will be a country-scale, the level 2 environment classes will be city-scale, and the level 3 environment classes will be buildings. The framework enables other environment types to be developed into a model, e.g. level 1 being a county, level 2 being farms and level 3 being fields.

**The following initial design and documented design have been designed for a country level implementation of the framework.**

## Initial Design

### Processes

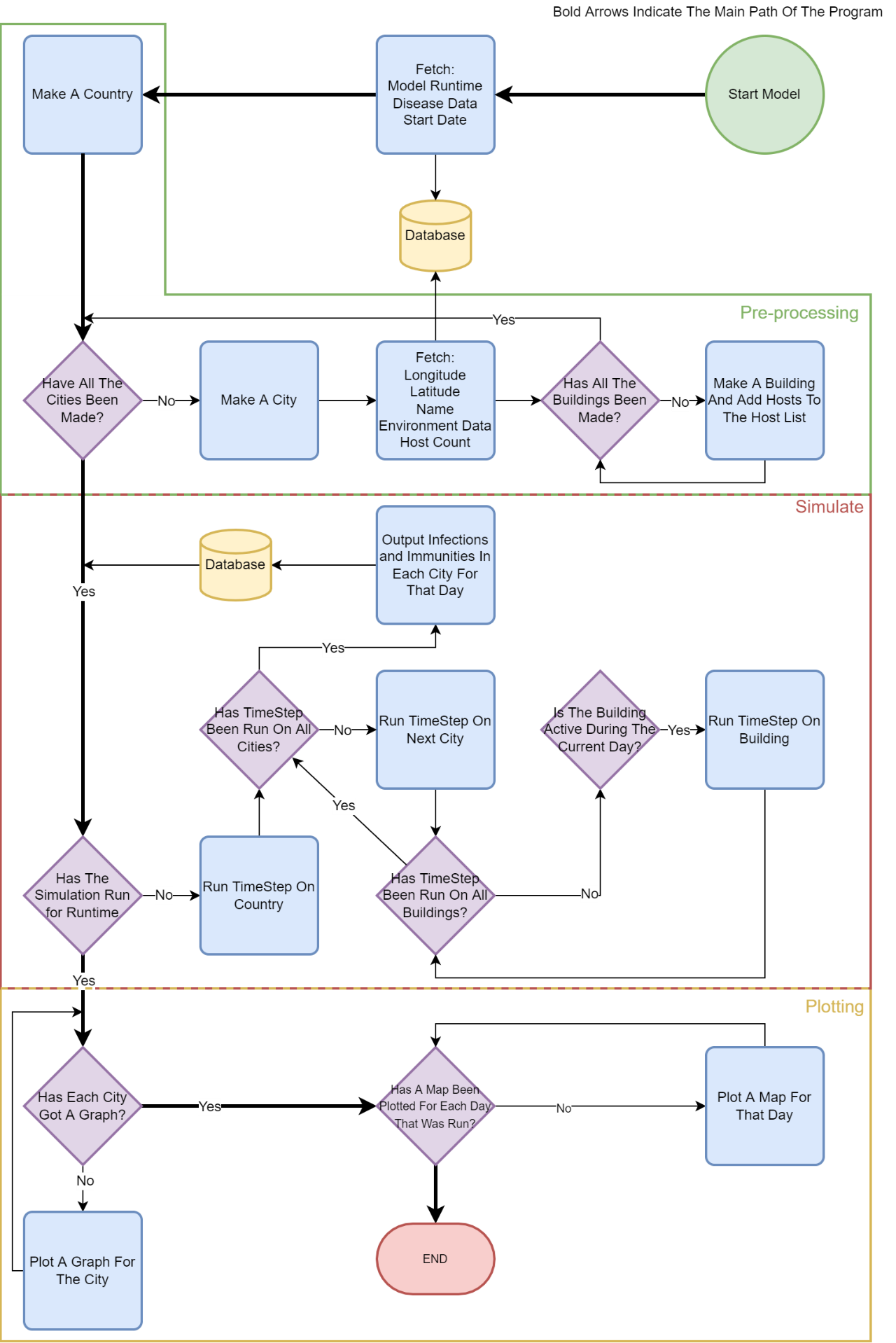
The model will have three phases: Pre-Processing, Simulating and Plotting.

The Pre-Processing phase is where fetching data from the database will occur and all the required objects for the model to run are created. Simulating is the phase where the model starts to simulate the spread of the disease. The Plotting phase will create graphs for the number of infections and immunities in each level 2 spatial scale environment against time and will also plot a map for each day to show the spread of the disease through the level 1 spatial scale environment.

Graphical user interface, text, application

Description automatically generated

### Flow Chart



# Documented Design

## Overall System Design

After taking the proposed design to Dr. Bradshaw she gave some pointers of things to change.

The database that my implementation uses doesn’t need to accommodate for other scenarios that potential users might need because it is a framework. The user needs to make their own implementation of the abstract classes for their own model if the ones my implementation uses do not suit their needs. This is the same for plotting of the data, the user must create their own plotting class to plot the data how they want.

As discussed in the proposed design, the system will have different spatial scale environments. These environments will either contain a higher level environment or the host objects. For example, the simulation could contain countries which then contain multiple cities which in turn contain multiple buildings which then contain the hosts. This means that I only need two environment type classes: a host environment and a container.

During Pre-processing, system will retrieve all the required data from the database to create and set up all the classes before it can begin simulating. After Pre-processing is complete the system will enter the Simulating phase where it begins to simulate the disease. Each container and environment will have a function to simulate time. Calling this function on the level 1 environment (a country in my case) will call the equivalent function on all the objects within that container until the environment with the hosts is reached where the actual infections will take place. After one of these cycles has been completed, the system will write an output to the database for that day. This will repeat until the system has run as many days as the user specified for in the database. Once the specified number of days has been reached, the system will enter the Plotting phase and create a series of graphs for the simulation data that was generated.

The test implementation of the framework is going to be an SIIR model at the country level where people are both the hosts and vectors. There will be houses, offices, shops, and schools as environments for the people. Environments such as offices and schools will only be counted in the simulation on weekdays to account for days where there are no people in the environment. The pre-processing stage will create a country as the level 1 spatial scale environment which contains cities which contains buildings which contains the hosts. The database made will only be configured for the test implementation. For modeling people moving between cities, a halfway house will be used. This will be used to model the people those traveling from city A to B will meet in city B and vice versa.

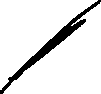
## Data Dictionary

*This is the data dictionary that the database for a model such as coivd will use when modelling at the country scale. For users who wish to model a disease at a different spatial scale, they must provide their own database.*

Table Simulation

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| SimulationConfiguration | Integer | An identification number for the simulation configuration | 1 |
| Disease | String | The disease to be modelled in the simulation configuration | Covid |
| RunTime | Integer | The length to run the simulation for in days | 200 |
| StartDate | Date/Time | The date at which the simulation will start | 1/12/2003 |

Table CityEnvironments



|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| CityID | Varchar(255) | The identifier for the city in the simulation | Exeter |
| EnvironmentID | Integer | The identifier for the environments that the city contains | House |
| Count | Integer | The number of that environment type the city contains | 51500 |
| LowerBound | Integer | The lower bound for the number of hosts to be present in the environment | 1 |
| UpperBound | Integer | The upper bound of hosts to be present in the environment | 6 |
| Average | Integer | The average number of hosts to be present in the environment | 3 |

Table SimulationCities

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| SimulationConfiguration | Integer | The identifier for the simulation configuration | 1 |
| CityID | Varchar(255) | The identifier for the city to be present in this configuration | Exeter |

Table Output

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| Iteration | Integer | Identification for the simulation number | 2 |
| TimeElapsed | Integer | The amount of time that has passed in the simulation in days. | 200 |
| SimulationConfiguration | Integer | The identifier for the simulation configuration | 1 |
| CityID | Varchar(255) | The identifier for the city | Exeter |
| InfectedHosts | Integer | The total number of infected hosts for that time in the city | 5000 |
| ImmuneHosts | Integer | The total number of immune hosts for that time in the city | 5000 |

Table Environment

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| Type | Varchar(255) | The name for that type of environment | House |
| ActivePeriod | Varchar(255) | The days/months that the environment can transfer the disease | Everyday/Weekday |
| InteractionRate | Float | A measure for the amount that hosts interact in an environment | 0.8 |

Table Disease

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| Name | Varchar(255) | Identification for the disease type | Common Cold |
| Duration | Integer | The amount of time a host will be infected with this disease for in days. | 14 |
| LatencyPeriod | Integer | The amount of time for a host to incubate the disease and then be infectious in days. | 2 |
| InfectionChance | Float | The chance of an infected person transmitting the disease | 0.103 |
| ImmuneProbability | Integer | The probability of being immune to the disease after being infected | 90 |
| ImmuneDuration | Integer | The amount of time a host will be immune for in days. | 30 |

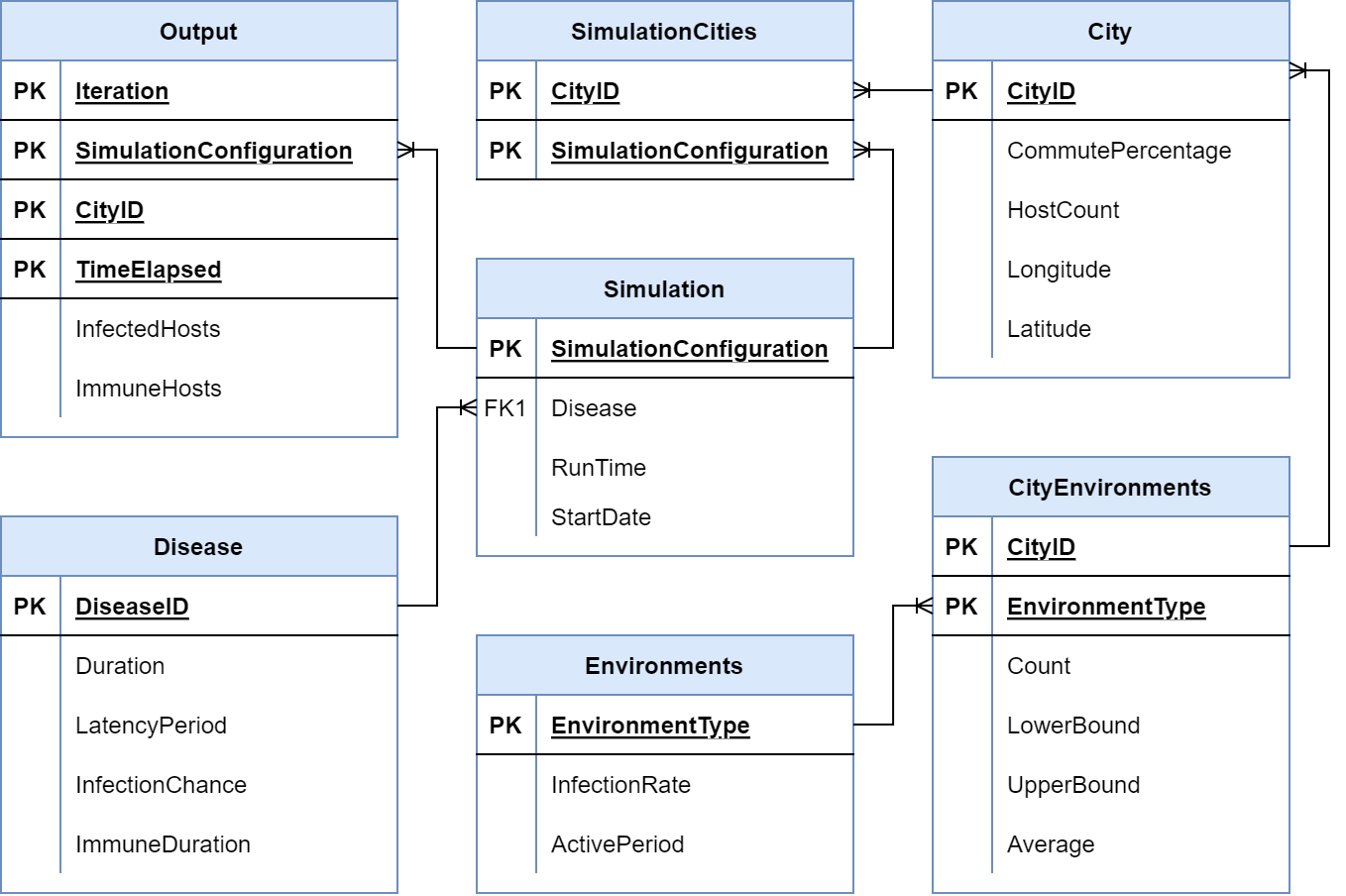
Table City

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example |
| CityID | Varchar(255) | The identifier for the city | Exeter |
| HostCount | Integer | The number of hosts that the city contains | 128000 |
| Longitude | Float | The longitude of the city | 50.0253 |
| Latitude | Float | The latitude of the city | 4.3443 |

## Entity Relationship Diagram

T*his is the Entity Relationship Diagram for the database defined above. For users who wish to model a disease at a different spatial scale, they must provide their own database.*

*(FK) -> Foreign Key. (PK) -> Primary Key.*



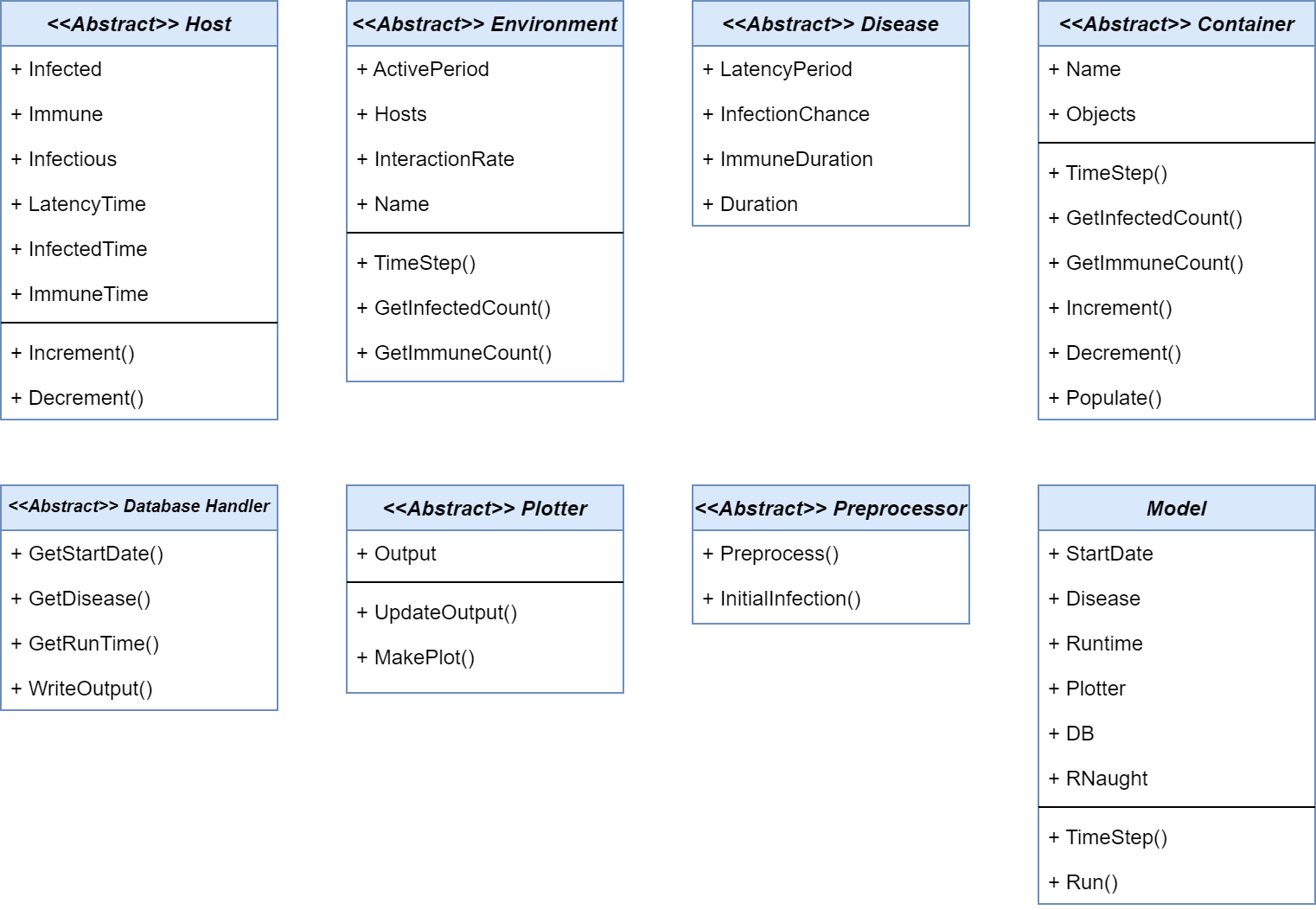
## Data Flow Diagram

A screen shot of a computer

Description automatically generated with low confidence

## Framework Class Diagrams

*These are the classes for the framework part of the project.*



## Algorithm Examples

### Algorithm for progressing time in the simulation

**Explanation:**

This algorithm simulates timesteps on the country for the requested amount of run time, updates the data in the plotter for plotting a graph at the end of the run time and then plots the graph at the end of the run time. Outputting to the database happens in the TimeStep function, not here.

**Plain English:**

Set i to 1.

While i is less than the RunTime

Get the date

Run time on the country

Add data from that TimeStep to the plotter for plotting

Increment i

Make the final plot

**Pseudocode:**

*Runtime is the amount of time that the simulation will run for. The function GetDate calculates the current day that is being simulated from the amount of time passed and the start date. Plotter is a class used for plotting the data, the UpdateOutput function on the plotter updates the data for outputting and the MakePlot function makes the final plot at the end of the simulation. The TimeStep function simulates a day in the country. Outputting to the database happens in the TimeStep function*

i ← 1

WHILE i <= Runtime

Date ← GetDate(I)

TimeStep(Country, Date, i)

Plotter.UpdateOutput(Date, Country)

i ← i + 1

ENDWHILE

Plotter.MakePlot()

### Algorithm for calculating hosts moving between cities

**Explanation:**

This algorithm will iterate over the top half of a matrix containing buildings and select hosts from two cities, based off a percentage calculated from the reciprocal of the distance between the two cities, and will add the two groups to the building to simulate hosts moving from city A to city B and vice versa.

**Plain English:**

Get the percentage of the population that will be traveling from city A to city B

Get the percentage of the population that will be traveling from city B to city A

For each row in the matrix

For each column in the top half of the matrix

Get and add the commuters from both cities to the halfway houses

**Pseudocode:**

*PercentageMatrix is a matrix of the percentages of a population to take. The getCommuters function returns the host objects that are going to travel to the other city. HalfwayHouses is a matrix of environments that will hold the hosts traveling to the other city.*

FOR row IN RANGE(LEN(HalfwayHouses[0]) - 1)

For col IN RANGE(row + 1, LEN(HalfwayHouses[0]))

Percentage1 ← PercentageMatrix[row, col]

Percentage2 ← percentageMatrix[col, row]

HalfwayHouses[row, col].hosts += city[row].getCommuters(Percentage1)

HalfwayHouses[row, col].hosts += city[col].getCommuters(Percentage2)

ENDFOR

ENDFOR

### Algorithm for calculating new infections in an environment

**Explanation**:

The Algorithm will take a list of all the hosts in an environment. It will then split the list up into two lists: one for infected people, one for uninfected people. The number of interactions is generated from the environment interaction rate. A sample of uninfected hosts is selected from the number of interactions. Then for each infected host who is infectious they will generate a pseudo-random number person between 0 and 100 for each uninfected, if this number is below the disease infection chance then that uninfected host becomes infected. This repeats for all infected people in that environment.

**Plain** **English**:

Get all uninfected hosts in the environment.

Get all infected hosts in the environment.

Calculate the number of interactions the hosts will have in this environment based on the environment infection rate

For each infected person repeat the steps below:

For each uninfected person in the environment create a random number

If this number is below the disease infection chance then that uninfected host is infected

**Pseudocode**:

*The generatePoisson function returns a number from a poisson distribution based on the input variable. InteractionRate is a number to represent how much hosts* *mix in an environment. Random.sample function returns the second arguments worth of elements from the first arguments list. hostList is a list of all the hosts in the environment.*

UninfectedHostList ← EMPTYLIST

FOR host IN hostList

IF host NOT infected AND NOT immune THEN

UninfectedHostList ← host

ENDIF

ENDFOR

InfectedHostList ← EMPTYLIST

FOR host IN hostList

IF host infected THEN

InfectedHostList ← person

ENDIF

ENDFOR

Interatctions ← min(round(generatePoisson(InfectionRate), LEN(UninfectedPeopleList))

FOR host IN random.sample(UninfectedPeopleList, Interactions)

IF RANDOM\_INT(0,100) < (disease.infectionChance \*100) THEN

host.infected ← TRUE

END IF

ENDFOR

## SQL Examples

*SQL queries in the framework do not require protection against SQL injection because there is no sensitive data stored in the database and only the user will have access to it because it is stored locally on the machine.*

### Query for getting required data about all the cities

**Explanation:**

Will select the CityID, longitude, latitude and commutePercentage for all cities in the simulation that is run.

**SQL:**

*Config is the configuration the simulation is running.*

“SELECT City.CityID, Longitude, Latitude, CommutePercentage FROM SimulationCities INNER JOIN City ON SimulationCities.CityID = City.CityID WHERE SimulationConfiguration = {config}”

### Query for getting required data about the environments in the city

**Explanation:**

Will select the environment type, number of them, the average, lower and upper bound of hosts, the active period and interaction rate of the environment type.

**SQL:**

*CityName is the name of the city the environments are being fetched for.*

“SELECT CityEnvironments.EnvironmentType, Count, LowerBound, UpperBound, Average, ActivePeriod, InteractionRate FROM CityEnvironments INNER JOIN Environments ON CityEnvironments.EnvironmentType = Environments.EnvironmentType WHERE CityEnvironments.CityID = ‘{CityName}’”

### Query for inputting data for a day

**Explanation:**

Will insert all the required information into the database after a day has been simulated.

**SQL:**

*Iteration is the number of times the config has been run.*

*Config is the configuration the simulation is running.*

*CityName is the name of the city the data is for.*

*Time is the amount of time that has passed in the simulation at this point.*

*InfectedCount is the number of infected hosts at this point in time.*

*ImmuneCount is the number of immune hosts at this point in time.*

“INSERT INTO Output (Iteration, SimulationConfiguration, CityID, TimeElapsed, InfectedHosts, ImmuneHosts)”

“VALUES ({Iteration}, {Config}, {CityName}, {Time}, {InfectedCount}, {ImmuneCount})”

### Query for getting the disease data for the simulation

**Explanation:**

Will select the duration, LatencyPeriod, InfectionChance and ImmuneDuration for the simulation configuration.

**SQL:**

*Config is the configuration the simulation is running.*

“SELECT Duration, LatencyPeriod, InfectionChance, ImmuneDuration FROM Disease INNER JOIN Simulation ON Disease.DiseaseID = Simulation.Disease WHERE SimulationConfiguration = {Config}”

### Query for getting the number of the next iteration

**Explanation:**

Will select the current max iteration from the output table and add one to it to find the iteration of the current run.

**SQL:**

*Config is the configuration the simulation is running.*

“SELECT MAX(Iteration) + 1 FROM Output WHERE TimeElapsed = 1 AND SimulationConfiguration = {Config}”

### Query for getting the number of hosts in a specified city

**Explanation:**

Will select the host count that the specified city will have.

**SQL:**

*CityName is the name of the city that is specified.*

“SELECT HostCount FROM City WHERE CityID = {CityName}”

### Query for getting the amount of time the simulation will run for

**Explanation:**

Will return the amount of time the simulation was requested to run for.

**SQL:**

*Config is the configuration the simulation is running.*

“SELECT RunTime FROM Simulation WHERE SimulationConfiguration = {Config}”

### Query for getting the start date

**Explanation:**

Will select the date at which the simulation will start.

**SQL:**

*Config is the configuration the simulation is running.*

*“SELECT StartDate FROM Simulation WHERE SimulationConfiguration = {Config}”*

# Technical Solution

## Framework Files

*The following files contain the code for the framework part of the program. Main.py is the entry point to the model.*

### main.py

import disease  
import model  
import databasehandler\_country\_level\_implementation as db  
import plotter\_country\_level\_implementation as plt  
import preprocessor\_country\_level\_implementation as pre  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 configurationNumber = 3  
  
 # Makes a connection to the database  
 dbHandler = db.DatabaseHandler(configurationNumber)  
 # Gets the starting date for the simulation  
 startDate = dbHandler.getStartDate()  
 # Gets the amount of time that the simulation will run for  
 runtime = dbHandler.getRuntime()  
 # Makes the disease that the simulation will model  
 disease = disease.Disease(dbHandler.getDisease())  
 # Makes a plotter  
 pltr = plt.Plotter(dbHandler)  
 # Makes a preprocessor  
 preprocess = pre.Preprocessing(dbHandler)  
  
 # Makes the model  
 model = model.Model(disease, runtime, startDate, dbHandler, pltr, preprocess)  
 # Runs the model  
 model.run()

### model.py

from datetime import datetime, date, timedelta  
import time  
import validation  
# To import custom parts replace the name after import with the custom part e.g. import preprocessor\_common\_cold as preprocessing for a script that preprocesses for the common cold  
  
  
class Model(object):  
  
 def \_\_init\_\_(*self*, disease, runtime, startDate, db, pltr, preprocess):  
 *"""  
 Constructor for the model class  
 makes and sets all the necessary details for the model to run* ***:param*** *disease: The disease class that the model will run for (disease)* ***:param*** *runtime: The amount of time for the model to run for (int)* ***:param*** *startDate: The date at which the model will start at (list) [day, month, year]* ***:param*** *db: The class which handles all actions with the database (databaseHandler)  
 """* validation.isInt(runtime)  
 validation.isDatabaseHandler(db)  
 validation.isList(startDate)  
 validation.isDisease(disease)  
  
 *self*.startDate = date(startDate[2], startDate[1], startDate[0])  
 *self*.disease = disease  
 *self*.runtime = runtime  
 *self*.plotter = pltr  
 *self*.preprocessor = preprocess  
 *self*.db = db  
 *self*.rNaughtList = []  
  
 def timeStep(*self*, topLevel, res, i):  
 *"""  
 Simulates a day in the model* ***:param*** *topLevel: The container that contains all other containers (container)* ***:param*** *res: The date (string)* ***:param*** *i: The amount of time that the simulation has run for (int)  
 """* # Converts the date into a day of the week  
 dayName = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']  
 day = datetime.strptime(res, '%d-%m-%Y').weekday()  
 # Counts the number of hosts infected before the day is simulated  
 infected = 0  
 prevInfected = 0  
 prevInfected += topLevel.getInfectedCount()  
 # Simulates the day  
 topLevel.timeStep(*self*.disease, dayName[day])  
 # Counts the number of infections after the day has been simulated  
 infected += topLevel.getInfectedCount()  
  
 # Makes an output to the database for each object contained in the top level container  
 for o in topLevel.objects:  
 imm = o.getImmuneCount()  
 inf = o.getInfectedCount()  
 *self*.db.writeOutput(o.name, i, inf, imm)  
 print("{} - {}, {}".format(o.name, inf, imm))  
  
 # Calculates the R number for that day  
 try:  
 rNaught = infected/prevInfected  
 except ZeroDivisionError:  
 rNaught = 0.0  
  
 *self*.rNaughtList.append(rNaught)  
 print("{} - {} ({})/({}) = ({})".format(i, dayName[day], infected, prevInfected, rNaught))  
  
 def run(*self*):  
 *"""  
 Starts the model  
 """* startTime = time.time()  
 # Preprocessing returns one container  
 topLevel = *self*.preprocessor.preprocess(*self*.disease)  
 validation.isContainer(topLevel)  
 # Starts simulating  
 i = 1  
 # While the model should still be simulating  
 while i <= *self*.runtime:  
 # Converts 'i' into a real date from the starting date  
 resDate = *self*.startDate + timedelta(days=i - 1)  
 res = resDate.strftime("%d-%m-%Y")  
 # Tells the top level container to simulate a day  
 *self*.timeStep(topLevel, res, i)  
 # Updates the output for plotting at the end of the models runtime  
 *self*.plotter.updateOutput(res, topLevel)  
 i += 1  
  
 totalRNaught = 0  
 for r in *self*.rNaughtList:  
 totalRNaught += r  
 print("Average R0 For This Simulation:", (totalRNaught/*self*.runtime))  
  
 # Makes the plotter makes the plots  
 *self*.plotter.makePlot()  
 # Would run the map plotting script here  
 print("Model Finished")  
 print("%s seconds" % (time.time() - startTime))

### host.py

from abc import ABC, abstractmethod  
  
  
class Host(ABC):  
 *"""  
 Abstract class for a host  
 The object that gets infected  
 """* def \_\_init\_\_(*self*):  
 *"""  
 The constructor for the host class  
 """  
 self*.\_infected = False  
 *self*.\_infectious = False  
 *self*.\_latencyTime = 0  
 *self*.\_infectedTime = 0  
 *self*.\_immune = False  
 *self*.\_immuneTime = 0  
  
 ##########  
 # Properties  
 ##########  
  
 @property  
 def immuneTime(*self*):  
 return *self*.\_immuneTime  
  
 @property  
 def infectious(*self*):  
 return *self*.\_infectious  
  
 @property  
 def latencyTime(*self*):  
 return *self*.\_latencyTime  
  
 @property  
 def infectedTime(*self*):  
 return *self*.\_infectedTime  
  
 @property  
 def immune(*self*):  
 return *self*.\_immune  
  
 @property  
 def infected(*self*):  
 return *self*.\_infected  
  
 ##########  
 # Property Setters  
 ##########  
  
 @immuneTime.setter  
 def immuneTime(*self*, value):  
 *self*.\_immuneTime = value  
  
 @infected.setter  
 def infected(*self*, value):  
 *self*.\_infected = value  
  
 @latencyTime.setter  
 def latencyTime(*self*, value):  
 *self*.\_latencyTime = value  
  
 @infectedTime.setter  
 def infectedTime(*self*, value):  
 *self*.\_infectedTime = value  
  
 @immune.setter  
 def immune(*self*, value):  
 *self*.\_immune = value  
  
 @infectious.setter  
 def infectious(*self*, value):  
 *self*.\_infectious = value  
  
 ##########  
 # Methods  
 ##########  
  
 @abstractmethod  
 def increment(*self*, disease):  
 *"""  
 Increments values on the host* ***:param*** *disease: The disease class that the model will run for (disease)  
 """* pass  
  
 @abstractmethod  
 def decrement(*self*, disease):  
 *"""  
 Decrements values on the host* ***:param*** *disease: The disease class that the model will run for (disease)  
 """* pass

### disease.py

from abc import ABC, abstractmethod  
  
  
class Disease(ABC):  
 *"""  
 Abstract class for a disease  
 """* def \_\_init\_\_(*self*):  
 *"""  
 Constructor for the disease  
 """  
 self*.\_latencyPeriod = 0  
 *self*.\_infectionChance = 0  
 *self*.\_duration = 0  
 *self*.\_immuneDuration = 0  
  
 ##########  
 # Properties  
 ##########  
  
 @property  
 def immuneDuration(*self*):  
 return *self*.\_immuneDuration  
  
 @property  
 def latencyPeriod(*self*):  
 return *self*.\_latencyPeriod  
  
 @property  
 def infectionChance(*self*):  
 return *self*.\_infectionChance  
  
 @property  
 def duration(*self*):  
 return *self*.\_duration  
  
 ##########  
 # Property setters  
 #########  
  
 @immuneDuration.setter  
 def immuneDuration(*self*, value):  
 *self*.\_immuneDuration = value  
  
 @latencyPeriod.setter  
 def latencyPeriod(*self*, value):  
 *self*.\_latencyPeriod = value  
  
 @infectionChance.setter  
 def infectionChance(*self*, value):  
 *self*.\_infectionChance = value  
  
 @duration.setter  
 def duration(*self*, value):  
 *self*.\_duration = value

### environment.py

from abc import ABC, abstractmethod  
  
  
class Environment(ABC):  
 *"""  
 Abstract class for the place where the hosts are  
 """* def \_\_init\_\_(*self*, name, activePeriod, interactionRate):  
 *"""  
 Constructor for an environment* ***:param*** *name: Name for the environment (string)* ***:param*** *activePeriod: The days/times when the hosts are in the environment and can infect each other/be infected (list)* ***:param*** *interactionRate: The amount the hosts mix with one another (determines the number of interactions) (float)  
 """  
 self*.\_activePeriod = activePeriod  
 *self*.\_hosts = []  
 *self*.\_interactionRate = interactionRate  
 *self*.\_name = name  
  
 @property  
 def activePeriod(*self*):  
 return *self*.\_activePeriod  
  
 @property  
 def hosts(*self*):  
 return *self*.\_hosts  
  
 @property  
 def interactionRate(*self*):  
 return *self*.\_interactionRate  
  
 @property  
 def name(*self*):  
 return *self*.\_name  
  
 @activePeriod.setter  
 def activePeriod(*self*, value):  
 *self*.\_activePeriod = value  
  
 @hosts.setter  
 def hosts(*self*, value):  
 *self*.\_hosts = value  
  
 @interactionRate.setter  
 def interactionRate(*self*, value):  
 *self*.\_interactionRate = value  
  
 @name.setter  
 def name(*self*, value):  
 *self*.\_name = value  
  
 @abstractmethod  
 def timeStep(*self*, disease, day):  
 *"""  
 Simulates a day on the environment* ***:param*** *disease: The disease class that the model will run for (disease)* ***:param*** *day: The current day that is being simulated (string)  
 """* pass  
  
 @abstractmethod  
 def getInfectedCount(*self*):  
 *"""  
 Gets the number of infected hosts in the environment* ***:return****: The number of infected hosts (int)  
 """* count = 0  
 for h in *self*.hosts:  
 if h.infected:  
 count += 1  
 return count  
  
 @abstractmethod  
 def getImmuneCount(*self*):  
 *"""  
 Gets the number of immune hosts in the environment* ***:return****: The number of immune hosts (int)  
 """* count = 0  
 for h in *self*.hosts:  
 if h.immune:  
 count += 1  
 return count

### container.py

from abc import ABC, abstractmethod  
  
  
class Container(ABC):  
 *"""  
 Abstract class for containers  
 A container is used for holding objects  
 Examples:  
 City (contains environments)  
 Country (contains cities)  
  
 The preprocessing must return one top level container (not contained by anything itself)  
 """* def \_\_init\_\_(*self*, name):  
 *"""  
 Constructor for the container class* ***:param*** *name: Name of the container (string)  
 """  
 self*.\_objects = []  
 *self*.\_name = name  
  
 ##########  
 # Properties  
 ##########  
  
 @property  
 def objects(*self*):  
 return *self*.\_objects  
  
 @property  
 def name(*self*):  
 return *self*.\_name  
  
 ##########  
 # Property setters  
 ##########  
  
 @objects.setter  
 def objects(*self*, value):  
 *self*.\_objects = value  
  
 @name.setter  
 def name(*self*, value):  
 *self*.\_name = value  
  
 ##########  
 # Methods  
 ##########  
  
 @abstractmethod  
 def timeStep(*self*, disease, day):  
 *"""  
 Simulates a day on each object in the container* ***:param*** *disease: The disease that is running in the simulation* ***:param*** *day: That name of the day that is being run  
 """* for o in *self*.objects:  
 o.timeStep(disease, day)  
  
 @abstractmethod  
 def getInfectedCount(*self*):  
 *"""  
 Gets the number of infected hosts from all the objects in the container* ***:return****: The number of infected hosts as an int  
 """* count = 0  
 for o in *self*.objects:  
 count += o.getInfectedCount()  
 return count  
  
 @abstractmethod  
 def getImmuneCount(*self*):  
 *"""  
 Gets the number of immune ghosts from all the objects in the container* ***:return****: The number of infected hosts as an int  
 """* count = 0  
 for o in *self*.objects:  
 count += o.getImmuneCount()  
 return count  
  
 @abstractmethod  
 def increment(*self*, disease):  
 *"""  
 Tells the contained objects to increment time  
 Incrementing time increases the counter on a host* ***:param*** *disease: The disease that the simulation is running  
 """* for o in *self*.objects:  
 o.increment(disease)  
  
 @abstractmethod  
 def decrement(*self*, disease):  
 *"""  
 Tells the contained objects to decrement time  
 Decrementing time checks if a hosts has been infected for the disease duration and makes them not infected if so* ***:param*** *disease: The disease that the simulation is running  
 """* for o in *self*.objects:  
 o.decrement(disease)  
  
 @abstractmethod  
 def populate(*self*, db):  
 *"""  
 Will populate the container with objects* ***:param*** *db: The database handler class for accessing the database  
 """* pass

### preprocessor.py

from abc import ABC, abstractmethod  
  
  
class Preprocessing(ABC):  
 *"""  
 Abstract class for preprocessing  
 """* def \_\_init\_\_(*self*):  
 *"""  
 Constructor for preprocessing  
 """* pass  
  
 ##########  
 # Methods  
 ##########  
  
 @abstractmethod  
 def preprocess(*self*, disease):  
 *"""  
 Sets up the environments and hosts for the simulation* ***:param*** *disease: The disease class that the model will run for (disease)* ***:return****: One container type object to the model (container)  
 """* pass  
  
 @abstractmethod  
 def initialInfection(*self*, disease, topLevel):  
 *"""  
 Decides the initial infection* ***:param*** *disease: The disease class that the model will run for (disease)* ***:param*** *topLevel: The container that contains all other containers (container)  
 """* pass

### plotter.py

from abc import ABC, abstractmethod  
  
  
class Plotter(ABC):  
 *"""  
 Abstract class for plotting  
 """* def \_\_init\_\_(*self*):  
 *"""  
 Constructor for plotter  
 """  
 self*.\_output = None  
  
 ##########  
 # Methods  
 ##########  
  
 @abstractmethod  
 def updateOutput(*self*, i, topLevel):  
 *"""  
 Updates the output data for the end graph* ***:param*** *i: Time passed in the simulation (int)* ***:param*** *topLevel: The container that contains all other containers (container)  
 """* pass  
  
 @abstractmethod  
 def makePlot(*self*):  
 *"""  
 Makes the final plot  
 """* pass

### databasehandler.py

from abc import ABC, abstractmethod  
  
  
class DatabaseHandler(ABC):  
 *"""  
 Abstract class for accessing the database  
 """* def \_\_init\_\_(*self*):  
 pass  
  
 ##########  
 # Methods  
 ##########  
  
 @abstractmethod  
 def getStartDate(*self*):  
 pass  
  
 @abstractmethod  
 def getDisease(*self*):  
 pass  
  
 @abstractmethod  
 def getRuntime(*self*):  
 pass  
  
 @abstractmethod  
 def writeOutput(*self*, cityName, time, infectedCount, immuneCount):  
 pass

### validation.py

from Framework import host, databasehandler, container, disease  
import datetime  
  
  
class OutOfRange(Exception):  
 pass  
  
  
def generatePoissonRange(rate):  
 if not (0 < rate < 1):  
 raise OutOfRange('Interaction Rate Is Out Of The Expected Range. Make Sure It Is Between 0 And 1 (Not Equal To)')  
  
  
def weightedRandomRange(lb, ub, avg):  
 if avg <= lb:  
 raise OutOfRange('Average Is Less Than Or Equal To The Lower Bound')  
 if avg >= ub:  
 raise OutOfRange('Average Is Greater Than Or Equal To The Lower Bound')  
  
  
def coordRange(coord):  
 if not (-180 <= coord <= 180):  
 raise OutOfRange('Coordinate Is Out Of The Expected Range')  
  
  
def percentageRange(percentage):  
 if not (0 <= percentage <= 100):  
 raise OutOfRange('Percentage Is Out Of The Expected Range For A Percentage')  
  
  
def knownActivePeriod(activePeriod):  
 knownActivePeriods = ['Everyday', 'Weekdays']  
 if activePeriod not in knownActivePeriods:  
 raise ValueError('Not A Known Active Period')  
  
  
def isString(data):  
 if not isinstance(data, str):  
 raise TypeError('Expected ' + str(str) + ' But Got ' + str(type(data)))  
  
  
def isInt(data):  
 if not isinstance(data, int):  
 raise TypeError('Expected ' + str(int) + ' But Got ' + str(type(data)))  
  
  
def isFloat(data):  
 if not isinstance(data, float):  
 raise TypeError('Expected ' + str(float) + ' But Got ' + str(type(data)))  
  
  
def isIntOrFloat(data):  
 if not (isinstance(data, int) or isinstance(data, float)):  
 raise TypeError('Expected ' + str(float) + 'Or ' + str(int) + ' But Got ' + str(type(data)))  
  
  
def isNoneNegativeInt(data):  
 isInt(data)  
 if data < 0:  
 raise ValueError('Expected A Positive Int But Got A Negative One')  
  
  
def isNoneNegativeFloat(data):  
 isFloat(data)  
 if data < 0:  
 raise ValueError('Expected A Positive Float But Got A Negative One')  
  
  
def isNoneNegativeFloatOrInt(data):  
 isIntOrFloat(data)  
 if data < 0:  
 raise ValueError('Expected A Positive Float Or Int But Got A Negative One')  
  
  
def isList(data):  
 if not isinstance(data, list):  
 raise TypeError('Expected ' + str(list) + ' But Got ' + str(type(data)))  
  
  
def isDate(data):  
 if not isinstance(data, datetime.datetime):  
 raise TypeError('Expected ' + str(datetime.datetime) + ' But Got ' + str(type(data)))  
  
  
def isDatabaseHandler(db):  
 if not issubclass(type(db), databasehandler.DatabaseHandler):  
 raise TypeError('Expected A ' + str(databasehandler.DatabaseHandler) + ' Subclass')  
  
  
def isDisease(d):  
 if not issubclass(type(d), disease.Disease):  
 raise TypeError('Expected A ' + str(disease.Disease) + ' Subclass')  
  
  
def isHost(h):  
 if not issubclass(type(h), host.Host):  
 raise TypeError('Expected A ' + str(host.Host) + ' Subclass')  
  
  
def isContainer(c):  
 if not issubclass(type(c), container.Container):  
 raise TypeError('Expected A ' + str(container.Container) + ' But Got ' + str(type(c)))

## Country Level Implementation Files

### person.py

import functionLib  
from Framework import host  
  
  
class Person(host.Host):  
 *"""  
 Class for the host of the model  
 """* def \_\_init\_\_(*self*):  
 *"""  
 Constructor for the host class  
 """  
 self*.age = functionLib.weightedRandom(1, 100, 35)  
 super(Person, *self*).\_\_init\_\_()  
  
 def getAge(*self*):  
 *"""  
 Gets the age of the person* ***:return****: The age property of the host (int)  
 """* return *self*.age  
  
 def increment(*self*, disease):  
 *"""  
 Increments the time a person is infected for if applicable* ***:param*** *disease: The disease the simulation is running (disease)  
 """* # Increases time infected on people infected if they are infected  
 if *self*.infected and not *self*.immune:  
 *self*.latencyTime += 1  
 *self*.infectedTime += 1  
 # Checks if they have been infected for the duration of the disease  
 if *self*.latencyTime >= disease.latencyPeriod:  
 *self*.infectious = True  
 # Increments the amount of time a person has been immune for  
 if *self*.immune:  
 *self*.immuneTime += 1  
  
 def decrement(*self*, disease):  
 *"""  
 Removes the infected status from a person if they have been infected for the disease duration* ***:param*** *disease: The disease the simulation is running (disease)  
 """* # Removes infected status from people who have been infected for the disease duration  
 if *self*.infectedTime >= disease.duration:  
 *self*.infected = False  
 *self*.infectious = False  
 *self*.latencyTime = 0  
 *self*.infectedTime = 0  
 *self*.immune = True  
 # Removes immune status from people who have been immune for the disease immunity period  
 if *self*.immuneTime >= disease.immuneDuration:  
 *self*.immune = False  
 *self*.immuneTime = 0

### disease.py

from Framework import disease  
  
  
class Disease(disease.Disease):  
 *"""  
 Class for the disease  
 """* def \_\_init\_\_(*self*, diseaseDetails):  
 *"""  
 Constructor for the disease class* ***:param*** *diseaseDetails: Info from the database for the disease (pyodbc row)  
 """* super(Disease, *self*).\_\_init\_\_()  
 *self*.latencyPeriod = diseaseDetails['LatencyPeriod']  
 *self*.infectionChance = diseaseDetails['InfectionChance']  
 *self*.duration = diseaseDetails['Duration']  
 *self*.immuneDuration = diseaseDetails['ImmuneDuration']

### place.py

import random  
import functionLib  
from Framework import environment, container  
import time  
  
  
class Building(environment.Environment):  
 *"""  
 Building class will contain hosts and handle their interactions  
 """* def \_\_init\_\_(*self*, activePeriod, name, interactionRate, hostObjects):  
 *"""  
 Constructor for the building class* ***:param*** *activePeriod: List of days that the building will be active (input: string)* ***:param*** *name: Name of the building type (string)* ***:param*** *interactionRate: A measure of how much the hosts interact with each other (0 - 0.99)* ***:param*** *hostObjects: A lists of all the host objects that the building will contain (list)  
 """* super(Building, *self*).\_\_init\_\_(name, activePeriod, interactionRate)  
 # Adds the input host objects to the buildings host population  
 *self*.hosts = hostObjects  
 # Converts the input active period into the days it represents  
 if activePeriod == "Everyday":  
 *self*.activePeriod = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']  
 elif activePeriod == "Weekdays":  
 *self*.activePeriod = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']  
  
 def getImmuneCount(*self*):  
 *"""  
 Counts the number of immune hosts in the building* ***:return****: The number of immune hosts as an int  
 """* return super().getImmuneCount()  
  
 def getInfectedCount(*self*):  
 *"""  
 Counts the number of infected hosts in the building* ***:return****: The number of infected hosts as an int  
 """* return super().getInfectedCount()  
  
 def timeStep(*self*, disease, day):  
 *"""  
 Simulates a day progressing on the building* ***:param*** *disease: The disease the simulation is running (disease)* ***:param*** *day: The day that is currently being run (string)  
 """* # If the building should run given the day  
 if day in *self*.activePeriod:  
 # Gets the infected host objects  
 infectedHosts = *self*.getInfectedHosts()  
 # If there are infected hosts in the environment  
 if len(infectedHosts) != 0:  
 # Gets the uninfected host objects  
 uninfectedHosts = *self*.getUninfectedHosts()  
 # Foreach infected host  
 for \_ in infectedHosts:  
 # Calculate their interactions with the other hosts in the building  
 interactions = min(round(functionLib.generatePoisson(*self*.interactionRate)), len(uninfectedHosts))  
 # Foreach uninfected host that the infected host has come into contact with  
 for uip in random.sample(uninfectedHosts, interactions):  
 # See if they get infected  
 if random.randint(0, 100) < (disease.infectionChance \* 100):  
 uip.infected = True  
  
 def getInfectedHosts(*self*):  
 *"""  
 Gets the objects of the hosts which are infected* ***:return****: The objects of the infected hosts  
 """* infectedHosts = []  
 for h in *self*.hosts:  
 if h.infected:  
 infectedHosts.append(h)  
 return infectedHosts  
  
 def getUninfectedHosts(*self*):  
 *"""  
 Gets the objects of the hosts which are uninfected* ***:return****: The objects of the uninfected hosts  
 """* uninfectedHosts = []  
 for h in *self*.hosts:  
 if not h.infected and not h.immune:  
 uninfectedHosts.append(h)  
 return uninfectedHosts  
  
 def increment(*self*, disease):  
 *"""  
 Increment each host* ***:param*** *disease: The disease the simulation is running (disease)  
 """* for h in *self*.hosts:  
 h.increment(disease)  
  
 def decrement(*self*, disease):  
 *"""  
 Decrement each host* ***:param*** *disease: The disease the simulation is running (disease)  
 """* for h in *self*.hosts:  
 h.decrement(disease)  
  
  
class City(container.Container):  
 *"""  
 Class which inherits from Container  
 Contains environments  
 Handle interactions between the buildings  
 """* def \_\_init\_\_(*self*, name, long, lat, commutePercentage):  
 *"""  
 Constructor for the city class* ***:param*** *name: Name of the city (string)* ***:param*** *long: Longitude of the city (float)* ***:param*** *lat: Latitude of the city (float)* ***:param*** *commutePercentage: Percentage of the population that will travel to a different city (int)  
 """* super().\_\_init\_\_(name)  
 *self*.longitude = long  
 *self*.latitude = lat  
 *self*.commutePercentage = commutePercentage  
 *self*.commutePopulation = []  
 *self*.hosts = []  
 *self*.shopRange = []  
  
 def populate(*self*, db):  
 *"""  
 Sets up the city  
 Makes host and building objects and then run the populate function on them* ***:param*** *db: The database handler class for accessing the database (databaseHandler)  
 """* print("Making host objects for {}...".format(*self*.name))  
 # Makes all the hosts that the city will contain  
 hostCount = db.getHostCount(*self*.name)  
 *self*.hosts = functionLib.makeHosts(hostCount)  
  
 print("Made host objects for {}".format(*self*.name))  
 print("Sorting host objects for {}...".format(*self*.name))  
  
 # Makes a copy of the hosts list  
 hostObjects = *self*.hosts.copy()  
  
 # Dictionary to store hosts sorted by age groups  
 peopleByAgeDict = functionLib.sortHosts(hostObjects)  
  
 print("Sorted host objects for {}".format(*self*.name))  
 print("Making environments and adding hosts for {}...".format(*self*.name))  
  
 # Gets the information about all the environments the city will contain  
 environmentInfo = db.getEnvironments(*self*.name)  
  
 for e in range(len(environmentInfo['Type'])):  
 # If the environment is not a shop (with a dynamic population)  
 if environmentInfo['Type'][e] != "Shop":  
 # For the number of requested environments of that type  
 for \_ in range(environmentInfo['Count'][e]):  
 # Get the number of people to add to that environment  
 hostCount = functionLib.weightedRandom(environmentInfo['LowerBound'][e], environmentInfo['UpperBound'][e], environmentInfo['Average'][e])  
 hostsToGo = functionLib.selectCount(hostCount, peopleByAgeDict[environmentInfo['Type'][e]])  
 # Makes the environment  
 *self*.objects.append(functionLib.makeBuilding(environmentInfo['ActivePeriod'][e], environmentInfo['Type'][e], environmentInfo['InteractionRate'][e], hostsToGo))  
 else:  
 # Adds the range of people that will be going to a shop as an attribute  
 *self*.shopRange += environmentInfo['LowerBound'][e], environmentInfo['UpperBound'][e], environmentInfo['Average'][e]  
 # Makes the required number of buildings with an empty population  
 for x in range(environmentInfo['Count'][e]):  
 *self*.objects.append(Building(environmentInfo['ActivePeriod'][e], environmentInfo['Type'][e], environmentInfo['InteractionRate'][e], []))  
  
 print("Made environments and added hosts for {}".format(*self*.name))  
 print("Finished {}".format(*self*.name))  
  
 def getImmuneCount(*self*):  
 *"""  
 Counts the number of immune hosts objects in each building  
 Checks if the building name is "House" so that hosts are not double counted* ***:return****: The number of immune hosts as an int  
 """* count = 0  
 for building in *self*.objects:  
 if building.name == "House":  
 count += building.getImmuneCount()  
 return count  
  
 def getInfectedCount(*self*):  
 *"""  
 Counts the number of infected hosts objects in each building  
 Checks if the building name is "House" so that hosts are not double counted* ***:return****: the number of infected hosts as an int  
 """* count = 0  
 for building in *self*.objects:  
 if building.name == "House":  
 count += building.getInfectedCount()  
 return count  
  
 def increment(*self*, disease):  
 *"""  
 Runs increment method on each host* ***:param*** *disease: The disease that the simulation is running (disease)  
 """* for h in *self*.hosts:  
 h.increment(disease)  
  
 def decrement(*self*, disease):  
 *"""  
 Runs decrement method on each host* ***:param*** *disease: The disease that the simulation is running (disease)  
 """* for h in *self*.hosts:  
 h.decrement(disease)  
  
 def getCommuters(*self*, percentage):  
 *"""  
 Gets the host objects that will travel out of the city for that time step* ***:param*** *percentage: The percentage of hosts to take (float)* ***:return****: The host objects that will travel out of the city (list)  
 """* count = round(len(*self*.commutePopulation)\*percentage)  
 return functionLib.selectCount(count, *self*.commutePopulation)  
  
 def timeStep(*self*, disease, day):  
 *"""  
 Simulates a day progressing in the city* ***:param*** *disease: The disease the simulation is running (disease)* ***:param*** *day: The name of the day that is being simulated (string)  
 """* # Make a copy of the host objects  
 hostObjects = *self*.hosts.copy()  
 # Iterates through each building in the city  
 for o in *self*.objects:  
 # If the building is a shop (getting the shops population for the day)  
 if o.name == 'Shop':  
 # Find how many hosts will visit this shop  
 numToTake = functionLib.weightedRandom(*self*.shopRange[0], *self*.shopRange[1], *self*.shopRange[2])  
 o.hosts = []  
 # Randomly take these hosts from the copied host list and append them to the shop  
 o.hosts = functionLib.selectCount(numToTake, hostObjects)  
 # Runs time step on the building  
 o.timeStep(disease, day)  
 # Updates the commuting population  
 *self*.commutePopulation = random.sample(*self*.hosts, round(len(*self*.hosts)\*(*self*.commutePercentage/100)))  
 # Runs the increment and decrement methods on all the buildings in the city  
 *self*.increment(disease)  
 *self*.decrement(disease)  
  
  
class Country(container.Container):  
 *"""  
 A class which inherits from Container  
 Will contain cities  
 Handles interactions between cities  
 Is this framework implementations top level environment  
 """* def \_\_init\_\_(*self*, name):  
 *"""  
 Constructor for the country* ***:param*** *name: the name of the country (is left blank because its not used anywhere and doesn't need one) (string)  
 """* super().\_\_init\_\_(name)  
 *self*.percentageMatrix = []  
 *self*.halfwayHouses = []  
  
 def populate(*self*, db):  
 *"""  
 Sets up the country* ***:param*** *db: The database handler class for accessing the database (databaseHandler)  
 """* # Get the information about the cities contained in the country  
 cityDetails = db.getCities()  
  
 print("Retrieved City Information")  
  
 for i in range(len(cityDetails['CityID'])):  
 *self*.objects.append(functionLib.makeCity(cityDetails['CityID'][i], cityDetails['Longitude'][i], cityDetails['Latitude'][0], cityDetails['CommutePercentage'][0]))  
 # Makes the cities  
  
 # Populate the cities (Could be parallel to decrease the time taken to run)  
 startTime = time.time()  
 for city in *self*.objects:  
 city.populate(db)  
  
 print("%s seconds" % (time.time() - startTime))  
 print("Finished All Cities")  
 print("Making City Matrix...")  
  
 *self*.percentageMatrix = functionLib.makeMatrix(cityDetails)  
  
 print("Made City Matrix")  
  
 # Make halfway houses between cities  
 *self*.halfwayHouses = functionLib.makeHalfwayHouses(cityDetails)  
  
 def timeStep(*self*, disease, day):  
 *"""  
 Simulates a day progressing in the country* ***:param*** *disease: The disease the simulation is running (disease)* ***:param*** *day: The current day that is being run (string)  
 """* # Could be parallel to decrease the time taken to run  
 super().timeStep(disease, day)  
  
 # Populates the halfway houses  
 # Halfway houses are used to simulate hosts going between cities and interacting with each other  
 # Only does half the matrix  
 for row in range(len(*self*.halfwayHouses[0])-1):  
 for col in range(row+1, len(*self*.halfwayHouses[0])):  
 *self*.halfwayHouses[row][col].hosts += *self*.objects[row].getCommuters(*self*.percentageMatrix[row][col])  
 *self*.halfwayHouses[row][col].hosts += *self*.objects[col].getCommuters(*self*.percentageMatrix[col][row])  
  
 # Progresses time on each halfway house (Could be parallel to decrease the time taken to run)  
 for row in *self*.halfwayHouses:  
 for h in row:  
 h.timeStep(disease, day)  
 h.hosts = []  
  
 def getInfectedCount(*self*):  
 *"""  
 Gets the number of infected hosts in the country* ***:return****: The number of infected hosts as an int  
 """* return super().getInfectedCount()  
  
 def getImmuneCount(*self*):  
 *"""  
 Gets the number of immune hosts in the country* ***:return****: The number of immune hosts as an int  
 """* return super().getImmuneCount()  
  
 def increment(*self*, disease):  
 *"""  
 Calls the increment method on each city* ***:param*** *disease: The disease the simulation is running (disease)  
 """* super().increment(disease)  
  
 def decrement(*self*, disease):  
 *"""  
 Calls the decrement method on each city* ***:param*** *disease: The disease the simulation is running (disease)  
 """* super().decrement(disease)

### databasehandler\_country\_level\_implementation.py

import pyodbc  
import os  
import validation  
from Framework import databasehandler  
  
  
class DatabaseHandler(databasehandler.DatabaseHandler):  
 *"""  
 Class for handling all input from and output to the database  
 """* def \_\_init\_\_(*self*, config):  
 *"""  
 Constructor of the DatabaseHandler class  
 Database file path is specified  
 Connection to and cursor for the database are made  
 Runs the getIteration function to find the number of the simulation* ***:param*** *config: the config number of the simulation that is being run (int)  
 """* super(DatabaseHandler, *self*).\_\_init\_\_()  
 # Database file path and connection string  
 filename = os.path.join(os.path.expanduser("~"), "Documents/databaseRevised.accdb")  
 conn\_str = (r'DRIVER={Microsoft Access Driver (\*.mdb, \*.accdb)};'  
 r'DBQ=' + filename + ';')  
 # Makes a connection to the database  
 conn = pyodbc.connect(conn\_str)  
 # Adds a cursor to the database connection  
 *self*.cursor = conn.cursor()  
 # What configuration to use  
 *self*.configuration = config  
 # Gets the iteration of this simulation  
 *self*.iteration = *self*.getIteration()  
  
 def writeOutput(*self*, cityName, time, infectedCount, immuneCount):  
 *"""  
 Adds a record to the Output table in the database with the param data* ***:param*** *cityName: Name of the city this output record is for (string)* ***:param*** *time: Time elapsed in the simulation (int)* ***:param*** *infectedCount: The number of infected hosts at time param (int)* ***:param*** *immuneCount: The number of immune hosts at time param (int)  
 """  
 self*.cursor.execute('insert into Output (Iteration, SimulationConfiguration, CityID, TimeElapsed, InfectedHosts, ImmuneHosts)'  
 'values ({},{},\'{}\',{},{},{})'.format(*self*.iteration, *self*.configuration, cityName, time, infectedCount, immuneCount))  
 *self*.cursor.commit()  
  
 def getIteration(*self*):  
 *"""  
 Selects all the records from the Output table where the TimeElapsed is 1 (to reduce the number of records returned)  
 Finds the largest number and then adds one for the current simulation  
 Iteration is the number of that run for the configuration* ***:return****: Largest iteration number + 1 (int)  
 """  
 self*.cursor.execute('select max(Iteration) + 1 from Output where TimeElapsed = 1 and SimulationConfiguration = {}'.format(*self*.configuration))  
 result = *self*.cursor.fetchall()[0][0]  
 return result  
  
 def getCities(*self*):  
 *"""  
 Selects CityID, Longitude, Latitude, CommutePercentage from the SimulationCities  
 where the SimulationConfiguration is the same as was specified that this simulation will use.  
  
 CityID is the name of the city  
 CommutePercentage is the percentage of the population from the city that will travel to different cities each day* ***:return****: CityID, Longitude, Latitude, CommutePercentage for all cities (dict)  
 """  
 self*.cursor.execute('select City.CityID, Longitude, Latitude, CommutePercentage '  
 'from SimulationCities inner join City '  
 'on SimulationCities.CityID = City.CityID '  
 'where SimulationConfiguration = {}'.format(*self*.configuration))  
 result = *self*.cursor.fetchall()  
 cityDict = {  
 'CityID': [],  
 'Longitude': [],  
 'Latitude': [],  
 'CommutePercentage': []  
 }  
 for row in result:  
 cityDict['CityID'].append(row[0])  
 cityDict['Longitude'].append(row[1])  
 cityDict['Latitude'].append(row[2])  
 cityDict['CommutePercentage'].append(row[3])  
  
 return cityDict  
  
 def getHostCount(*self*, cityName):  
 *"""  
 Gets the number of hosts that the specified city will contain* ***:param*** *cityName: The name of the city of which to fetch the data for (string)* ***:return****: the number of hosts in the specified city (int)  
 """  
 self*.cursor.execute('select HostCount from City where CityID = \'{}\''.format(cityName))  
 result = *self*.cursor.fetchall()[0][0]  
 validation.isNoneNegativeInt(result)  
 return result  
  
 def getEnvironments(*self*, cityName):  
 *"""  
 Gets the environments, the number of them, and the population average and bounds,  
 days where the environment is active and the interactionRate for the specified city* ***:param*** *cityName: The name of the city of which to fetch the data for (string)* ***:return****: EnvironmentType, Count, LowerBound, UpperBound, Average, ActivePeriod, interactionRate (dict)  
 """  
 self*.cursor.execute('select CityEnvironments.EnvironmentType, Count, LowerBound, UpperBound, Average, ActivePeriod, interactionRate '  
 'from CityEnvironments inner join Environments on CityEnvironments.EnvironmentType = Environments.EnvironmentType '  
 'where CityEnvironments.CityID = \'{}\''.format(cityName))  
 result = *self*.cursor.fetchall()  
  
 dict = {  
 'Type': [],  
 'Count': [],  
 'LowerBound': [],  
 'UpperBound': [],  
 'Average': [],  
 'ActivePeriod': [],  
 'InteractionRate': []  
 }  
 for row in result:  
 dict['Type'].append(row[0])  
 dict['Count'].append(row[1])  
 dict['LowerBound'].append(row[2])  
 dict['UpperBound'].append(row[3])  
 dict['Average'].append(row[4])  
 dict['ActivePeriod'].append(row[5])  
 dict['InteractionRate'].append(row[6])  
 return dict  
  
 def getDisease(*self*):  
 *"""  
 Gets the information about the disease from the disease table* ***:return****: Duration, LatencyPeriod, InfectionChance, ImmuneDuration of the disease (dict)  
 """  
 self*.cursor.execute('select Duration, LatencyPeriod, InfectionChance, ImmuneDuration '  
 'from Disease inner join Simulation on Disease.DiseaseID = Simulation.Disease '  
 'where SimulationConfiguration = {}'.format(*self*.configuration))  
 result = *self*.cursor.fetchall()[0]  
 validation.isNoneNegativeInt(result[0])  
 validation.isNoneNegativeInt(result[1])  
 validation.isNoneNegativeFloat(result[2])  
 validation.isNoneNegativeInt(result[3])  
 dict = {  
 'Duration': result[0],  
 'LatencyPeriod': result[1],  
 'InfectionChance': result[2],  
 'ImmuneDuration': result[3]  
 }  
 return dict  
  
 def getRuntime(*self*):  
 *"""  
 Gets the runtime that the SimulationConfiguration has specified for this simulation* ***:return****: the simulations runtime (int)  
 """  
 self*.cursor.execute('select RunTime from Simulation where SimulationConfiguration = {}'.format(*self*.configuration))  
 result = *self*.cursor.fetchall()[0][0]  
 validation.isNoneNegativeInt(result)  
 return result  
  
 def getStartDate(*self*):  
 *"""  
 Gets the start date that the SimulationConfiguration has specified for this simulation* ***:return****: the start date (list) [day, month, year]  
 """  
 self*.cursor.execute('select StartDate from Simulation where SimulationConfiguration = {}'.format(*self*.configuration))  
 date = *self*.cursor.fetchall()[0][0]  
 validation.isDate(date)  
 return [date.day, date.month, date.year]

### plotter\_country\_level\_implementation.py

import pandas as pd  
import matplotlib.pyplot as plt  
from Framework import plotter  
import os  
  
  
class Plotter(plotter.Plotter):  
 *"""  
 Class which does all the plotting for the model  
 """* def \_\_init\_\_(*self*, db):  
 *"""  
 Constructor for the plotter* ***:param*** *db: The class which handles all interactions with the database (databaseHandler)  
 """* super(Plotter, *self*).\_\_init\_\_()  
 *self*.db = db  
 *self*.cities = *self*.db.getCities()  
 *self*.output = []  
 for \_ in range(len(*self*.cities)):  
 *self*.output.append(pd.DataFrame(columns=['Day', 'Infected', 'Immunities']))  
  
 def makePlot(*self*):  
 *"""  
 Outputs the final plot for each city once the simulation has finished running  
 """* for i in range(len(*self*.cities['CityID'])):  
 ax = plt.gca()  
  
 *self*.output[i].plot(kind='line', x='Day', y='Infected', ax=ax, color='red')  
 *self*.output[i].plot(kind='line', x='Day', y='Immunities', ax=ax, color='k')  
 plt.xticks(rotation=20)  
  
 filename = os.path.join(os.getcwd(), "Output/{}Output.png".format(*self*.cities['CityID'][i]))  
 plt.savefig(filename)  
 plt.close()  
 print("Plot made for {}".format(*self*.cities['CityID'][i]))  
  
 def updateOutput(*self*, i, topLevel):  
 *"""  
 Updates the pandas dataframe with the hosts infected and immune for the day that was just simulated* ***:param*** *i: The number of days passed since the simulation started (int)* ***:param*** *topLevel: The container that contains all other containers (container)  
 """* for num, city in enumerate(topLevel.objects):  
 *self*.output[num] = *self*.output[num].append({'Day': i, 'Infected': city.getInfectedCount(), 'Immunities': city.getImmuneCount()}, ignore\_index=True)

### functionLib.py

import math  
import random  
from numpy import random as rn  
import place  
import person  
import validation  
  
  
def generatePoisson(rate):  
 # Checking the rate is within the expected range  
 validation.generatePoissonRange(rate)  
  
 res = random.randint(0, 100)/101.0  
 a = -math.log(1.0-res)/(1-rate)  
  
 return a  
  
  
def weightedRandom(lb, ub, avg):  
 # Checking the average is within the lower and upper bounds  
 validation.weightedRandomRange(lb, ub, avg)  
  
 x = []  
 valid = False  
 while not valid:  
 if avg - lb > ub - avg:  
 x = rn.normal(loc=avg, scale=(ub - avg) / 2, size=1)  
 else:  
 x = rn.normal(loc=avg, scale=(avg - lb) / 2, size=1)  
 if lb <= x[0] <= ub:  
 valid = True  
 return round(x[0])  
  
  
def coordsToDistance(lon1, lon2, lat1, lat2):  
 validation.coordRange(lon1)  
 validation.coordRange(lon2)  
 validation.coordRange(lat1)  
 validation.coordRange(lat2)  
  
 # R = 3959.87433 for distance in miles. For Earth radius in kilometers use 6372.8 km  
 R = 6372.8  
 dLat = math.radians(lat2 - lat1)  
 dLon = math.radians(lon2 - lon1)  
 lat1 = math.radians(lat1)  
 lat2 = math.radians(lat2)  
  
 a = math.sin(dLat / 2) \*\* 2 + math.cos(lat1) \* math.cos(lat2) \* math.sin(dLon / 2) \*\* 2  
 c = 2 \* math.asin(math.sqrt(a))  
  
 return R \* c  
  
  
def makeMatrix(cityDetails):  
 # Matrix that stores the distances between the cities  
 matrix = [[0.0 for x in range(len(cityDetails['CityID']))] for y in range(len(cityDetails['CityID']))]  
 # the percentage (in decimal) that the number of people going to that city will be out of all the selected people to travel between cities  
 percentageMatrix = matrix.copy()  
 for i in range(len(cityDetails['CityID'])):  
 for j in range(len(cityDetails['CityID'])):  
 if cityDetails['CityID'][i] != cityDetails['CityID'][j]:  
 distance = coordsToDistance(cityDetails['Longitude'][i], cityDetails['Longitude'][j], cityDetails['Latitude'][i], cityDetails['Latitude'][j])  
 # inverts the distance as number of people traveling will be inversely proportional to the distance  
 # removes distances longer than 250km to simulate a typical journey length  
 if distance < 250:  
 matrix[i][j] = 1 / distance  
 else:  
 matrix[i][j] = 0.0  
  
 # Turns the distance into a percentage from each city  
 for rowNum, row in enumerate(matrix):  
 # Finds the total amount of distance that each row has  
 rowDistance = 0  
 for item in row:  
 rowDistance += item  
 if rowDistance == 0:  
 raise ZeroDivisionError('At least one city is too far from any other city to send hosts between')  
 # Turns each distance into a percentage  
 for itemNum, item in enumerate(row):  
 percentageMatrix[rowNum][itemNum] = item / rowDistance  
  
 return percentageMatrix  
  
  
def makeHalfwayHouses(cityDetails):  
 return [[place.Building("Everyday", "HalfwayHouse", 0.1, []) for x in range(len(cityDetails['CityID']))] for y in range(len(cityDetails['CityID']))]  
  
  
def makeCity(ID, lon, lat, commutePercentage):  
 validation.isFloat(lon)  
 validation.isFloat(lat)  
 validation.isString(ID)  
 validation.isNoneNegativeInt(commutePercentage)  
 validation.percentageRange(commutePercentage)  
 return place.City(ID, lon, lat, commutePercentage)  
  
  
def makeBuilding(activePeriod, name, interactionRate, hosts):  
 validation.isString(activePeriod)  
 validation.knownActivePeriod(activePeriod)  
 validation.isString(name)  
 validation.isNoneNegativeFloat(interactionRate)  
 validation.isList(hosts)  
 if len(hosts) != 0:  
 validation.isHost(hosts[0])  
 return place.Building(activePeriod, name, interactionRate, hosts)  
  
  
def sortHosts(hosts):  
 peopleByAgeDict = {  
 "House": hosts,  
 "Office": [],  
 "School": [],  
 }  
  
 # Sorts the hosts out by age  
 for h in hosts:  
 # If a child  
 if h.age <= 18:  
 peopleByAgeDict["School"].append(h)  
 # If not a child and of working age  
 elif 18 < h.age <= 65:  
 peopleByAgeDict["Office"].append(h)  
  
 return peopleByAgeDict  
  
  
def selectCount(count, items):  
 toGo = []  
 # Tries to take the required number of hosts  
 try:  
 for \_ in range(count):  
 toGo.append(items.pop(random.randint(0, len(items)-1)))  
 # If the list is empty it cant generate a random number so it will catch the error  
 except ValueError:  
 toGo += items  
  
 return toGo  
  
  
def makeHosts(count):  
 hosts = []  
 for \_ in range(count):  
 hosts.append(person.Person())  
  
 return hosts

### preprocessor\_country\_level\_implementation.py

import math  
from Framework import preprocessor  
import place  
import random  
  
  
class Preprocessing(preprocessor.Preprocessing):  
 *"""  
 Class which creates all the other objects for the model  
 """* def \_\_init\_\_(*self*, db):  
 *"""  
 Constructor for the preprocessor* ***:param*** *db: The class which handles all actions with the database (databaseHandler)  
 """* super(Preprocessing, *self*).\_\_init\_\_()  
 *self*.db = db  
  
 def preprocess(*self*, disease):  
 *"""  
 Runs the preprocessing process* ***:param*** *disease: The disease that the simulation with model (disease)* ***:return****: One container type object to the model (container)  
 """* topLevel = place.Country("")  
 topLevel.populate(*self*.db)  
  
 *self*.initialInfection(disease, topLevel)  
 return topLevel  
  
 def initialInfection(*self*, disease, topLevel):  
 *"""  
 Makes the initial infection in the simulation* ***:param*** *disease: The disease the simulation is running (disease)* ***:param*** *topLevel: The container that contains all other containers (container)  
 """* try:  
 # Selects a random city, building and host  
 cityNum = random.randint(0, len(topLevel.objects)-1)  
 c = topLevel.objects[cityNum]  
 buildingNum = random.randint(0, len(c.objects)-1)  
 building = c.objects[buildingNum]  
 hostNum = random.randint(0, len(building.hosts)-1)  
 toInfect = building.hosts[hostNum]  
 toInfect.infected = True  
 toInfect.infectious = True  
 toInfect.infectedTime = 1  
 toInfect.latencyTime = disease.latencyPeriod  
 print("Initial Infection Location:")  
 print("City: {}".format(cityNum))  
 print("Building: {}".format(buildingNum))  
 print("Host: {}".format(hostNum))  
 except ValueError:  
 # Catches an error if the building has no hosts in it so a random number cannot be generated  
 *self*.initialInfection(disease, topLevel)

## Unit Test Files

### city\_tests.py

import unittest  
from unittest.mock import MagicMock, patch, Mock  
import functionLib  
import place  
  
  
class TestConstructor(unittest.TestCase):  
  
 def testConstructorIsAssigningAndMakingProperly(*self*):  
 city = functionLib.makeCity("City", 1.0, 5.0, 10)  
 *self*.assertEqual(city.name, "City")  
 *self*.assertEqual(city.longitude, 1.0)  
 *self*.assertEqual(city.latitude, 5.0)  
 *self*.assertEqual(city.commutePercentage, 10)  
 *self*.assertIsNotNone(city.commutePercentage)  
 *self*.assertIsNotNone(city.hosts)  
 *self*.assertIsNotNone(city.shopRange)  
  
  
class TestCityPopulate(unittest.TestCase):  
  
 @patch("databaseHandler.DatabaseHandler")  
 def testItShouldMakeTheRightNumberOfHosts(*self*, mockDB):  
 mockDB.getHostCount.return\_value = 10  
 City = functionLib.makeCity("City", 10.0, 10.0, 10)  
 City.populate(mockDB)  
 *self*.assertEqual(len(City.hosts), 10)  
  
 @patch("databaseHandler.DatabaseHandler")  
 def testItMakesABuildingAsExpectedWhenNotAShop(*self*, mockDB):  
 environmentDict = {  
 'Type': ['House'],  
 'Count': [10],  
 'LowerBound': [1],  
 'UpperBound': [6],  
 'Average': [4],  
 'ActivePeriod': ['Everyday'],  
 'InteractionRate': [0.7]  
 }  
 mockDB.getEnvironments.return\_value = environmentDict  
 City = functionLib.makeCity("City", 10.0, 10.0, 10)  
 City.populate(mockDB)  
 *self*.assertEqual(len(City.objects), 10)  
 *self*.assertIsInstance(City.objects[0], place.Building)  
 *self*.assertEqual(City.objects[0].activePeriod, ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])  
 *self*.assertEqual(City.objects[0].interactionRate, 0.7)  
  
 @patch("databaseHandler.DatabaseHandler")  
 def testItMakesABuildingAsExpectedWhenAShop(*self*, mockDB):  
 environmentDict = {  
 'Type': ['Shop'],  
 'Count': [10],  
 'LowerBound': [1],  
 'UpperBound': [6],  
 'Average': [4],  
 'ActivePeriod': ['Everyday'],  
 'InteractionRate': [0.3]  
 }  
 mockDB.getEnvironments.return\_value = environmentDict  
 City = functionLib.makeCity("City", 10.0, 10.0, 10)  
 City.populate(mockDB)  
 *self*.assertEqual(len(City.objects), 10)  
 *self*.assertIsInstance(City.objects[0], place.Building)  
 *self*.assertEqual(City.shopRange, [1, 6, 4])  
 *self*.assertEqual(len(City.objects[0].hosts), 0)  
 *self*.assertEqual(City.objects[0].activePeriod, ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])  
 *self*.assertEqual(City.objects[0].interactionRate, 0.3)

### country\_tests.py

import unittest  
from unittest.mock import MagicMock, patch, Mock  
import place  
  
  
class TestPopulate(unittest.TestCase):  
  
 def setUp(*self*):  
 *self*.country = place.Country("Country")  
  
 @patch("databaseHandler.DatabaseHandler")  
 @patch("functionLib.makeMatrix")  
 def testItGivesTheExpectedResults(*self*, mockMakeMatrix, mockDb):  
 mockMakeMatrix.return\_value = [[]]  
 mockDb.getCities.return\_value = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218],  
 'CommutePercentage': [0, 0, 0]  
 }  
 *self*.country.populate(mockDb)  
 *self*.assertEqual(len(*self*.country.objects), 3)  
  
  
class TestConstructor(unittest.TestCase):  
  
 def testConstructor(*self*):  
 country = place.Country('Country')  
 *self*.assertIsNotNone(country.percentageMatrix)  
 *self*.assertIsNotNone(country.halfwayHouses)

### functionLib\_tests.py

import unittest  
from unittest.mock import patch  
import functionLib  
import place  
import validation  
from person import Person  
  
  
class TestMakeMatrix(unittest.TestCase):  
  
 def testItShouldNotBeNone(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 result = functionLib.makeMatrix(cityDetails)  
 *self*.assertIsNotNone(result)  
  
 def testItShouldBeASquareMatrix(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 result = functionLib.makeMatrix(cityDetails)  
 *self*.assertEqual(len(result), len(result[0]))  
  
 def testItShouldHaveExpectedDimensions(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 result = functionLib.makeMatrix(cityDetails)  
 *self*.assertEqual(len(result), 3)  
  
 def testCityOutOfRange(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52, 10, 43],  
 'Latitude': [52, 10, 52]  
 }  
 *self*.assertRaises(ZeroDivisionError, functionLib.makeMatrix, cityDetails)  
  
  
class TestGeneratePoisson(unittest.TestCase):  
  
 @patch('random.randint')  
 def testItShouldHaveKnownResultWhenInRange(*self*, rng):  
 rng.return\_value = 50  
 result = functionLib.generatePoisson(0.3)  
 *self*.assertAlmostEqual(result, 0.9761355487, 10)  
  
 @patch('random.randint')  
 def testItShouldRaiseOutOfRangeExceptionWhenAboveRange(*self*, rng):  
 rng.return\_value = 50  
 *self*.assertRaises(validation.OutOfRange, functionLib.generatePoisson, 1.1)  
  
 @patch('random.randint')  
 def testItShouldRaiseOutOfRangeExceptionWhenBelowRange(*self*, rng):  
 rng.return\_value = 50  
 *self*.assertRaises(validation.OutOfRange, functionLib.generatePoisson, -0.3)  
  
 @patch('random.randint')  
 def testItShouldRaiseOutOfRangeExceptionWhenLowerBoundOfRange(*self*, rng):  
 rng.return\_value = 50  
 *self*.assertRaises(validation.OutOfRange, functionLib.generatePoisson, 0)  
  
 @patch('random.randint')  
 def testItShouldRaiseOutOfRangeExceptionWhenUpperBoundOfRange(*self*, rng):  
 rng.return\_value = 50  
 *self*.assertRaises(validation.OutOfRange, functionLib.generatePoisson, 1)  
  
  
class TestCoordsToDistance(unittest.TestCase):  
  
 def testItShouldHaveKnownValue(*self*):  
 result = functionLib.coordsToDistance(52.4862, 51.4545, -1.8904, -2.5879)  
 *self*.assertAlmostEqual(result, 138.4, 1)  
  
  
class TestHalfwayHouseMatrix(unittest.TestCase):  
  
 def testItShouldNotBeNone(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 halfwayHouses = functionLib.makeHalfwayHouses(cityDetails)  
 *self*.assertIsNotNone(halfwayHouses)  
  
 def testItShouldBeASquareMatrix(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 halfwayHouses = functionLib.makeHalfwayHouses(cityDetails)  
 *self*.assertEqual(len(halfwayHouses), len(halfwayHouses[0]))  
  
 def testItShouldHaveExpectedDimensions(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 halfwayHouses = functionLib.makeHalfwayHouses(cityDetails)  
 *self*.assertEqual(len(halfwayHouses), 3)  
  
 def testItShouldContainABuilding(*self*):  
 cityDetails = {  
 'CityID': ["City1", "City2", "City3"],  
 'Longitude': [52.4862, 51.4545, 52.2053],  
 'Latitude': [-1.8904, -2.5879, 0.1218]  
 }  
 halfwayHouses = functionLib.makeHalfwayHouses(cityDetails)  
 *self*.assertIsInstance(halfwayHouses[0][0], place.Building)  
  
  
class TestSortHosts(unittest.TestCase):  
  
 def testItShouldSortTheHostsCorrectly(*self*):  
 hosts = [Person(), Person(), Person(), Person(), Person(), Person()]  
 hosts[0].age = 4 # school  
 hosts[1].age = 18 # school  
 hosts[2].age = 20 # office  
 hosts[3].age = 65 # office  
 hosts[4].age = 70 # none  
 hosts[5].age = 99 # none  
 result = functionLib.sortHosts(hosts)  
 *self*.assertEqual(len(result["House"]), 6)  
 *self*.assertEqual(len(result["Office"]), 2)  
 *self*.assertEqual(len(result["School"]), 2)

### validation\_tests.py

import unittest  
from ddt import data, ddt  
import validation  
import datetime  
from Framework import databasehandler, disease, host, container  
  
  
class TestValidationGeneratePoissonRange(unittest.TestCase):  
  
 def testItShouldRaiseOutOfRangeWhenInputIsLessThanLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.generatePoissonRange, -0.3)  
  
 def testItShouldRaiseOutOfRangeWhenInputIsTheSameAsTheLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.generatePoissonRange, 0)  
  
 def testItShouldNotRaiseOutOfRangeWhenInputIsInRange(*self*):  
 try:  
 validation.generatePoissonRange(0.3)  
 except validation.OutOfRange:  
 *self*.fail('generatePoissonRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldRaiseOutOfRangeWhenInputIsTheSameAsTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.generatePoissonRange, 1)  
  
 def testItShouldRaiseOutOfRangeWhenInputIsGreaterThanTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.generatePoissonRange, 1.3)  
  
  
class TestValidationWeightedRandomRange(unittest.TestCase):  
  
 def testItShouldRaiseOutOfRangeExceptionWhenAverageIsGreaterThanTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.weightedRandomRange, 1, 3, 4)  
  
 def testItShouldRaiseOutOfRangeExceptionWhenAverageIsLessThanTheLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.weightedRandomRange, 3, 6, 2)  
  
 def testItShouldRaiseOutOfRangeExceptionWhenAverageIsEqualToTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.weightedRandomRange, 1, 3, 3)  
  
 def testItShouldRaiseOutOfRangeExceptionWhenAverageIsEqualToTheLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.weightedRandomRange, 3, 6, 3)  
  
 def testItShouldNotRaiseOutOfRangeExceptionWhenInputDataIsWithinRange(*self*):  
 try:  
 validation.weightedRandomRange(3, 5, 4)  
 except validation.OutOfRange:  
 *self*.fail('WeightedRandomRange Raised OutOfRange Unexpectedly')  
  
  
class TestValidationCoordRange(unittest.TestCase):  
  
 def testItShouldRaiseOutOfRangeWhenTheInputIsLessThanTheLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.coordRange, -200)  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsTheEqualToTheLowerBound(*self*):  
 try:  
 validation.coordRange(-180)  
 except validation.OutOfRange:  
 *self*.fail('CoordRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsWithinTheRange(*self*):  
 try:  
 validation.coordRange(100)  
 except validation.OutOfRange:  
 *self*.fail('CoordRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsEqualToTheUpperBound(*self*):  
 try:  
 validation.coordRange(180)  
 except validation.OutOfRange:  
 *self*.fail('CoordRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldRaiseOutOfRangeWhenTheInputIsGreaterThanTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.coordRange, 200)  
  
  
class TestValidationPercentageRange(unittest.TestCase):  
  
 def testItShouldRaiseOutOfRangeWhenTheInputIsLessThanTheLowerBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.percentageRange, -200)  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsTheEqualToTheLowerBound(*self*):  
 try:  
 validation.percentageRange(0)  
 except validation.OutOfRange:  
 *self*.fail('PercentageRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsWithinTheRange(*self*):  
 try:  
 validation.percentageRange(80)  
 except validation.OutOfRange:  
 *self*.fail('PercentageRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldNotRaiseOutOfRangeWhenTheInputIsEqualToTheUpperBound(*self*):  
 try:  
 validation.percentageRange(100)  
 except validation.OutOfRange:  
 *self*.fail('PercentageRange Raised OutOfRange Unexpectedly')  
  
 def testItShouldRaiseOutOfRangeWhenTheInputIsGreaterThanTheUpperBound(*self*):  
 *self*.assertRaises(validation.OutOfRange, validation.percentageRange, 200)  
  
  
@ddt  
class TestValidateKnownActivePeriod(unittest.TestCase):  
  
 @data('Everyday', 'Weekdays')  
 def testItShouldNotRaiseValueErrorForKnownActivePeriods(*self*, activePeriod):  
 try:  
 validation.knownActivePeriod(activePeriod)  
 except ValueError:  
 *self*.fail('KnownActivePeriod Raised ValueError Unexpectedly')  
  
 def testItShouldRaiseValueErrorForAnUnknownActivePeriod(*self*):  
 *self*.assertRaises(ValueError, validation.knownActivePeriod, 'NotKnown')  
  
  
class TestValidateIsString(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotAString(*self*):  
 *self*.assertRaises(TypeError, validation.isString, 1)  
  
 def testItShouldNotRaiseTypeErrorWhenTheInputIsAString(*self*):  
 try:  
 validation.isString("string")  
 except TypeError:  
 *self*.fail('IsString Raised TypeError Unexpectedly')  
  
  
class TestValidateIsInt(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotAnInt(*self*):  
 *self*.assertRaises(TypeError, validation.isInt, "Not An Int")  
  
 def testItShouldNotRaiseTypeErrorWhenTheInputIsAnInt(*self*):  
 try:  
 validation.isInt(1)  
 except TypeError:  
 *self*.fail('IsString Raised TypeError Unexpectedly')  
  
  
class TestValidateIsFloat(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotFloat(*self*):  
 *self*.assertRaises(TypeError, validation.isFloat, 1)  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsFloat(*self*):  
 try:  
 validation.isFloat(1.1)  
 except TypeError:  
 *self*.fail('IsFloat Raised TypeError Unexpectedly')  
  
  
class TestIsIntOrFloat(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotIntOrFloat(*self*):  
 *self*.assertRaises(TypeError, validation.isIntOrFloat, 'Not An Int Or Float')  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAnInt(*self*):  
 try:  
 validation.isIntOrFloat(1)  
 except TypeError:  
 *self*.fail('IsIntOrFloat Raised TypeError Unexpectedly')  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAFloat(*self*):  
 try:  
 validation.isIntOrFloat(1.2)  
 except TypeError:  
 *self*.fail('IsIntOrFloat Raised TypeError Unexpectedly')  
  
  
class TestIsNoneNegativeInt(unittest.TestCase):  
  
 def testItShouldRaiseValueErrorWhenInputIsLessThanZero(*self*):  
 *self*.assertRaises(ValueError, validation.isNoneNegativeInt, -1)  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotAnInt(*self*):  
 *self*.assertRaises(TypeError, validation.isNoneNegativeInt, 'Not An Int')  
  
 def testItShouldNotRaiseValueErrorWhenInputIsPositive(*self*):  
 try:  
 validation.isNoneNegativeInt(1)  
 except ValueError:  
 *self*.fail('IsNoneNegativeInt Raised ValueError Unexpectedly')  
  
  
class TestIsNoneNegativeFloat(unittest.TestCase):  
  
 def testItShouldRaiseValueErrorWhenInputIsNegativeFloat(*self*):  
 *self*.assertRaises(ValueError, validation.isNoneNegativeFloat, -1.2)  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotAFloat(*self*):  
 *self*.assertRaises(TypeError, validation.isNoneNegativeFloat, 'Not A Float')  
  
 def testItShouldNotRaiseValueErrorWhenInputIsAPositiveFloat(*self*):  
 try:  
 validation.isNoneNegativeFloat(1.5)  
 except ValueError:  
 *self*.fail('IsNoneNegativeFloat Raised ValueError Unexpectedly')  
  
  
class TestIsNoneNegativeFloatOrInt(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotFloatOrInt(*self*):  
 *self*.assertRaises(TypeError, validation.isNoneNegativeFloat, 'Not A Float Or Int')  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsPositiveFloat(*self*):  
 try:  
 validation.isNoneNegativeFloatOrInt(1.2)  
 except TypeError:  
 *self*.fail('IsNoneNegativeFloatOrInt Raised TypeError Unexpectedly')  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsFloat(*self*):  
 try:  
 validation.isNoneNegativeFloatOrInt(1)  
 except TypeError:  
 *self*.fail('IsNoneNegativeFloatOrInt Raised TypeError Unexpectedly')  
  
 def testItShouldRaiseValueErrorWhenInputIsNegativeFloat(*self*):  
 *self*.assertRaises(ValueError, validation.isNoneNegativeFloatOrInt, -6.3)  
  
 def testItShouldRaiseValueErrorWhenInputIsNegativeInt(*self*):  
 *self*.assertRaises(ValueError, validation.isNoneNegativeFloatOrInt, -2)  
  
  
class TestIsList(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotList(*self*):  
 *self*.assertRaises(TypeError, validation.isList, 3)  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAList(*self*):  
 try:  
 validation.isList([1])  
 except TypeError:  
 *self*.fail('IsList Raised TypeError Unexpectedly')  
  
  
class TestIsDate(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputIsNotDate(*self*):  
 *self*.assertRaises(TypeError, validation.isDate, 'Not A Date')  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsADate(*self*):  
 try:  
 validation.isDate(datetime.datetime(2022, 3, 12))  
 except TypeError:  
 *self*.fail('IsDate Raised TypeError Unexpectedly')  
  
  
class TestIsDatabaseHandler(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputAClassThatDoesNotInheritFromDatabaseHandler(*self*):  
 *self*.assertRaises(TypeError, validation.isDatabaseHandler, NoneDatabaseHandler())  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAClassThatInheritsFromDatabaseHandler(*self*):  
 try:  
 validation.isDatabaseHandler(DummyDatabaseHandler())  
 except TypeError:  
 *self*.fail('IsDatabaseHandler Raised TypeError Unexpectedly')  
  
  
class TestIsDisease(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputAClassThatDoesNotInheritFromDisease(*self*):  
 *self*.assertRaises(TypeError, validation.isDisease, NoneDisease())  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAClassThatInheritsFromDisease(*self*):  
 try:  
 validation.isDisease(DummyDisease())  
 except TypeError:  
 *self*.fail('IsDisease Raised TypeError Unexpectedly')  
  
  
class TestIsHost(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputAClassThatDoesNotInheritFromHost(*self*):  
 *self*.assertRaises(TypeError, validation.isHost, NoneHost())  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAClassThatInheritsFromHost(*self*):  
 try:  
 validation.isHost(DummyHost())  
 except TypeError:  
 *self*.fail('IsHost Raised TypeError Unexpectedly')  
  
  
class TestIsContainer(unittest.TestCase):  
  
 def testItShouldRaiseTypeErrorWhenInputAClassThatDoesNotInheritFromContainer(*self*):  
 *self*.assertRaises(TypeError, validation.isContainer, NoneContainer())  
  
 def testItShouldNotRaiseTypeErrorWhenInputIsAClassThatInheritsFromContainer(*self*):  
 try:  
 validation.isContainer(DummyContainer())  
 except TypeError:  
 *self*.fail('IsContainer Raised TypeError Unexpectedly')  
  
  
class NoneContainer:  
 pass  
  
  
class DummyContainer(container.Container):  
  
 def \_\_init\_\_(*self*):  
 name = ""  
 super().\_\_init\_\_(name)  
  
 def timeStep(*self*, d, day):  
 pass  
  
 def getInfectedCount(*self*):  
 pass  
  
 def getImmuneCount(*self*):  
 pass  
  
 def increment(*self*, d):  
 pass  
  
 def decrement(*self*, d):  
 pass  
  
 def populate(*self*, db):  
 pass  
  
  
class NoneHost:  
 pass  
  
  
class DummyHost(host.Host):  
  
 def increment(*self*, d):  
 pass  
  
 def decrement(*self*, d):  
 pass  
  
  
class NoneDisease:  
 pass  
  
  
class DummyDisease(disease.Disease):  
 pass  
  
  
class NoneDatabaseHandler:  
 pass  
  
  
class DummyDatabaseHandler(databasehandler.DatabaseHandler):  
  
 def getStartDate(*self*):  
 pass  
  
 def getDisease(*self*):  
 pass  
  
 def getRuntime(*self*):  
 pass  
  
 def writeOutput(*self*, cityName, time, infectedCount, immuneCount):  
 pass

## Map Plotting Script

*This script must be run separately and the number for iteration, runtime and configuration must be changed manually. It is also made specifically for the country level implementation.*

### Map\_plotter.py

import matplotlib.pyplot as plt  
import cartopy.crs as ccrs  
import cartopy.feature as cfeature  
import pyodbc  
import os  
import matplotlib.patches as mpatches  
from datetime import datetime, date, timedelta  
  
  
class cityInfo(object):  
  
 def \_\_init\_\_(*self*, name, long, lat, maxPop):  
 *self*.name = name  
 *self*.long = long  
 *self*.lat = lat  
 *self*.maxPop = maxPop  
 *self*.infections = []  
 *self*.immunities = []  
  
  
class mapPlot:  
  
 def \_\_init\_\_(*self*, iteration, runtime, simconfig):  
 filename = os.path.join(os.path.expanduser("~"), "Documents/databaseRevised.accdb")  
 conn\_str = (r'DRIVER={Microsoft Access Driver (\*.mdb, \*.accdb)};'  
 r'DBQ=' + filename + ';')  
 conn = pyodbc.connect(conn\_str)  
 *self*.cursor = conn.cursor()  
 *self*.iteration = iteration  
 *self*.simconfig = simconfig  
 *self*.runtime = runtime  
  
 def plot(*self*):  
 *self*.cursor.execute('select StartDate from Simulation where SimulationConfiguration={}'.format(*self*.simconfig))  
 startDate = *self*.cursor.fetchall()[0][0]  
 *self*.cursor.execute('select Disease from Simulation where SimulationConfiguration={}'.format(*self*.simconfig))  
 diseaseName = *self*.cursor.fetchall()[0][0]  
  
 *self*.cursor.execute('select City.CityID, Longitude, Latitude, HostCount from SimulationCities inner join City on SimulationCities.CityID = City.CityID where SimulationConfiguration={}'.format(*self*.simconfig))  
 cityDBInfo = *self*.cursor.fetchall()  
 cities = []  
 for city in cityDBInfo:  
 cities.append(cityInfo(city[0], city[1], city[2], city[3]))  
  
 for city in cities:  
 for i in range(*self*.runtime):  
 *self*.cursor.execute('select InfectedHosts, ImmuneHosts from Output where CityID = \'{}\' and Iteration = {} and SimulationConfiguration = {} and TimeElapsed = {}'.format(city.name, *self*.iteration, *self*.simconfig, i+1))  
 data = *self*.cursor.fetchall()[0]  
 city.infections.append(data[0])  
 city.immunities.append(data[1])  
  
 cmap = plt.get\_cmap('Reds')  
  
 for i in range(*self*.runtime):  
 ax = plt.axes(projection=ccrs.Miller())  
  
 resDate = startDate + timedelta(days=i - 1)  
 res = resDate.strftime("%d-%m-%Y")  
  
 ax.coastlines()  
 # ax.set\_extent([-12, 2, 49, 59])  
 ax.set\_extent([-4.5, 1.5, 50, 54])  
  
 ax.add\_feature(cfeature.NaturalEarthFeature('physical', 'land', '50m', edgecolor='black', facecolor=cfeature.COLORS['land']))  
 ax.add\_feature(cfeature.NaturalEarthFeature('physical', 'lakes', '50m', edgecolor='none', facecolor=cfeature.COLORS['water']), alpha=0.5)  
 ax.add\_feature(cfeature.NaturalEarthFeature('physical', 'rivers\_lake\_centerlines', '50m', edgecolor=cfeature.COLORS['water'], facecolor='none'))  
 for city in cities:  
 ax.text(city.lat, city.long, city.name, transform=ccrs.Geodetic())  
 ax.add\_patch(mpatches.Ellipse(xy=[city.lat, city.long], width=0.4, height=0.3, color=cmap((city.infections[i]/city.maxPop)), alpha=1, transform=ccrs.PlateCarree()))  
 sm = plt.cm.ScalarMappable(cmap=cmap)  
 sm.\_A = []  
 cb = plt.colorbar(sm)  
  
 rect = [0.0, 0.04, 1, 0.0]  
 ax1 = plt.axes(rect)  
 ax1.text(0.5, 0.0, res + " ({})".format(i), ha='center', fontsize=15)  
 ax1.spines['right'].set\_visible(False)  
 ax1.spines['top'].set\_visible(False)  
 ax1.spines['bottom'].set\_visible(False)  
 ax1.spines['left'].set\_visible(False)  
 ax1.yaxis.set\_ticks\_position('none')  
 ax1.xaxis.set\_ticks\_position('none')  
 ax1.xaxis.set\_ticklabels([])  
 ax1.yaxis.set\_ticklabels([])  
  
 rect = [0.9, 0.35, 1, 0]  
 ax2 = plt.axes(rect)  
 ax2.text(0, 0, "Proportion Infected", ha='center', fontsize=10, rotation=90)  
 ax2.spines['right'].set\_visible(False)  
 ax2.spines['top'].set\_visible(False)  
 ax2.spines['bottom'].set\_visible(False)  
 ax2.spines['left'].set\_visible(False)  
 ax2.yaxis.set\_ticks\_position('none')  
 ax2.xaxis.set\_ticks\_position('none')  
 ax2.xaxis.set\_ticklabels([])  
 ax2.yaxis.set\_ticklabels([])  
  
 rect = [0.0, 0.92, 1, 0.0]  
 ax3 = plt.axes(rect)  
 ax3.text(0.5, 0.0, diseaseName, ha='center', fontsize=20)  
 ax3.spines['right'].set\_visible(False)  
 ax3.spines['top'].set\_visible(False)  
 ax3.spines['bottom'].set\_visible(False)  
 ax3.spines['left'].set\_visible(False)  
 ax3.yaxis.set\_ticks\_position('none')  
 ax3.xaxis.set\_ticks\_position('none')  
 ax3.xaxis.set\_ticklabels([])  
 ax3.yaxis.set\_ticklabels([])  
  
 plt.savefig('{}-{}-{}'.format(*self*.simconfig, *self*.iteration, i))  
 plt.figure()  
 print('Made pic {}'.format(i))  
  
  
mp = mapPlot(7, 100, 3)  
# Edit numbers above for the desired iteration, runtime and configuration to plot for  
mp.plot()

# Testing

## Tables Of Tests

### Validation tests

All the tests for the functions that are in the script validation.py. Unit tests for these functions are in the validation\_tests.py file.

*OutOfRange is a custom exception that is raised when an input is outside of the range the function was expecting.*

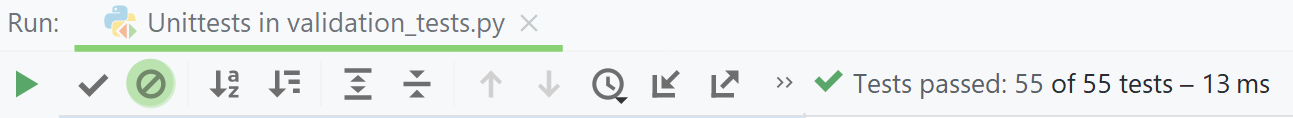
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | Case | Test Data | Reason | Expected Result | Actual Result | Date | Corrective Actions |
| 1 | Generate Poisson Range  (Has Unit Test) | -0.3 | To test that the OutOfRange exception is raised when the input data is less than the lower bound of range (0) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 2 | Generate Poisson Range  (Has Unit Test) | 0 | To test that the OutOfRange exception is raised when the input data is the same as the lower bound of range (0) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 3 | Generate Poisson Range  (Has Unit Test) | 0.3 | To test that the OutOfRange exception is not raised when the input data is within the expected range of values | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 4 | Generate Poisson Range  (Has Unit Test) | 1 | To test that the OutOfRange exception is raised when the input data is the same as the upper bound of range (1) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 5 | Generate Poisson Range  (Has Unit Test) | 1.3 | To test that the OutOfRange exception is raised when the input data is greater than the upper bound of range (1) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 6 | Weighted Random Range  (Has unit test) | Lb = 1  Ub = 3  Avg = 4 | Test it raises OutOfRange exception when the average is greater than the upper bound | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 7 | Weighted Random Range  (Has unit test) | Lb = 3  Ub = 6  Avg = 2 | Test it raises OutOfRange exception when the average is less than the lower bound | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 8 | Weighted Random Range  (Has unit test) | Lb = 1  Ub = 3  Avg = 3 | Test it raises OutOfRange exception when the average is the same as the upper bound | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 9 | Weighted Random Range  (Has unit test) | Lb = 3  Ub = 6  Avg = 3 | Test it raises OutOfRange exception when the average is the same as the lower bound | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 10 | Weighted Random Range  (Has unit test) | Lb = 3  Ub = 5  Avg = 4 | Test it does not raise an exception when the data is within the expected range | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 11 | Coord Range  (Has unit test) | -200 | Test it raises OutOfRange exception when the input data is less than the lower bound of the range (-180) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 12 | Coord Range  (Has unit test) | -180 | Test it does not raise an exception when the input data is the same as the lower bound of the range | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 13 | Coord Range  (Has unit test) | 100 | Test it does not raise an exception when the input data is within the expected range | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 14 | Coord Range  (Has unit test) | 180 | Test it does not raise an exception when the input data is the same as the upper bound of the range (180) | No Exception Raised | No Exception Raised | 12/3/22 | N/A |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | Coord Range  (Has unit test) | 200 | Test it raises OutOfRange exception when the input data is greater than the upper bound of the range (180) | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 16 | Percentage Range  (Has unit test) | -200 | Test it raised OutOfRange exception when the input data is less than the lower bound of the range | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 17 | Percentage Range  (Has unit test) | 0 | Test it does not raise an exception when the input data is the same as the lower bound of the range | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 18 | Percentage Range  (Has unit test) | 80 | Test it does not raise an exception when the input data is within the expected range of values | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 19 | Percentage Range  (Has unit test) | 100 | Test it does not raise an exception when the input data is the same as the upper bound of the range | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 20 | Percentage Range  (Has unit test) | 200 | Test it raises OutOfRange exception when the input data is greater than the upper bound of the range | OutOfRange Exception Raised | OutOfRange Exception Raised | 12/3/22 | N/A |
| 21 | Known Active Period  (Has unit test) | ‘Everyday’ | Test it does not raise an exception when the input data is a known active period | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 22 | Known Active Period  (Has unit test) | ‘Weekdays’ | Test it does not raise an exception when the input data is a known active period | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 23 | Known Active Period  (Has unit test) | ‘NotKnown’ | Test it raises ValueError exception when the input data is not a known active period | ValueError Exception Raised | ValueError Exception Raised | 12/3/22 | N/A |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 24 | Is String  (Has unit test) | 1 | Test it raises TypeError when the input data is not a string | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 25 | Is String  (Has unit test) | ‘string’ | Test it does not raise an exception when the input data is a string | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 26 | Is Int  (Has unit test) | ‘Not An Int’ | Test it raises TypeError when the input data is not an int | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 27 | Is Int  (Has unit test) | 1 | Test it does not raise an exception when the input data is an int | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 28 | Is Float  (Has unit test) | 1 | Test it raises TypeError when the input data is not a float | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 29 | Is Float  (Has unit test) | 1.2 | Test it does not raise an exception when the input data is a float | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 30 | Is Int Or Float  (Has unit test) | ‘Not An Int Or Float’ | Test it raises TypeError when the input data is neither a float or an int | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 31 | Is Int Or Float  (Has unit test) | 1 | Test it does not raise an exception when the input value is an int | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 32 | Is Int Or Float | 1.2 | Test it does not raise an exception when the input value is a float | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 33 | Is None Negative Int  (Has unit test) | -1 | Test it raises ValueError exception when the input data is a negative int | ValueError Exception Raised | ValueError Exception Raised | 12/3/22 | N/A |
| 34 | Is None Negative Int  (Has unit test) | ‘Not An Int’ | Test it raises TypeError exception when the input data is not an int | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 35 | Is None Negative Int  (Has unit test) | 1 | Test it does not raise an exception when the input data is a positive int | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 36 | Is None Negative Float  (Has unit test) | -1.2 | Test it raises ValueError exception when the input data is a negative float | ValueError Exception Raised | ValueError Exception Raised | 12/3/22 | N/A |
| 37 | Is None Negative Float  (Has unit test) | ‘Not A Float’ | Test it raises TypeError exception when the input data is not a float | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 38 | Is None Negative Float  (Has unit test) | 1.5 | Test it does not raise an exception when the input data is a positive float | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 39 | Is None Negative Float Or Int  (Has unit test) | ‘Not A Float Or Int’ | Test it raises TypeError when the input data is neither a float or an int | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 40 | Is None Negative Float Or Int  (Has unt test) | 1.2 | Test it does not raise an exception when the input data is a positive float | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 41 | Is None Negative Float Or Int  (has unit test) | 1 | Test it does not raise an exception the input data is a positive int | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 42 | Is None Negative Float Or Int  (Has unit test) | -6.3 | Test it raises ValueError exception when the input data is a negative float | ValueError Exception Raised | ValueError Exception Raised | 12/3/22 | N/A |
| 43 | Is None Negative Float Or Int  (Has unit test) | -2 | Test it raises ValueError exception when the input data is a negative int | ValueError Exception Raised | ValueError Exception Raised | 12/3/22 | N/A |
| 44 | Is List  (Has unit test) | 3 | Test it raises TypeError when the input data is not a list | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 45 | Is List  (Has unit test) | [1] | Test it does not raise an exception when the input data is a list | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 46 | Is Date  (Has unit test) | ‘Not A Date’ | Test it raises TypeError when the input data is not a date | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 47 | Is Date  (Has unit test) | 12/3/2022 as a datetime | Test it does not raise an exception when the input data is a date | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 48 | Is Database Handler  (Has unit test) | None DatabaseHandler class (does not inherit from database handler) | Test it raises TypeError exception when the input data does not inherit from database handler | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 49 | Is Database Handler  (Has unit test) | Dummy DatabaseHandler class (does inherit from database handler) | Test it does not raise an exception when the input data does inherit from database handler | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 50 | Is Disease  (Has unit test) | None Disease class (does not inherit from disease) | Test it raises TypeError exception when the input data does not inherit from disease | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 51 | Is Disease  (Has unit test) | Dummy Disease class (does inherit from disease) | Test it does not raise an exception when the input data does inherit from disease | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 52 | Is Host  (Has unit test) | None Host class (does not inherit from host) | Test it raises TypeError exception when the input data does not inherit from host | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 53 | Is Host  (Has unit test) | Dummy Host class (does inherit from host) | Test it does not raise an exception when the input data does inherit from disease | No Exception Raised | No Exception Raised | 12/3/22 | N/A |
| 54 | Is Container  (Has unit test) | None Container class (does not inherit from container) | Test it raised TypeError when the input data does not inherit from container | TypeError Exception Raised | TypeError Exception Raised | 12/3/22 | N/A |
| 55 | Is Container  (Has unit test) | Dummy Container class (does inherit from container) | Test it does not raise an exception when the input data does inherit from container | No Exception Raised | No Exception Raised | 12/3/22 | N/A |

### Validation unit tests passing



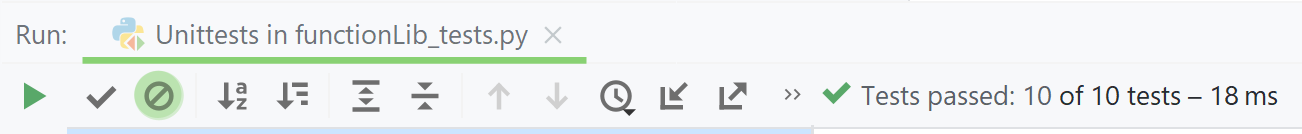
### Function library tests

All the tests for the functions that are in the script functionLib.py. Unit tests for these functions are in the functionLib\_tests.py file.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | Case | Test Data | Reason | Expected Result | Actual Result | Date | Corrective Actions |
| 1 | Generate Poisson  (Has unit test) | Rate = 0.3  Rng = 50 | To test it gives the expected result with a known rate and random number | 0.9761355487 | 0.9761355487 | 01/03/22 | N/A |
| 2 | Generate Poisson  (Has unit test) | Rate = 1.1  Rng = 50 | To test it will throw an exception when the rate is above the upper bound | OutOfRange Exception | OutOfRange Exception | 01/03/22 | N/A |
| 3 | Generate Poisson  (Has unit test) | Rate = -0.3  Rng = 50 | To test it will throw an exception when the rate is below the lower bound | OutOfRange Exception | OutOfRange Exception | 01/03/22 | N/A |
| 4 | Generate Poisson  (Has unit test) | Rate = 0  Rng = 50 | To test it will throw an exception when the rate is the lower bound | OutOfRange Exception | OutOfRange Exception | 01/03/22 | N/A |
| 5 | Generate Poisson  (Has unit test) | Rate = 1  Rng = 50 | To test it will throw an exception when the rate is the upper bound | OutOfRange Exception | OutOfRange Exception | 01/03/22 | N/A |
| 6 | Sorting Hosts  (Has unit test) | Ages:  [4, 18, 20, 65, 70, 99] | Test the function sorts hosts into the correct groups based off their age | 6 Hosts in houses list  2 Hosts in office list  2 Hosts in school list | 6 Hosts in houses list  2 Hosts in office list  2 Hosts in school list | 02/03/22 | N/A |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | Make Matrix  (Has unit test) | Dictionary  'CityID': ["City1", "City2", "City3"],  'Longitude': [52.4862, 51.4545, 52.2053],  'Latitude': [-1.8904, -2.5879, 0.1218] | Test the matrix has equal x and y dimensions | Number of rows = number of columns | Number of rows = number of columns | 03/03/22 | N/A |
| 8 | Make Matrix  (Has unit test) | Dictionary  'CityID': ["City1", "City2", "City3"],  'Longitude': [52.4862, 51.4545, 52.2053],  'Latitude': [-1.8904, -2.5879, 0.1218] | Test the matrix has the expected number of items in a row or column | Has 3 items in a row | Has 3 items in a row | 03/03/22 | N/A |
| 9 | Make Matrix  (Has unit test) | Dictionary  'CityID': ["City1", "City2", "City3"],  'Longitude': [52, 10, 43],  'Latitude': [52, 10, 52] | Test that an exception is thrown when all the cities are to far apart from each other | ZeroDivisionError Exception | ZeroDivisionError Exception | 5/03/22 | N/A |
| 10 | Converting Coordinates to distance  (Has unit test) | (52.4862, - 1.8904)  (51.4545, -2.5879)  Radius of Earth in km = 6372.8 | Test the function gives the correct result correct to 1 decimal place | 138.4 | 138.4 | 5/03/22 | N/A |

### Function library unit tests passing



### Database handler tests

*Images that are referenced are below the table in the next heading*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | Case | Test Data | Reason | Expected Result | Actual Result | Date | Corrective Actions |
| 1 | Write Output Function | 1, 1, ‘Exeter’, 124, 16, 2632 | Test it is writing data to the database properly | Iteration = 1,  SimulationConfiguration = 1,  CityID = ‘Exeter’,  TimeElapsed = 124,  InfectedHosts = 16,  ImmuneHosts = 2632 | *Database Handler Picture IMAGE 1* | 10/03/22 | N/A |
| 2 | Get Cities Function | SimulationConfiguration = 3 | Test it is retrieving the correct data about the cities from the database for the specified simulation configuration | (‘Cambridge’, 52.2053, 0.1218, 15), (‘Peterborough’, 52.5695, -0.2405, 10), (‘Norwich’, 52.6309, 1.2974, 10) | *Database Handler Picture IMAGE 2* | 10/03/22 | N/A |
| 3 | Get Host Count Function | cityName = ‘Cambridge’ | Test it is retrieving the correct host count from the database for the city | 129000 | *Database Handler Picture IMAGE 3* | 10/03/22 | N/A |
| 4 | Get Environments Function | cityName = ‘Cambridge’ | Test it is retrieving the correct data about the environments for the specified city | ('House', 50000, 1.0, 6.0, 3, 'Everyday', 0.8), ('Office', 2150, 20.0, 450.0, 220, 'Weekdays', 0.3), ('School', 50, 40.0, 800.0, 290, 'Weekdays', 0.7), ('Shop', 60, 25.0, 500.0, 150, 'Everyday', 0.5) | *Database Handler Picture IMAGE 4* | 10/03/22 | N/A |
| 5 | Get Disease Function | SimulationConfiguration = 3 | Test it is retrieving the correct data about the disease for the specified simulation configuration | (14, 2, 0.1, 30) | *Database Handler Picture IMAGE 5* | 10/03/22 | N/A |
| 6 | Get Runtime Function | SimulationConfiguration = 3 | Test it is retrieving the correct runtime for the specified simulation configuration | 100 | *Database Handler Picture IMAGE 6* | 10/03/22 | N/A |
| 7 | Get Start Date Function | SimulationConfiguration = 3 | Test it is retrieving the correct start date for the specified simulation configuration | 2022-12-09 00:00:00 | *Database Handler Picture IMAGE 7* | 10/03/22 | N/A |

### Database handler pictures

|  |  |
| --- | --- |
| # | Picture |
| 1 | Table  Description automatically generated |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |

### Testing of graphs produced

*Images that are referenced are below the table in the next heading.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | Case | Test Data | Reason | Expected Result | Actual Result | Date | Corrective Actions |
| 1 | Gradient of infections | InfectionChance = 0.1  Runtime = 100 | Test that the graph which is output for a city has a positive gradient for infections followed by a negative one | Gradient of line for infections (red) is positive followed by a negative. | *Graph picture IMAGE 1* | 25/3/22 | N/A |
| 2 | Gradient of infections | InfectionChance = 0.2  Runtime = 100  Same city as test 1 | Test that increasing the infection chance makes the disease more infectious | Gradient of the infection line is steeper than the one in test 1. The disease also infects more people in the same amount of time. | *Graph picture IMAGE 2*  *(the max y on the graph is different than the one in graph 1 because graph 1 didn’t get to a high enough number to display)* | 25/3/22 | N/A |
| 3 | Plotting the map | (‘Cambridge’, 52.2053, 0.1218, 15), (‘Peterborough’, 52.5695, -0.2405, 10), (‘Norwich’, 52.6309, 1.2974, 10) | Test that the map plotting script has the expected number of locations on the map | Three locations marked on the map: Cambridge, Peterborough and Norwich | *Graph picture IMAGE 3*  *Image at the 68th day* | 25/3/22 | N/A |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | Herd Immunity | Population = 213160  R Naught = 1.245  Runtime = 1000 | Test that the proportion of the population that becomes immune after running for a long time holds true to the herd immunity equation | The average RNaught for this simulation was 1.245  The proportion of immune hosts on average is 0.197 of the population of the city. 42000 hosts.  HIT = 1 – 1/1.245 = 0.197  0.197 \* 213160 = 42000 | *Graph picture IMAGE 4*  *Blue line indicates the average amount of hosts which are immune.* | 25/3/22 | N/A |
| 5 | Correct number of maps | Runtime = 200  Number of cities = 11 | Test that the script outputs the correct number of maps | 200 maps with 11 cities on | *Graph picture IMAGE 5*  *For the sake of saving space, only every 10 pictures are shown* | 06/04/22 | N/A |

### Graph pictures

|  |  |
| --- | --- |
| # | Picture |
| 1 | Chart  Description automatically generated |
| 2 |  |
| 3 |  |
| 4 |  |

|  |  |
| --- | --- |
| 5 |  |

# Evaluation

## Evaluation against project objectives

|  |  |  |
| --- | --- | --- |
| Objective | How was it met? | How could it be expanded upon? |
| Simulate the spread of a disease through a population where the hosts are also the vectors. | This objective was met by creating a host class and a building class and having hosts assigned to multiple buildings so the disease can spread through them. | Use of differential equations to calculate how many people should be infected on a given timestep.  Could also run the simulation multiple times to average the results to improve liability |
| Create base classes for inheritance for the framework. | This objective has been met through the identification of the components of the model that are required for the simulation to run and the user requirements. These components have been turned into abstract classes for inheritance. | Some additional implementations of the abstract classes to demonstrate their use. |
| Read simulation data to and from a database. | This was met using the database handler class where all the SQL statements are. | No room for improvement at this point in the project. |
| Control what environments are run for a given simulation with the database. | This was met through the use of the SimulationCities table in the database. It allows you to assign cities to a simulation number and then can tell the model what simulation number to run. | Make a user interface for easier usage as the current solution assumes the user knows how to use Microsoft Access. |
| Control the properties of a environments with the database. | This has been achieved through the use of the CityEnvironments table in the database. It allows you to assign types of environments to different cities and control the number of that type and the range of hosts in it. | Make a user interface for easier usage as the current solution assumes the user knows how to use Microsoft Access. |
| Plot a graph to show infections and immunities at the end of the model run. | This has been met by using the plotting class. After each timestep the infections and immunities are stored in a pandas data frame and then converted into a graph | Add options for different types of graphs to display the data in a more understandable way |

|  |  |  |
| --- | --- | --- |
| Accommodate for scenarios where the spatial axis is different (simulate a whole country vs simulate just a city). | This objective has been met with the use of the container class. It means that things can just be contained by another container until the desired spatial axis has been reached. | Make it easier to mix the population between containers. Currently it is only designed with moving people between cities in mind. |
| EXTRA TIME: Plot and animate a choropleth map to show infections and their locations at the end of the model run. | This objective has been met with the use of the map plotting script that can be run after the simulation. | If at the end of the simulation the map plotting script was automatically run this would be great however, because the main model is built in a pip environment, I cannot use the plotting software as it requires an anaconda environment.  This script also requires a third-party service to combine the pictures into a gif file. |
| EXTRA TIME: Accommodate for scenarios where the host and the vectors are different entities. | This objective has not been met due to time constraints | It could be implemented into the model in a few ways. Like Dr. Bradshaw’s wheat rust model, the vector locations could be computed beforehand and read from the database during the simulation. It could also be implemented by storing the number of infectious vectors in an environment at the end of each time step so when the next time step begins the environment knows if there are any infectious vectors inside. |

## Client Feedback

In order to hand over the framework to the client, I arranged a training session with Dr. Bradshaw to show her how to use the framework to create an implementation of her existing wheat rust model. I made a branch of the source code in GitHub and after inheriting from the abstract classes I showed her how her existing code would fit into my framework. I then conducted an interview with Dr. Bradshaw for the purpose of obtaining some feedback on the project.

*Q. Would you say the project has met your requirements?*

A. Yes, I am very impressed with what you have produced and I will definitely use it in my future work. You’ve made me a tool in python that can operate at different spatial scales using initial conditions that I can enter easily into a database and you have enabled me to store the disease statistics through time and space in a database as well as produce plots, both maps and time series graphs.

*Q. Is there any specific feedback you can give me on the framework?*

A. I wanted to simplify my existing model such that it could be used to simulate other diseases and your framework does exactly that. I like the way it’s simple to add environments inside other environments in order to make the spatial scale more detailed as you have in your example implementation for covid, or alternatively to just have a top level environment as was the case for my wheat rust model.

I really like the way the set up for each scenario I want to simulate is stored in the database because it makes it easy to look back at it when I come to write things up. In my existing system, I have to make a new file for each scenario, give it a run ID, import the appropriate receptors and sources for each scenario and make sure I include all the information in the result files so that I can figure out what I did. Now in the new framework, not only is all that information stored more simply, it also stores the output statistics that my existing model never did.

When we implemented my model in your framework, I was amazed at how many functions we could just delete because they were no longer needed because the framework already handled those aspects and all that was requires was accessing a property on the class.

*Q. Did you find it easy to use?*

A. I’m not sure that I could have implemented my model without you being there to explain things as I’m not a software engineer and I’ve never come across abstract classes before so I recommend that you write a user guide to go with the code on GitHub to explain fully how to use it. In terms of running the model, it was extremely simple to use, just setting the configuration number of the simulation to run from the database and that was it. Then using that same number, I can access all the results I need.

*Q. Do you have any suggestions for improvements that could be made?*

A. Other than writing a user guide, I would suggest renaming all your variables in the snake case format to be consistent with typical python notation. Also, although you couldn’t use it in your implementation because it’s a private Met Office library, I would probably get the plotting to use the library Ascend so that polygon shapes as well as just markers could be used in the choropleth mapping, e.g. to plot country level or regional county level infection rates.

## List of Improvements

As suggested by Dr. Bradshaw, it would be good if I wrote a user guide to using the framework as currently, I am assuming someone trying to use the framework knows how it works or has had me tell them how it works. It would also be helpful for other users if I renamed public variables so that they were using snake case because this is the standard for python libraries and frameworks.

I believe it would be beneficial for creating a part of the framework that will allow users to easily swap hosts between the containers, like in my implementation where hosts are travelling between cities. This will make using the framework simpler as less user code would be required when creating an implementation.

For obtaining more realistic statistics, differential equations could be implemented into calculations for the number of interactions between hosts or for the disease infection chance. The model could also run multiple times and average the output to increase the accuracy of results.

Given more time, I think it would be beneficial to potential users of the framework to include more implementations of some of the modules such as plotting. This will allow the user to see how the framework is used through examples and potentially save them time by having an implementation that already works.

Dealing with hosts and vectors that are separate entities is a little annoying. In the framework, the host has both infected and infectious state which means that the hosts is also spreading the disease. To solve this the infectious state from the host could be moved into a separate vector class. However implementing this would mean the need to simulate the movement of a whole new object and will drastically increase the time taken for the simulation to run.

Plotting on the map is a script that must be run separately, this script could be run after the simulation however the library that is used for map plotting requires a different python interpreter to what the framework is running and as such it cannot be run at the same time. To fix this the framework could be rebuild in the same interpreter as the mapping library however if the user does not require plotting to a map, then this change would be meaningless.

It would be very beneficial to the framework if there was support for parallel processing when running timesteps on environments or when populating the containers and environments, however this is not possible to do when running the framework on Windows so I could not implement this. Given more time and access to a Linux machine, I could have implemented this by creating a new parallel thread for each item that will have timestep run on it so long as there is no shared data between the different containers. I have left comments in the code for places where this could be implemented.