Final Project: Zombie Virus Modeling

1 Introduction

Computer science has so many applications beyond just cryptography and robotics. As we have learned through the past 2 years, we need to fight infectious disease with every field available. In particular, modelling can allow us to understand how a virus is spreading, the efficacy of different solutions, and inform public policy.

This project is all about modelling a fictional disease: particularly a zombie outbreak! You will be implementing a GUI (graphical user interface) that allows a school administrator to test different solutions after a zombie virus hits their school.

Note that all of this text isn't meant to scare you! This handout should be a guide/source of (hopefully) interesting tidbits. All of the instructions should be commented in your zombies_skeleton.py. If anything at all is confusing, please feel free to come to office hours and ask your questions or discuss your code! We have tried to provide an explanation for each line of the code, even if you aren't asked to code it yourself, and some of the packages we use might reference statistics concepts that you haven't seen. That's completely normal! It's important to note that python also has its own concept of 'black boxes' and that is the idea of packages: we import these packages and just know how to use them, not how they work. While I might not understand the graphics package, I can use it to print a zombie, and that's enough for this task. With that said, go forth and HAVE FUN!

2 Background

The SIR model, which stands for 'Susceptible', 'Infected', and 'Resistant' is a classic model used to show the spread of an infectious disease. Using a couple of parameters (namely, time to cure and number of people infected/day), the model uses Euler's method to solve for the expected impact of the disease. (You may remember Euler's method from calculus: it is when we know the derivative, and step-wise solve for the original line). But don't worry if this is all new to you, because we will be coding a simulation! Simulations are excellent for computational biology problems, because they let you see how minor changes can play out, given rough estimates of their effects.

In this project, you will be coding a zombie outbreak. While we have provided the graphics, you will code most of the guts of the program, and see how minor changes to things like school safety can affect your overall outcome.

3 The setup

In this lab, we are asking you to implement the computational 'guts' of the programs, as opposed to the GUI. While you are absolutely free to edit the graphics as you see fit, a preliminary 'zombie_graphics.py' is available for you to use. Download both these into the same folder. The rest of this sheet deals only with the zombie_skeleton file, and does not require understanding the graphics packages at all. If you have any issues throughout the program such as 'no module pygame' or the like, reach out to the TAs and we will help you import new modules.

4 Understanding the Setup

We have provided you a skeleton code, with some methods named and created, but not filled in. As you can see in the zombie_skeleton.py file, we are using two main classes, one called 'person' and another called 'school.' The person class is how we actually create a student, and it holds both the state of that student, as well as a method that says whether or not the student is infected after coming in contact with the virus. The school class has a list of students (of type Person), and has methods for running a round of the game, and also improving school health by doing things such as curing the student.

You should not have to understand anything about the zombie_graphics file, but please message a tutor or come to office hours if you see any error messages you don't understand, especially if they mention 'graphics'.

5 Implementing Person Class: __init__ method

As you have learned in this course, every time that we make an object from a class, we call the `__init__` method. As you can see, this `__init__` method takes in only one parameter ('state'), but binds several to the student. This is allowed!!

As you should be able to see in the zombies_skeleton_2.py file, you have two tasks in the `__init__` method. The first is to create a variable `num_rounds.` It's important that this variable is tied to each student, rather than an overall variable, so think about what that means. Each student is created in round 0, so set that variable to 0 initially. Secondly, we have to bind the 'state' variable to the student.

You'll also see that we have two lines of code written for you at the top of the method. These probably introduce some new terms for you! First of all, we see random.uniform.

This is from a package called random we imported. random.uniform takes two float arguments, and chooses a number between them, with each possible value having equal probability. As you can see in Figure 1, it's like an on/off switch, where we have an equal change of any value within the specified range.

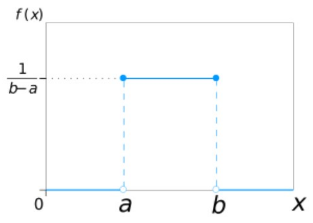

Figure 1: Example of uniform distribution. Source: wikipedia.org

When we use a '*' notation before a tuple, we are 'unpacking' that tuple into arguments. Essentially, it's a cleaner (or lazier!) way of writing something like `some_method(some_tuple[0], some_tuple[1], some_tuple[2], ...)`, because `some_method(*some_tuple)` is a lot simpler!

The second line might look a lot like the first, and that's because it is! Here we are calling `random.gaussian (*cure_time).` The only thing that changes is the *distribution* of values.

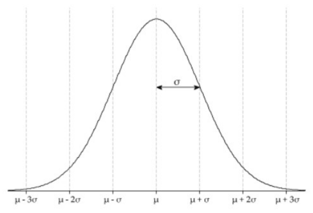

Figure 2: Example of Gaussian distribution. Source: amsi.org.au

Rather than a uniform distribution, the Gaussian distribution (also called normal distribution or a 'bell curve') is centered around a particular point.

The majority of the samples will be near the mean, and as we go further away from the mean, we see those values less often. The gaussian distribution doesn't take in a lower and upper bound, but rather two variables: μ (mu) and σ (sigma). σ is the first variable in the tuple, which is just the mean of the distribution. μ is the standard deviation, or spread of the data. For now, we have provided the tuple (σ=10, μ=4), but feel free to edit those variables and see how they affect the simulation.

tl;dr: Write two lines of code, one that stores a variable called num_rounds for each student, and another that stores the state of each student.

6 Implementing Person Class: Simulating an Interaction

The second method we have in the person class is the `interact` method. This method takes in no arguments except for self (the student it's being applied to), and returns a Boolean: True if the student is infected, False otherwise. Using another implementation of the random class, `viral_strength` is a random float between 0 and 1. You have two tasks. First of all, if the student is Resistant, we should always return False. Secondly, if the student is NOT resistant (i.e. their state != 'R'), if their personal resistance (a float) is less than the viral strength (also a float), then they should be infected.

tl;dr: Return False if the student is fully resistant: otherwise compare the student and viral strength and return True if the student is 'weaker' than the virus.

7 Implementing Person Class: Working with States

Now we'll leave you on your own! There are two more classes to implement in the person class. First, the `get_state` method. This method returns the state of the student, using the `self` as the only input.

Hint: you can do this with only one line of code!

Secondly, implement the `set_state` method. This method should not return anything, but rather change the student's state.

tl;dr: Finish the rest of the person class by implementing the get_state and set_state methods.

8 Implementing School Class: __init__ method

As with the students, we also have a School class. While we are not using this to create multiple schools at the moment, it's still useful for organizing our code. We have a few parameters for you to set.

- Bind the variable `beta` to `self.beta`
- Bind the variable `num_students` `self.num_students`
- Create a variable called `list_students` for the school, and initialize that as an empty list (recall an empty list can be formed like `list()` or [])

You might wonder about the header of this function, which lists beta=2, as opposed to just the regular variable name you're used to seeing. This is a handy Python trick to say that we don't NEED to specify the variable value when creating an implementation of that class, but we can if we want, simply by adding `beta=our_value` to the arguements.

As for the rest of the function, it is adding the students, either 'S', or 'I' to the `list_students`.

Your task is to take the value of `rand_disease` and use that to add a student with that state to the list of students you created up above.

tl;dr: Write the school init class by binding beta and num_students to the school instance, creating an empty list_students, and populating that list with students with the given rand_disease state.

9 Implementing School Class: infect_round method

This is the 'meat' of the code, which is called every time we start a new round by pressing enter. You have two tasks: first, generate a list of infected students. You have free reign on how you want to do this and please as a TA if you're stuck! The list should consist of Person objects, and should be called infected_students. (Note: if you choose a different variable name, that's completely okay, just change the name in the for-loop below.)

The for loop of interactions is already written for you. As you can see, for each student in infected students, we have them 'contact' a number of students. That number is defined by the parameter beta, which we round, because it doesn't make sense to interact with a partial student. Self-check question: what type of variable is `infected` in this code?

Your second task is to increment the rounds. For each INFECTED student in the school, set the `num_rounds` for that student to be 1 greater than the original value (hint: there's a python shorthand for this!! You can type `student.num_rounds += 1`. Then if the number of rounds for that student is greater than or equal to the student's recovery time, the student should be set to recovered. (Hint: you wrote a method to do this already!) This part of the code allows for spontaneous recovery: we generally don't stay sick until the end of time, and it's important for our model to represent that. Of course, we could complicate our model even further by making the resistance impermanent, but it's always good to start with a simple model before adding complexity.

tl;dr: Finish the infect_round method by generating a list of infecting students, and working with the num_rounds of the students as specified in the code skeleton.

10 Implementing School Class: get_num methods

Now we truly leave you on your own: find the `get_num_infected` and `get_num_immune` methods in the code, follow the specifications in the doc strings, and don't be afraid to ask for help if you get stuck!

After you've done those, fill out the `can_continue` method. If everyone is a zombie (aka infected), return False because you have lost the game. If everyone is a robot (aka resistant), return False because you have won. Otherwise, you can continue, so keep playing!

11 Implementing School Class: Methods to Lower Infection Rates

The last part of this project is to actually implement the methods that will allow you to lower infection rates. We have implemented cure_student, but the other three methods are listed in the doc strings. Ask for help if you're confused about any of the specifications! Each of these methods should take just one or two lines, and look at the cure_students method for help iterating through all students. Good luck!

12 Conclusion

Congradulations! You have now finished the zombie model project! Feel free to spend the rest of this time playing around with you simulation. You can add more methods (ask a TA for help with adding the graphics functionality), or simply try to 'win' the game with a new lowest number of rounds. Enjoy simulating a zombie outbreak!