Programming for Social Science: Bacterial Bomb Model

https://github.com/sickotra/Bacterial_Bomb.git

Shivani Sickotra

November 22, 2020

Contents

1 Program Context and Aims

A deadly biological weapon has been detonated in the middle of the town on top of a building. As a member of a secret government anti-terrorist unit, this program has been built to model the spread of the contagious bacterial particles. This model will trace the bacteria that is released to give an estimate of the end locations, in order for the contamination to be dealt with.

The program has been built to accomplish the following:

1. Pull in a data file to find out the bombing point
2. Calculate where the bacteria end up
3. Draw a density map of where all the bacteria end up as an image and display it on the screen
4. Save the density map to a file as text

2 Running the Program

To run the program, first the required packages will need to be installed. The packages needed for this program are:

- `csv`
- `random`
- `matplotlib`
- `pandas`
- `seaborn`
- `time`
- `tkinter`
- `runpy`
- `os`
- `sys`

The packages that the user may not have installed are `pandas` and `seaborn`. These are used to create the density map of the particles after they have been spread.

The script file that the user should run is the `bact_bomb_gui.py` file. This will open the GUI and the user can then run the program from there, explained in section 4.1.

The `model_spread.py` script can also be run independently from the anaconda command prompt/terminal. The number of particles and probability that in any given second, the current wind will blow a particle North, East, South or West can be set. To run the program the following can be entered in the command prompt:

```
python model_spread.py arg1=5000 arg2=75 arg3=5 arg4=10 arg5=10,
```

where `arg1` is the number of particles and `arg2`-`arg5` are the wind-based probabilities.

The default arguments shown above will be used if none are specified by the user. Only integer values should be used as arguments and the wind-based probabilities should have a sum of 100%. Error messages will inform the user if the inputs are not suitable.

The program can also be run from a Jupyter Notebook shown in section 4.2. From here the number of particles and other probabilities that will be used in the program can be chosen, rather than use the default values.

Note: The program has been written using Python version 3.8.3 and so a version of Python 3 should be used to run the code.

3 Program Features

3.1 ABM

The program has adopted a Object Oriented Programming approach and has a Agent Based Model (ABM) structure. The `particles` are the agents and the `town` is the environment.

- The particles are object instances from the `Particles` class in `particle_framework.py`. All the particle attributes and behaviours are defined in this class.
- The particles have an `x` and `y` co-ordinates that are initially identical since each particle must start at the location the bomb is detonated. The particles also all have an initial height set as 75m, the same height as the building.
- In this ABM, the particles interact only with the town and not with each other. Each particle has access to the `town`, but also the other particles for any future development, see section 6. The variables `x, y, height` as well as `particles` and `town` have all been protected.
- The town data in which the particles exist is read in from the raster file `wind.raster`. This town has the bomb location marked, but the program has the feature to locate the bomb point itself. The location is marked with a red diamond.
- The program then uses wind-based probabilities to determine where the particles will end up. The `p_east, p_west, p_north, p_south` variables set the percentage probabilities that a particle will be moved by the wind either East, West, North or South respectively. The `spread` method in the `Particle` Class uses a random walk to allow this behaviour.
- The building that the bomb is located has a height of 75m and there are turbulence probabilities to determine the height of the particle at a given second. The `p_rise, p_same, p_fall` variables set the percentage probabilities that a particle will either rise each second by 1m, stay at the same level or fall respectively due to turbulence. Once a particle is below the height of the building, turbulence ceases and it is set to fall each second by 1m, until it has reaches the ground and settles. The `turbulence` method in the `Particles` Class uses a random walk to allow this behaviour.

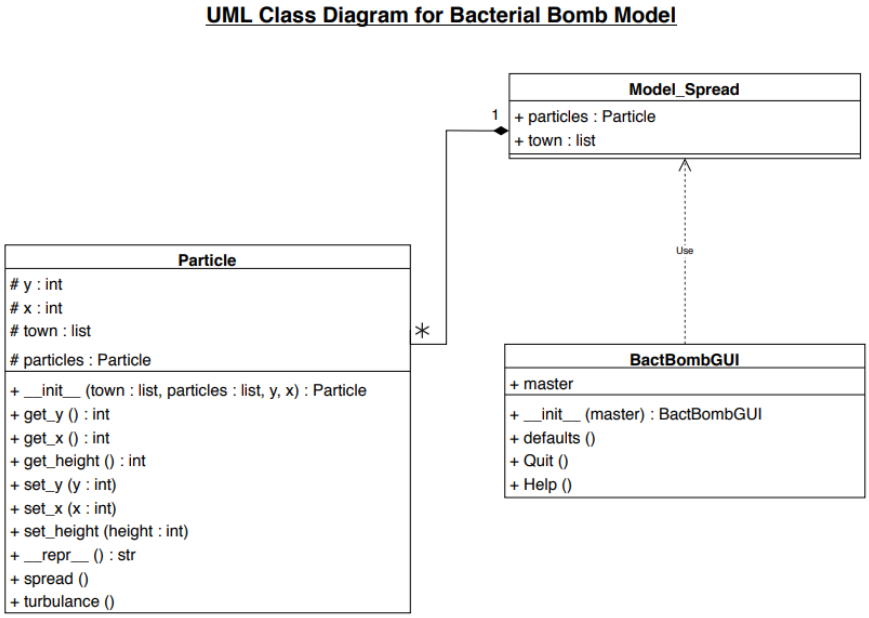

Figure 1: A UML class diagram showing the classes in the program, detailing their attributes, methods and their relationship with each other.

3.2 Outputs/Visualisations

There are several outputs and visualisations created when the program is run. These will either be displayed on screen or be saved in the *outputs* folder:

- *Fig 1*: A plot to show the bacterial particle locations after they have been spread will be displayed on screen. The original bomb location is marked by a red diamond and so it is easy to see the particles locations in relation to their starting point.

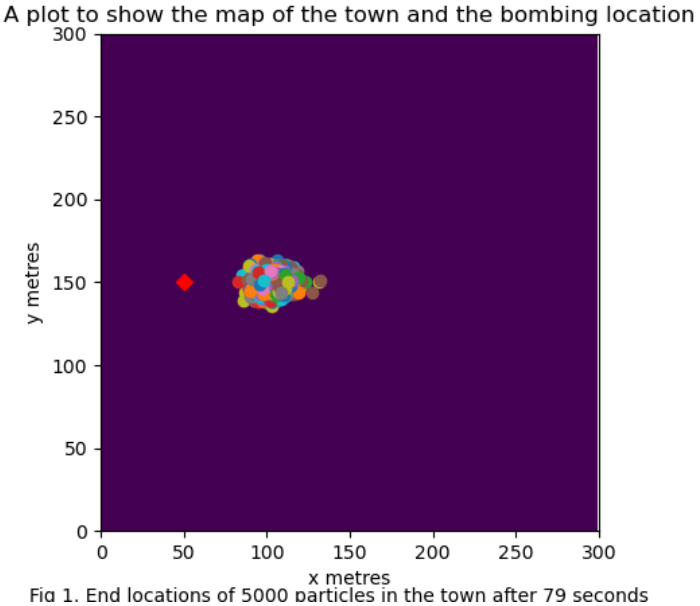

Fig 1. End locations of 5000 particles in the town after 79 seconds

- *Fig 2* or *density_map.png* : An image showing the particle locations and their density. The map shows denser areas in a darker colour than less dense.

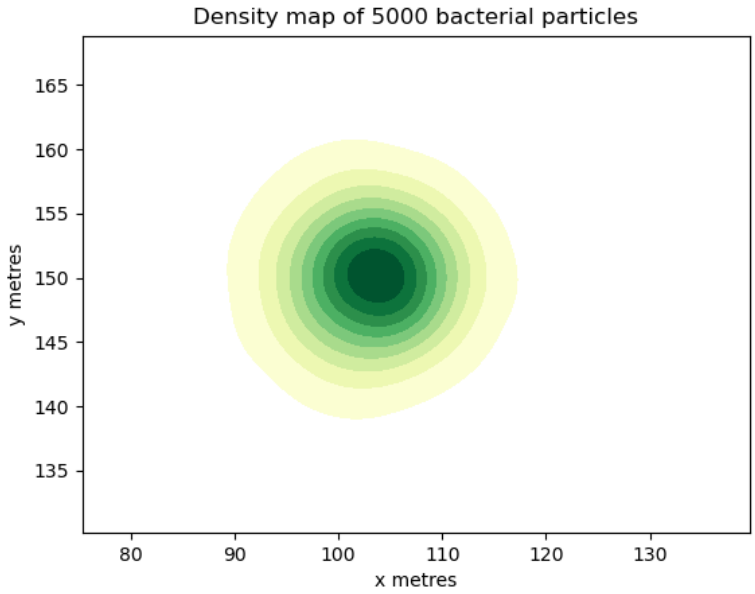

Figure 2: The plot showing the density of 5000 particles with the default wind-based and turbulence probabilities

- *end_locations.txt* : A text file containing all of the end co-ordinates of the particles after spreading.
- *density_map_text.txt* : A text file that represents the density map. The pixels with the value `0` have no particles at that location. If thirty particles are present at a given location, then the corresponding pixel will have a value of `30`.

4 User-facing Features

The two main aspects created, with consideration to the end-user, is a GUI and a Jupyter Notebook. The user can also read about the program and access the GitHub repository link through the website that has been created, `https://sickotra.github.io/`. The code for the website has been updated to include a `CSS` folder.

4.1 Graphical User Interface (GUI)

The GUI that has been created features a button that can be used to run the program. There is information about the parameters that will be used when the button is clicked. The user also has the ability to access additional help using a *Help* button which opens up a text file, and the ability to close the window using a *Quit* button.

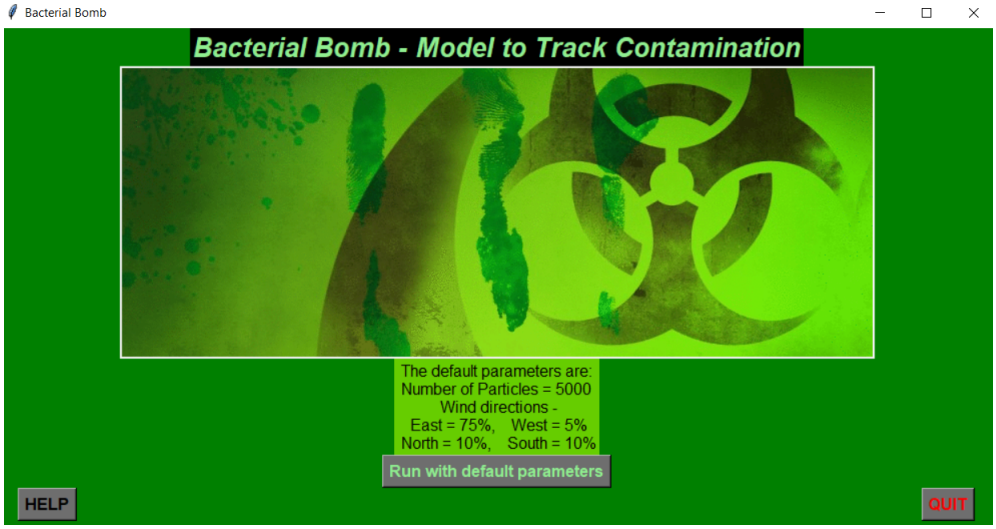

Figure 3: GUI for the Bacterial Bomb program

In order to create this GUI, an Object Oriented approach was taken again. The `BactBombGUI` Class in the `bact_bomb_gui.py` file creates the items to be displayed in the main window. Methods created in this class are used as the commands to run when buttons are clicked. This Object Oriented approach meant that the code was better organised and easier to understand.

4.2 Jupyter Notebook

A Jupyter Notebook, *BactBomb_sickotra.ipynb* has been created to explain the code further. The notebook is interactive as the number of particles can be controlled using a slider. The wind-based probabilities can also be set using sliders. The figures and outputs that are generated from the rest of the notebook code reflect these choices.

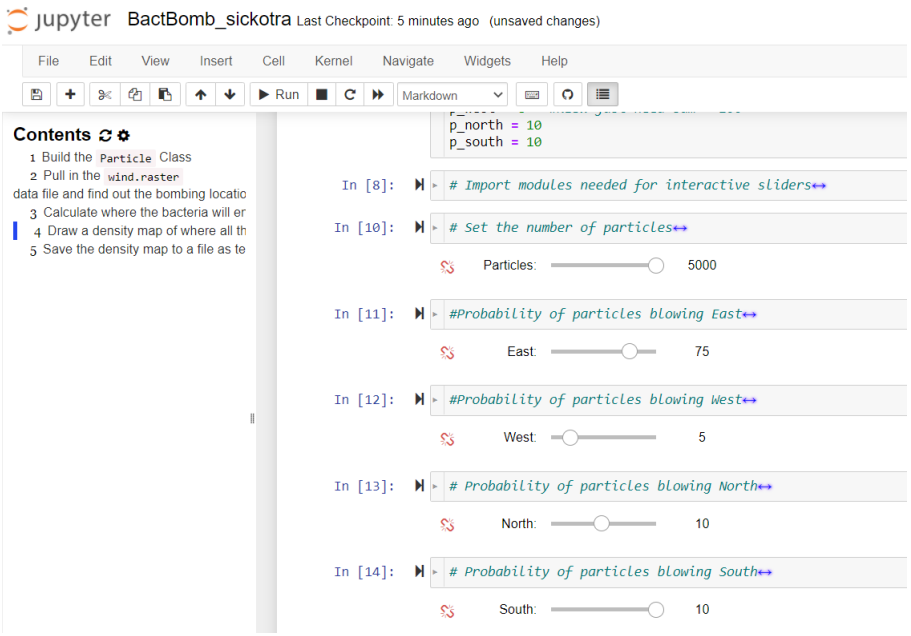

Figure 4: Jupyter notebook section where sliders have been added

4.3 Sphinx Documentation

Sphinx has been used to generate HTML web pages that display the documentation for all the Classes, methods and functions. The web pages are located in the `docs/_build` folder. The pages created makes the documentation easier to access and read for the user.

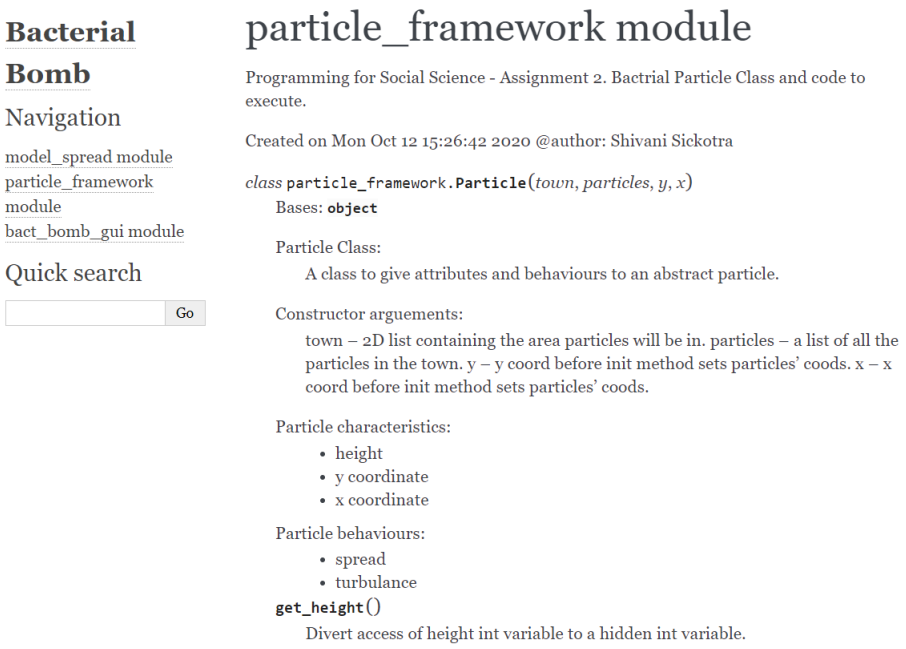

Figure 5: Sphinx generated HTML page for the `particle_framework` module

5 Software Development Process

5.1 Methodology

The software development process that was used when building the program was the "Iterfall" model, a combination of "The Waterfall Process" and "Incremental/Iterative Development".

The Waterfall Process is linear and includes:

1. **Requirement Analysis** - outlining the overall requirements of the program
2. **Design** - the specifications from step 1 are studied to create a plan/design
3. **Development/Coding** - the program is created
4. **Testing** - assessing the program for any errors
5. **Implementation** - the developed program is assessed by a user
6. **Maintenance** - fix any issues that may arise using patches or new releases

The Iterative process involves beginning with a small set of software requirements and building a program to meet those. Once this has been achieved, the program is then iteratively improved by repeating the process.

When building the Bacterial Bomb program, the two of these methods were combined naturally. The Waterfall process was suited as there was a clear set of requirements outlined by the project brief. This meant that there was a clear staring point once the specifications were understood. The next stage was designing the program, with the requirements in mind, the classes and functions needed were thought about. Pen and paper were used to brainstorm and a UML Class diagram was created to map out the structure of the program. From this led to the stage of writing the code for the program and then some unit tests were conducted. The code was then refactored and any commenting/documentation was 'cleaned up' or updated. The program was then shown to a user to ensure that the initial requirements had been met successfully.

The Iterative process then began, and the Waterfall method was repeated again. However, this time the requirements stage was open-ended and specifications for the GUI were outlined. The GUI was then designed by sketching and labelling the key components. The original UML was also updated. After this the code was written for the GUI and again shown to a user to explore the functionality. This software development process was well suited for building this program as it was an independent project and so approaches like the "Continuous Integration Method", where code is developed with other members in a group, were not appropriate.

5.2 Testing

Unit testing for the methods in the `particle_framework.py Particle` class are outlined in the *unit_testing.docx* document located in the *development docs* folder. The tests were conducted for the `spread` and `turbulence` methods in the class. The code that was used for the tests can be found in the `unit_tests.py` script file in the same folder.

5.3 Known Issues

The current known issues are:

- Docstrings for all `.py` scripts are not being generated. Only the docstrings included in the `particle_framework.py` script are being processed. The HTML pages corresponding to the other two scripts `model_spread.py` and `bact_bomb_gui` have been generated, however the docstrings are not being imported.
- The wind-based probability sliders in the Jupyter Notebook are not functioning correctly, unless the user ensures that all directions equal to 100%. Even then the slider maximum values do not appear as expected, but if care is taken to set the probabilities to add to 100, then the program runs with the specified directions. The solution to this issue may be to create the maximum slider values as a proportion of 100, this would mean that the 4 sliders max value would always equal 100%, and so the user can choose freely.
- The solid wall boundary used in the `spread` method in the `Particle` class is not very realistic. In a real town, bacterial particles would not be stopped by a 'solid wall' where the town ended, but would continue spreading with the wind. This boundary has been used to contain particles for this model, but it has to be noted that if a very high wind-based probability is set for just one direction, the model may not reflect real particle spread.

6 Future Development

There is scope for future development of both the core functionality of the program, as well as the user-facing features. Some improvements could include:

- Sliders/scales could be implemented to display on the GUI that allowed the number of particles and wind-based probabilities to be adjusted. Some code has been written to create the slider for the total particles and a button that would be used to run the program with the slider value. The code for this can be found in a branch called `development` of the main `Bacterial_Bomb` GitHub repository.
- A desktop icon could be created to run the GUI. This would be easier for the end-user as they would not have to run the program from a script file. To do this the `bact_bomb_gui.py` python file would need to be converted to `.exe`. An icon image can then be added to this. The GUI could also have additional features such as having the console output displayed live on the window when the program is run. A menu bar could also be added.
- A 3D animation showing the particles being spread could be created. The figures and density plot produced are 2D and unable to capture the rise and fall behaviour of each particle, so this would provide a better visualisation.
- A subclass from the `Particles` Class could be made for a second bomb at a different location in the town. This bomb could hold antidote particles to neutralise the bacterial particles from the original bomb. This would mean that the particles would interact with each other as well as the town. When a bacterial particle comes into contact with an antidote particle, it can be removed from the town. The model could find the best location to place the antidote bomb depending on where the bacterial particles have landed.