Classes Challenge 40: Epidemic Outbreak GUI App

Description:
You will be responsible for writing a program that simulates the spread of an infectious disease throughout a population similar to the previous program. Using classes, you will model an individual person's and an entire population's attributes and behaviors. Your program will allow users to set various initial conditions in regards to the infection such as population size, infection rate, mortality rate, and infection duration. The program will then simulate the interaction of people within a population and spread the disease. Each iteration of spreading the disease will result in a summary displaying statistics of the population. This time however, rather than storing the population in a list and checking if the person to the left or right of the current person is infected, we will store the population in a two dimensional list using nested for loops. This will allow us to check for infections both to the left and right and above and below. This added feature will help allow the program to create a Graphical User Interface or GUI (goo-e) to visually show the spread of the infection. Rather than representing the spread of the infection using O, I, and X in the terminal, each person in the population will be represented by a color square in a GUI; green being healthy, yellow being infected, and red being dead.

Step BY Step Guide:
Defining your classes

- Define a class Simulation
- Define a class Person
- Define a class Population

Defining your class methods
The Simulation Class

- Define an __init__() method for the Simulation class, which takes one parameter, the required self parameter.
    - Initialize an attribute called day_number and set it equal to 1.

    - Print a message about needing to know population size.
    - Get user input for the population size.
    - For visual purposes, we only want user input that is a perfect square.
    - Create a variable root and set it equal to the square root of the given population size.
        - Type import math as the first line of code in your program.
    - If int(root + .5)**2 is not equal to the given population size, then you do not have a perfect square:
        - The reason is that the int function will round the decimal value. If we add .5 to the root, unless it is a perfect square, it will always be rounded up and the condition will hold.
            - For example, if the user entered in 79, the square root of 79 is 8.8881.
            - Adding 0.5 to that yields 9.3881.

            - Taking the int of that yields 9.
            - Squaring that yields 81.
            - Since 81 is not equal to 79, we must change the simulations population_size attribute..
        - Round root to 0 decimals.
            - In our example, the user entered 79.
            - Taking the root yields 8.8881.
            - Rounding this to zero decimals yields 9.
        - Initialize an attribute called grid_size and set it equal to the integer value of root.
        - Set the attribute population_size equal to the grid_size attribute squared.
            - This will be used to create an NxN grid. In our previous example, the grid would be 9x9 for 81 people in the population.
        - Print a message informing the user that the program is rounding the population to the given size for visual purposes.
    - Else, the given population is a perfect square:
        - Initialize an attribute called grid_size and set it equal to the square of the population_size attribute.
    - Store the response in an attribute called population_size.

    - Print a message about needing to know the percentage of the population initially infected.
    - Get user input for the starting infection size.
        - Divide this value by 100 to turn it into a percentage.
    - Store the response in an attribute called infection_percent.

    - Print a message about needing to know the probability that a person will get infected if they come in contact with the disease.
    - Get user input for the infection probability.
    - Store the response in an attribute called infection_probability.

    - Print a message about needing to know how long the infection will last.
    - Get user input for the infection duration.
    - Store the response in an attribute called infection_duration.

    - Print a message about needing to know the mortality rate of those infected.
    - Get user input for the mortality rate.
    - Store the response in an attribute called mortality_rate.

    - Print a message about needing to know how long to run the simulation for.
    - Get user input for the number of days to simulate.
    - Store the response in an attribute called sim_days.

The Person Class

- Define an __init__() method for the Person class which takes one parameter, the required self parameter.
    - Initialize an attribute called is_infected and set it equal to False.
    - Initialize an attribute called is_dead and set it equal to False.
    - Initialize an attribute called days_infected and set it equal to zero.

- Define an infect() method for the Person class which takes two parameters, the required self parameter, and a simulation object.
    - Create a random integer from 0 to 100 to represent the chance this individual person becomes infected.
        - Type import random as the first line of code in your program.
    - If this value if less that the simulations infection_probability attribute:
        - Infect the person by setting their is_infected attribute to True.

- Define a heal() method for the Person class which takes one parameter, the required self parameter.
    - Set the persons is_infected attribute to False.
    - Set the persons days_infected attribute to 0.

- Define a die() method for the Person class which takes one parameter, the required self parameter.
    - Set the persons is_dead attribute to True.

- Define an update() method for the Person class which takes two parameters, the required self parameter and a simulation object.
    - First, check that the person is not dead. If the person is not dead:
        - Check if they are infected. If the person is infected:
            - Increase their days_infected attribute by 1.
            - Create a random integer from 0 to 100.
            - If this integer is less than the simulations mortality_rate attribute:
                - Call the persons die() method to kill the person.
            - Elif, the person didn't die, check to see if they can be healed. If the persons days_infected attribute is equal to the simulations infection_duration attribute:
                - Call the persons heal() method to heal the person.

The Population Class

- Define an __init__() method for the Population class, which takes two parameters, the required self parameter, and a simulation object.
    - Initialize an attribute called population and set it equal to a blank list. This list will represent a two dimensional list or array.
        - This list will be a list of N lists.
        - Each list within the list will represent a row in an NxN grid.
        - Each element of the row will represent a Person object.
        - Each of these lists will hold N Person objects.

    - Use a for loop to loop through the needed number of rows. This information is stored in the simulation objects grid_size attribute. For each iteration of the loop you should:
        - Create a variable called row and set it equal to a blank list.
        - Loop through the needed number of Person objects. This information is stored in the simulation objects grid_size attribute. For each iteration of the loop you should:
            - Create a person object.
            - Append the person to the row.
        - Append the completed row to the Populations population attribute.

- Define an initial_infection() method for the Population class, which takes two parameters, the required self parameter, and a simulation object.
    - Create a variable called infected_count. This will represent the number of people who must start infected based on the user's initial conditions.
    - To determine this value, multiply the simulation objects infection_percent and population_size attributes together.
        - Round this value to zero decimals.
        - Cast this value to an integer so we can use it in a for loop.
    - Create a variable infections and set it equal to 0.
    - While infections is less than infected_count:
        - Create a random integer x from zero to the simulations grid_size attribute - 1.
        - Create a random integer y from zero to the simulations grid_size attribute - 1.
        - self.population[x][y] represents an individual person in the two dimensional list.
        - If this given person is not infected:
            - Set the is_infected attribute to True for the current person in the population.
            - Set the days_infected attribute to 1 for the current Person in the population.
            - Increment the infections variable by 1.

- Define a spread_infection() method for the Population class which takes two parameters, the required self parameter, and a simulation object.
    - Use a numerical for loop to loop through the simulations grid_size attribute. This represents each row of the grid. Each iteration you should:
        - Use a numerical for loop to loop through the simulations grid_size attribute. This represents an individual Person object in a row. Each iteration you should:
            - Check if the is_dead attribute is False for the current person in the population, meaning they are currently alive. If they are alive we will check to spread the infection.
            - If i is equal to 0:

        - This is the first row in the population list, so you cannot check rows above.
        - If j is equal to 0:
            - This is the first person in the first row. You cannot check to the left.
        - Elif j is equal to the simulation objects grid_size attribute - 1:
            - This is the last person in the first row. You cannot check to the right.
        - Else:
            - You can check to the right, left, and below.
    - If i is equal to the simulation objects grid_size attribute - 1:
        - This is the last row in the population list, so you cannot check rows below:
        - If j is equal to 0:
            - This is the first person in the last row. You cannot check to the left.
        - Elif j is equal to the simulation objects grid_size attribute - 1:
            - This is the last person in the last row. You cannot check to the right.
        - Else:
            - You can check to the right, left, and above.
    - Else:
        - i represents any row other than the first or last row in the grid.
        - You can check to the left, right, above, and below.
        - If j is equal to 0:
            - This is the first person in a row. You cannot check to the left.
        - Elif j is equal to the simulation objects grid_size attribute - 1:
            - This is the last person in a row. You cannot check to the right.
        - Else:
            - You can check to the right, left, above, and below.
- Recall, an individual Person object in this function is represented by self.population[i][j].
    - To check if the person to the right is infected:
        - self.population[i][j+1].is_infected
    - To check if the person to the left is infected:
        - self.population[i][j-1].is_infected
    - To check if the person above is infected:
        - self.population[i-1][j].is_infected
    - To check if the person below is infected:

                - self.population[i+1][j].is_infected
            - Make the appropriate checks based on the position of the current Person object in the loop. If any of the adjacent Person objects in the NxN grid are infected, call the current persons infect() method.
                - self.population[i][j].infect()

- Define an update() method for the Population class which takes two parameters, the required self parameter, and a simulation object.
    - Increase the simulation objects day_number attribute by 1.
    - Loop through the population list. Recall that each element in this list is another list that represents a row in an NxN grid. For each iteration you should:
        - Loop through the elements of the row. For each iteration you should:
            - Call the Persons update() method.

- Define a display_statistics() method which takes two parameters, the required self parameter, and a simulation object.
    - Create a variable total_infected_count and set it equal to 0.
    - Create a variable total_death_count and set it equal to 0.
    - Loop through the population list. Recall that each element in this list is another list that represents a row in an NxN grid. For each iteration you should:
        - Loop through each element of the row: For each iteration you should:
            - Check if the current Person is infected. If they are:
                - Increment total_infected_count by 1.
                - Check if the current Person is dead. If they are:
                    - Increment total_death_count by 1.
    - Create a variable infected_percent and calculate the percentage of the population that is infected.
        - Round this value to 4 decimals.
    - Create a variable death_percent and calculate the percentage of the population that is dead.
        - Round this value to 4 decimals.
    - Print a summary of the population statistics for the current day of the simulation.
        - This should include the day number.
        - The percentage of the population infected.
        - The percentage of the population that is dead.
        - The total number of people infected.
        - The total number of deaths that have occured.
        - See example output for formatting.

Defining your functions

- Define a function graphics() which is not associated with any Class. This function should have three parameters, a Simulation object, a Population object, and a canvas. This is a helper function that will draw squares, that represent a Person object, onto a Tkinter canvas.

- Create a variable square_dimension and set it equal to 600 divided by the simulation objects grid_size attribute.
    - Make sure to use floor division //.
    - I am using 600 because that is the size of the overall GUI window 600 x 600. If you decide to change your window size, change this value.
- Use a numerical for loop to loop through all of the possible rows of the grid. This is represented by the simulation objects grid_size attribute. For each iteration:
    - Define a variable y and set it equal to i*square_dimension.
    - Use a numerical for loop to loop through all of the possible people in each row of the grid. This is represented by the simulation objects grid_size attribute. For each iteration:
        - Define a variable x and set it equal to j*square_dimension.
        - If the Person object stored at index i, j in the Population object is dead:
            - Draw a red rectangle onto the Tkinter canvas.
            - canvas.create_rectangle(x, y, x+square_dimension, y+square_dimension, fill="red")
        - Else:
            - If the Person object stored at index i,j in the Population object is infected:
                - Draw a yellow rectangle onto the Tkinter canvas.
                - canvas.create_rectangle(x, y, x+square_dimension, y+square_dimension, fill="Yellow")
            - Else:
                - The person is healthy.
                - Draw a green rectangle onto the Tkinter canvas.
                - canvas.create_rectangle(x, y, x+square_dimension, y+square_dimension, fill="green")

The main code

- Create a Simulation object.

- Create a constant variable called WINDOW_WIDTH and set it equal to 600.
- Create a constant variable called WINDOW_HEIGHT and set it equal to 600.

- Create a Tkinter window and canvas.
    - Tkinter is a library which allows users to make a graphical user interface (GUI) for their python applications.
    - Type import tkinter as the first line of code in your program.

- Create a tkinter object, set it as our window, and give the window a title.
    - sim_window = tkinter.Tk()
    - sim_window.title("Epidemic Outbreak")

- Create a canvas object that we can draw on and assign it to the created window.
    - sim_canvas = tkinter.Canvas(sim_window, width=WINDOW_WIDTH, height=WINDOW_HEIGHT, bg="lightblue")
    - sim_canvas.pack(side=tkinter.LEFT)

- Create a Population object.

- Call the Populations initial_infection() method.
- Call the Populations display_statistics() method.
- Prompt the user to press enter to being the simulation.

- Use a for loop to simulate each day of the simulation. For each iteration you should:
    - Spread the infection by calling the Populations spread_infection() method.
    - Update the population by calling the Populations update() method.
    - Display the statistics for the current day by calling the Populations display_statistics() method.
    - Show the graphics of the data by calling the graphics() function.
    - Use tkinters built in update() method on to update the window.
        - sim_window.update()
    - If you are currently not on the last day of the simulation:
        - Use tkinters built in delete() method to erase the previously draw information on the given canvas.
            - sim_canvas.delete("all")

Example Output:

To simulate an epidemic outbreak, we must know the population size.
---Enter the population size: 9700
Rounding population size to 9604 for visual purposes.

We must first start by infecting a portion of the population.
--Enter the percentage (0-100) of the population to initially infect: 3

We must know the risk a person has to contract the disease when exposed.
--Enter the probability (0-100) that a person gets infected when exposed to the disease: 15

We must know how long the infection will last when exposed.
--Enter the duration (in days) of the infection: 5

we must know the mortality rate of those infected.
--Enter the mortality rate (0-100) of the infection: 10

We must know how long to run the simulation.
--Enter the number of days to simulate: 365

-----Day # 1-----
Percentage of Population Infected: 2.9988%
Percent of Population Dead: 0.0%
Total People Infected: 288 / 9604
Total Deaths: 0 / 9604
Press enter to begin simulation.

-----Day # 2-----
Percentage of Population Infected: 4.7168%
Percent of Population Dead: 0.5831%
Total People Infected: 453 / 9604
Total Deaths: 56 / 9604

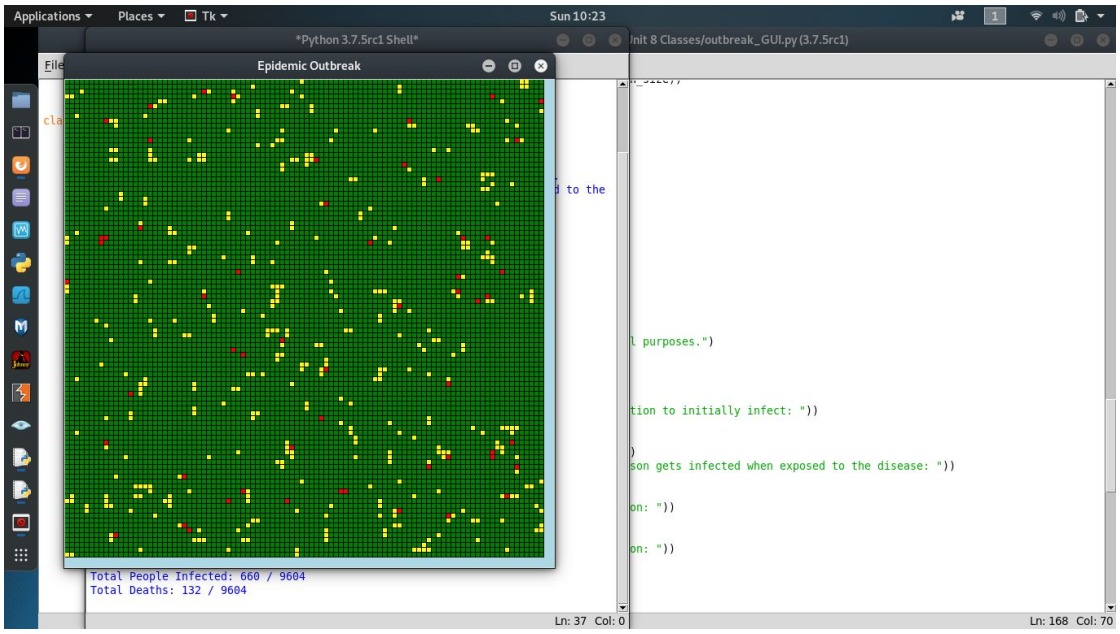
-----Day # 3-----
Percentage of Population Infected: 6.8721%
Percent of Population Dead: 1.3744%
Total People Infected: 660 / 9604
Total Deaths: 132 / 9604

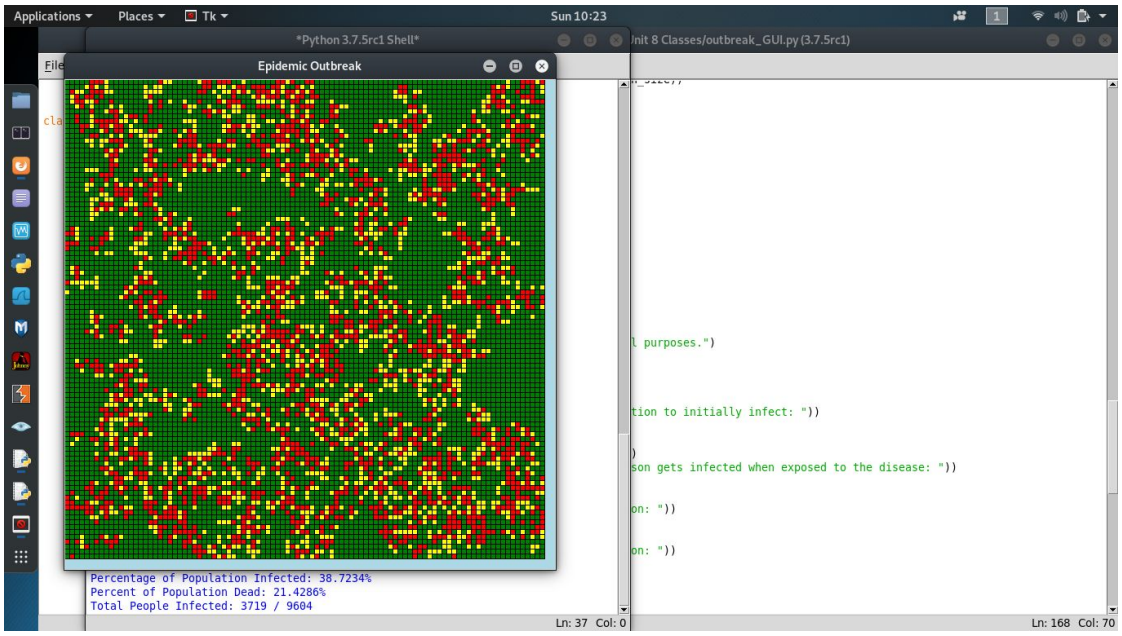
-----Day # 4-----
Percentage of Population Infected: 9.5377%
Percent of Population Dead: 2.2178%
Total People Infected: 916 / 9604
Total Deaths: 213 / 9604

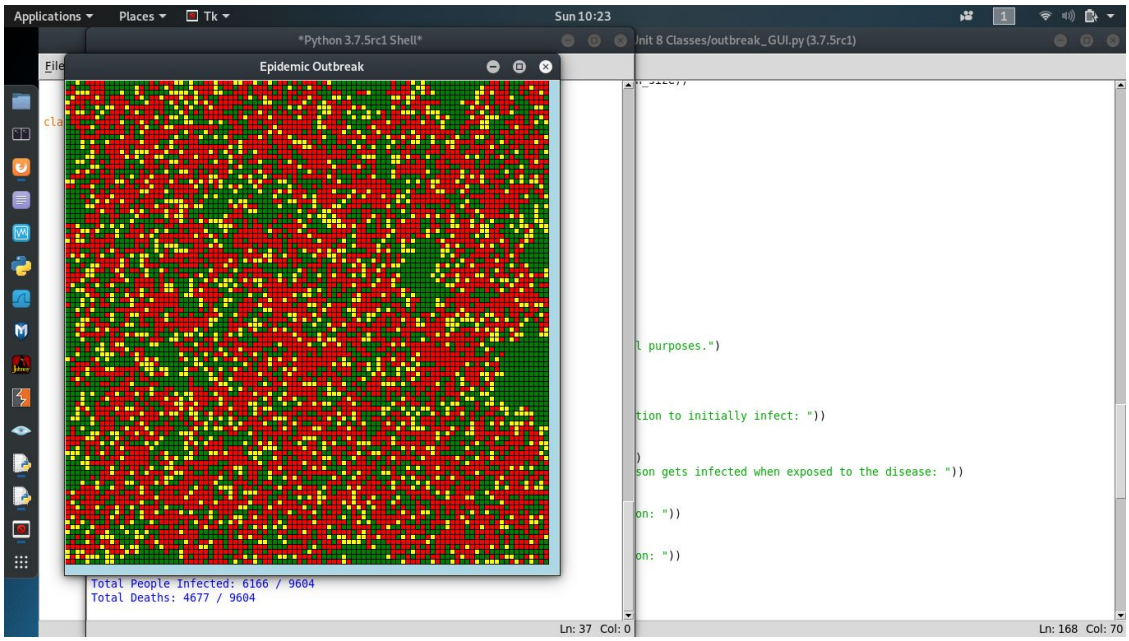
-----Day # 5-----
Percentage of Population Infected: 10.7039%
Percent of Population Dead: 3.2799%
Total People Infected: 1028 / 9604
Total Deaths: 315 / 9604

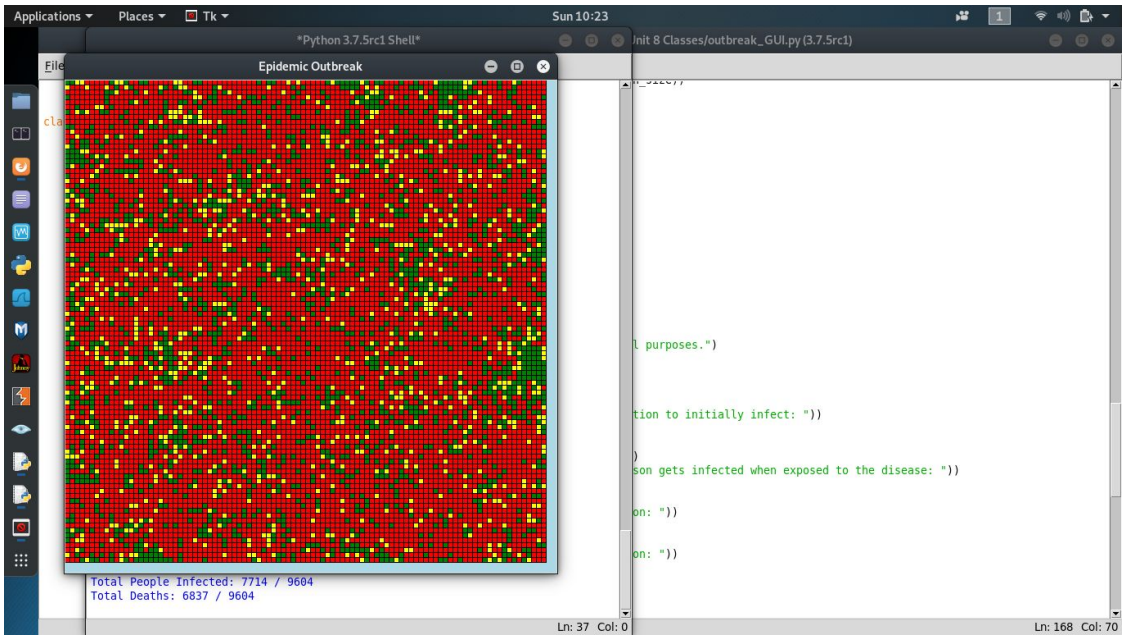
****CUT FOR BREVITY****

Example Screenshots:






Applications
Places
Tk
Sun 10:23
Python 3.7.5rc1 Shell
Epidemic Outbreak
Total People Infected: 6166 / 9604
Total Deaths: 4677 / 9604
Ln: 37 Col: 0
Ln: 168 Col: 70


Applications
Places
Tk
Sun 10:23
Python 3.7.5rc1 Shell
Epidemic Outbreak
Total People Infected: 7714 / 9604
Total Deaths: 6837 / 9604
Ln: 37 Col: 0
Ln: 168 Col: 70