User Guide:

Overview of Program:

The *diseaseSim.py* is a program that simulates the spread of disease though a population. It also includes antidotes and dying capabilities. As expressed in the *README* file, the program can be run either with command line arguments or without it, which will run the default values. Exceptions were used to control the flow of the program. This prevents other users who don't know now how to use the code from using it incorrectly.

More Detailed View:

Neighbourhood.py File:

The program is split into multiple different files to break up the code to make it more readable and extendable. The program has the ability to use Moore or Von Neumann neighbourhoods. The moore function found in the *neighbourhood.py* file takes advantage of its similar code with the vonNeumann function and reuses it.

IO.py File:

This file contains a very useful function and is in a seperate file so the code contained in it can be reused. If I implemented any fileIO operations I would have stored them in here as well.

The setUpEnvironment function creates a new directory called *Plots*, if it doesn't already exhists. Then it creates a new directory with a unique name (unqiue from timestamp) for all the plot images to be saved into so it can be contained. This makes it easy to identify which plots came from which simulation run.

Plots.py File:

Contained code that has been given to us to do the assignment. I have extended the plotGrids to plot not only the infected and uninfected cells, but also the world/barrier and any cells that may have died.

If the user had chosen that they don't want the plots to go to the screen, but just to the file as created from the IO.py file the *plt.show()* function call would never occur.

I also created another graph to show the number of occurrences for infections, antidotes given, and cells killed.

Validation.py File:

I created a seperate file to do all the input validation. This means that the code inside can easily be extended or reused. This also reduced the amount of code in my *diseaseSim.py* file. I created a validateGreater function that was reused 5 times, to check each command line argument that was a number.

WolrdOps. py File:

This file also created some code that was given to us to complete the assignment. I created a seperate function to initialise the world boarders. This reduced the complexity inside my *diseaseSim.py* file. If I ever wanted to change the boarders I can go to this function and modify them directly. With the use of comments and slicing I simplified the confusing code, making it more readable.

I had to update the *movePeeps* and *distribute* functions to not only check if the cell was trying to move out of bounds, but to also check they weren't trying to move into the barrier/world.

In the *kill*, *infect*, *heal* functions I passed a variable called **NEIGH_FUNC** which is basically a function "pointer" to either the Moore function or the Von Neumann function. This saved checking which function to use every time these functions were called.

DiseaseSim.py File

This file is the main that runs all the other code. It checks the number of command line arguments to see if the user wants to use their own values or the default ones. Taking advantage of the **validation.py** file and functions, it checks the command line arguments, if any were given. If any command line argument isn't valid, the program exists gracefully and tells the user what invalid argument they entered. Otherwise, everything is valid and it sets the **NEIGH_FUNC** variable to either the Moore or the Von Neumann function in the **neighbourhood.py** file.

After that it initialises and sets up all the arrays that contain the cells and boarders. From there the actually simulation runs. For each time step it runs though the events possible (**die, heal, infect**).

At the end, the total counts are displayed to the user in the terminal and in a seperate plot for a more stimulating effect.

The plots can also be seen in the **Plots folder**.

Usage - Directly & Parameter Sweep:

- diseaseSim.py:
  - `$ python3 diseaseSim.py`
  - `$ python3 diseaseSim.py V 100 60 15 15 10 Y [0.60 0.15]`
  - Note: [] are optional parameters

- disease_sweep.sh:
  - `$ ./disease_sweep.sh M 100 50 5 5 15 5 15`