# Homework 4 (Project 1 - Epidemiology)

**Due** Nov 20, 2017 by 11:59pm **Points** 100 **Submitting** a file upload
**File Types** py and zip **Available** Nov 1, 2017 at 12am - Nov 24, 2017 at 11:59pm 24 days

This assignment was locked Nov 24, 2017 at 11:59pm.

**ENGI E1006 - Intro to Computing for Engineers & Applied Scientists**

**Project 1**

**Due: 5:30pm on Monday, November 20th**
**Total Points: 100**

You are encouraged to discuss this project with other students but you must turn in your own work. Please review the academic honesty policy for this course (at the end of the **Course Syllabus**).

This project has multiple parts. You should work on these parts step-by-step, reading the description carefully. You should only move on once a part works correctly. **We will discuss the project in detail in class, so make sure to attend.** Start working on the project early, rather than trying to do everything last minute.

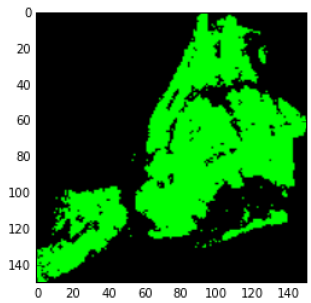
**Important: You must use Anaconda and Spyder for this project. We will not be able to help you debug other environments.**

## Overview

The purpose of this project is to simulate the outbreak of an infectious disease. This is a common problem in the field of epidemiology, which is the study of infectious diseases. While the simulation we will write is an extremely simplified version of actual models, it is based on some of the the same ideas.

We will simulate the spread of the disease on a map of New York City. Our map is a 150x150 grid. You can think of each cell (or pixel) of this grid as representing a number of individuals that live in that area. In step 1 of the project, you will read in map data from a file and display the map. The map will look something like this:

```
In [456]: m = read_map('nyc_map.csv')

In [457]: m.display()
```



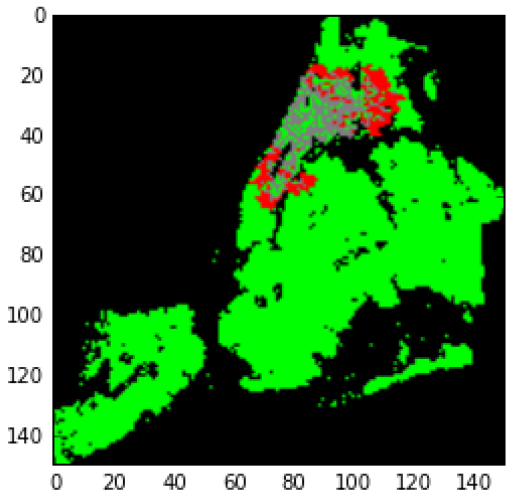

Our model is based on the so-called SRI model. At any time step, each cell on the map can be in one of three states:

- S (susceptible): The cell is healthy and not infected. Displayed green on the map.
- R (resistant): The cell cannot be infected (in our case, that means the cell is dead). Displayed gray on the map.
- I (infected): The cell is currently infected. Displayed red on the map.

When a cell is infected, it has the potential to infect neighboring cells, it might recover after a certain time (revert to state S), or it might die (move to state R). Initially all cells will be susceptible and only one cell will be infected. The details about how and when these changes occur are described below.

Time is measured in discrete steps. After a number of steps, the disease will have spread and infected a number of cells. Some may have recovered and others may have died. The map might look something like this:

```
In [452]: m.time_step()
```




### Step 1 (25 points) - Modeling and displaying the map

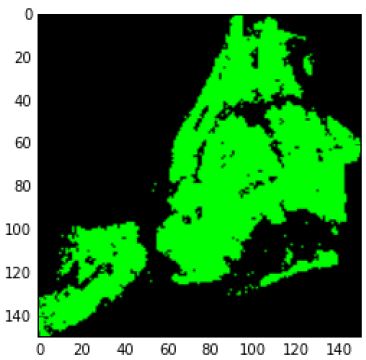
Start by downloading the file **simulator.py**, which contains skeleton code with some function and class definitions already in place.

We first need to create a model for each cell and then store the cells together into a map. We will use two classes: `Cell` and `Map`. `Cell` has attributes the x and y coordinate of each instance, as well as the current state of this cell as a string ("S"=susceptible, "R"=resistant, "I"=infected).

`Map` is going to be the main class that implements the simulator. The `Map` has attributes for height and width, as well as a dictionary `cells` that stores the actual cells on the map. The keys of this dictionary will be tuples of (x,y) coordinates, and the values will be Cell instances.

1. Write the method `add_cell(self, cell)` in the `Map` class. This method takes a cell object as its parameter and inserts it into the cells dictionary.
2. Write the function `read_map(filename)`. This function should reads in x,y coordinates from a file, create a new `Cell` instance for each coordinate pair. The function should return a `Map` instance containing all the cells. The file **nyc_map.csv** contains the coordinates for each cell of the 150x150 New York City map in comma separated value format.
3. Next, write the method `display(self)` in the `Map` class. To display the map, we will use **matplotlib** (a graph plotting package, which we will discuss in class). To plot the map, we first need to turn the data in the map into a suitable format, described below, and then call matplotlib to display it in the IPython console (this is why you need to use Spyder for this assignment). To display the map, you can use the line `plt.imshow(image)`. matplotlib requires the image to be in the following data format:
   - Each cell on the map will be displayed as one pixel. Each pixel is represented by a (Red,Green,Blue) tuple indicating the color of the pixel. The values for each color is a float ranging between 0 and 1. So (0.0, 0.0, 0.0) stands for a black pixel, (1.0, 0.0, 0.0) stands for a red pixel, (0.0, 1.0, 0.0) stands for a green pixel and (0.5, 0.5, 0.5) stands for a gray pixel.
   - A single row of pixels is simply a Python list of 150 of such pixel tuples.
   - The complete image is a Python list, containing 150 rows (i.e. the image is a list of lists).
   - Each cell should be displayed in the color that represents its state: green if the state is S, red if the state is I and gray if the state is R. Pixels in the image that do not correspond to a cell on the map should be displayed in black.
4. You should now be able to run your program, which will load the class and function definitions and then type the following into the IPython console:

```
In [456]: m = read_map('nyc_map.csv')

In [457]: m.display()
```



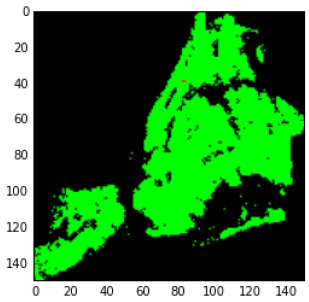

## Step 2 (30 points) - Making cells infectious

Next, we will work on the mechanism that allows cells to become infected and infect their neighbor cells. Any cell that is infected can infect its neighbors.

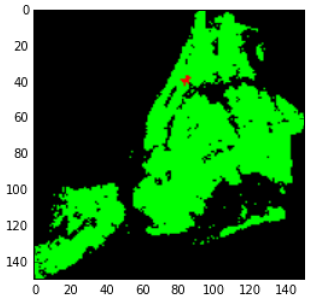
1. Add a method `infect(self)` to the Cell class. When infect is called, the state attribute should be set to "I".
2. We need to obtain a list of neighbor cells for a given cell. This cannot be a method of of `Cell` because each `Cell` instance is unaware of its neighbors. We therefore need to add a method to `Map` and pass the cell's coordinates. Write the method adjacent_cells(self, x, y) that returns a list of cell instances that are adjacent to coordinate (x,y). Adjacent means that the cell could be to the north, east, south, or west. Diagonal cells do not count as adjacent. Pay attention to the boundary of the map. There cannot be any cells outside of the map area.
3. Next, write the method `process(self, adjacent_cells) in the Cell` class. adjacent_cells is a list of cell instances (as returned by the `adjacent_cells(...)` method of Map.). The process method is only relevant if the status of the cell is "I". In all other cases it can return immediately. The method should look at each of the adjacent cells. If the adjacent cell is in state "S", it can become infected. For each such cell, we decide randomly if an infection happens or not. The file contains a variable `virality` (near the top of the file). This variable stores the probability that a cell infects an adjacent cell. Use the function `random.random()` (in the random module) to obtain a random float between 0.0 and 1.0. If that number is <= the virality, call the `infect()` method of the neighboring cell.
4. Finally, we need to be able to process all cells on the map. Our simulation uses discrete time steps. Write the method `time_step(self)` in the `Map` class, which evaluates such a time step. The method should simply call the `process()` method on each of the cells on the map and finally call display() to display the new state of the map.

You should now be able to infect a cell on the map and see the disease spread on each time step.

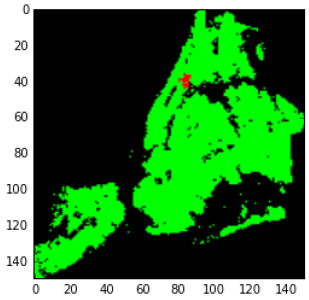
```
In [468]: m = read_map('nyc_map.csv')

In [469]: m.cells[(39, 82)].infect()

In [470]: m.time_step()
```



```
In [471]: m.time_step()
```



```
In [472]: m.time_step()
```



# Step 3 (20 points) - Recovery

We would also like our cells to be able to recover from the infection. In our model, we will recover after a certain number of time-steps. The variable `recovery_time` (near the beginning of the file) specifies after how many time steps a cell recovers. When a cell recovers, it's state moves from "I" to "S". Modify the `Cell` class and the `process` method to allow recovery. Hint: add a counter to remember how long each cell has been in state "I".

### Step 4 (25 pts) - Mortality

Finally, some cells may not recover from an infection and will die instead. Our model assumes that an infected cell dies with a certain probability in each time step. The probability depends on the number of time steps that the cell has been infected. The probability of dying, giving the number of time steps, is normally

distributed. So, for example, a particularly deadly disease might kill most people on time step 3 after becoming infected. Some people might die earlier and some might survive longer (or long enough to recover).

The function `pdeath(x, mean, stdev)` models the mortality of a cell using the normal distribution (this has already been implemented). You can think of it as returning the probability that an infected cell dies at time step x. The mean represents the average time step at which people typically die. Modify the `process(self)` method. Use the `random.random()` function to obtain a random float between 0.0 and 1.0 and use it to decide if the cell dies or not. Note: The order in which each cell is processed matters. When a cell is processed, first decide if the cell recovers, then if it dies. Then, if it is still infected, proceed to infect adjacent cells, as described above.

**Step 5 - Experiment!**

Finally, modify the parameters of the model (virality, recovery time, mean and standard deviation for the normal distribution) and see how the disease spreads. Infect any cell, as in the example above, and repeatedly call time_step() to advance the simulation. The most interesting (and horrible) part about modeling deadly infectious diseases is to study the interaction between mortality and virulence. A particularly deadly disease may kill the hosts before they can infect others, thereby preventing the disease from spreading further (this has happened in the case of Ebola epidemics). Many diseases are usually harmless but may be extremely virulent (such as the flu or common cold), or hosts can stay infectious for a long time (for example, herpes). Changing the parameters a little bit can have devastating consequences!