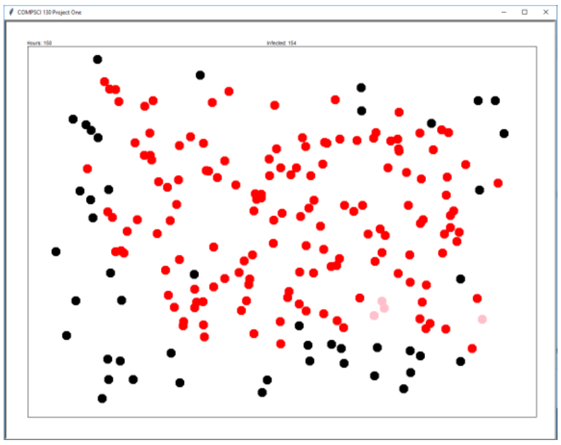
Project One: A simple simulation

Due: 28 April 23:59
Submit: A single text file containing your Python code via Canvas.
Worth: 10% of the final grade (10 marks)

In this project, you will implement a simple simulation demonstrating the spread of viruses within a population. It will require multiple classes.



The simulation consists of several classes. Some are provided for you, and some you will have to write yourself.

- GraphicalWorld is a class that is used to handle the user interface, including the creation of the windows, registering the key bindings and animation bindings, and creating an object to represent the simulated world.
- AnimationFramework is a class used to handle the animation of the world by repeatedly calling a function to simulate the passing of an hour in the simulated world.

The main components of the assignment are the World class, which handles the data and logic required for the simulation, and the Person class which handles the data and logic for an individual "dot" representing a person in this simulation.

Part A: Create a world with a single person that moves around

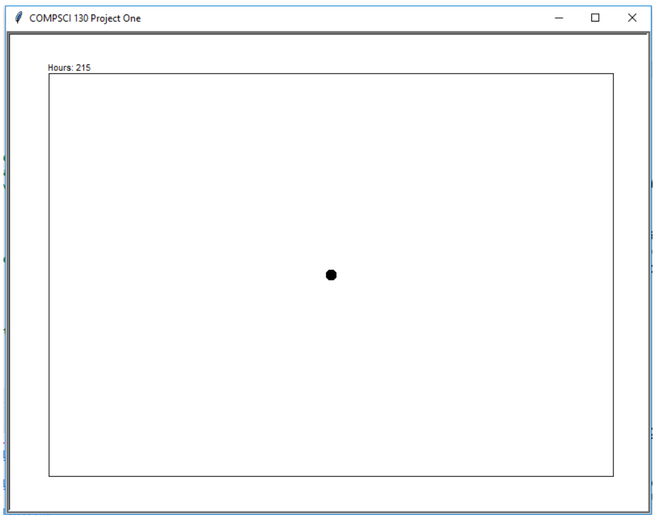
This part is worth 5 marks.

You are provided with method stubs for the Person and World classes.

1. Implement the **draw** method of the **World** class. At this stage, just draw the frame with size equal to width and height of the World. Note that the turtle coordinate system has the origin in the centre of the window.
2. Modify the **draw** method of the **World** class to write the number of hours at the top left of the frame. Update the hours in the **simulate** method of the

World class. At this point you should be able to start and stop the simulation using the space bar, and reset the hours when you press the z key.
3. Modify the **add_person** method of the **World** class so it will create a single Person and add it to the list of people. Call that method from the constructor, so the initial World has one person.
4. Modify the **draw** method of the **World** class so it will also call the draw method of any people that are in the list of people.
5. Modify the **draw** method of the **Person** class to go to the location of the person and draw a dot (turtle.dot) with a diameter of radius * 2.

At this stage, you should have a single Person being drawn on the screen.



6. Modify the **simulate** method of the **World** class so that it updates all the people in the world by calling the **update** method of each person.
7. The **update** method of the Person class moves each person, then checks to see if the person has reached their destination, and if they have then a new (random) destination is chosen. This requires several methods to be modified.

- **_get_random_location** is used to pick a random position within the bounds of the world. The location may not be closer than 1 radius to the boundary or the person would extend over the edge of the frame.
- **self.destination** is used to hold a randomly chosen location as the point that the person will move towards. This should initially be set to a random location whenever the person is created.
- each time the **move** method is called, the person will move half the radius directly towards the destination.
- **reached_destination** checks to see if the location is within one radius of the destination.

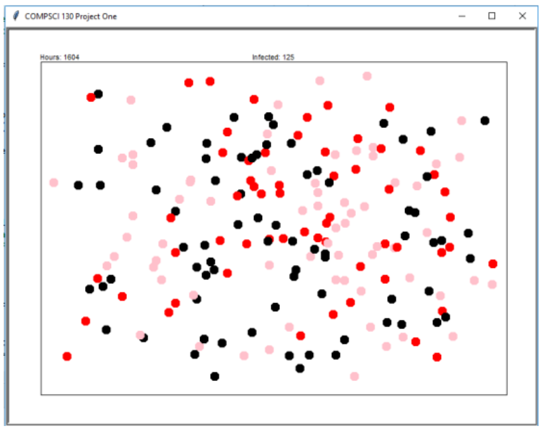
At this stage, the person should be bouncing around the screen very rapidly.
The first stage is complete.

Part B: Add lots of people, and some sick people

This part is worth 3 marks. Less guided detail is provided. You will frequently have to make changes to both the World and the Person class to introduce the functionality required.

8. Modify the **World** class so it adds N people to the list of people. All N people will need to be moved and drawn.
9. Add the ability to infect a person when the 'x' key is pressed by creating a new **Virus** object and passing to a random person from the people in the world. The person should be drawn with the colour of the Virus.
10. Implement the **count_infected** method to count the number of people who are infected with a Virus, and make sure that the number infected is displayed above the frame.
11. Add the ability to cure all the people in the world by pressing the 'c' key.
12. Implement the ability for people to recover from the virus after the duration is reached. When infected with a virus, a person gets sick for the duration of the virus. After each hour (each update), the illness progresses (implement and call the **progress_illness** method), the sickness duration is reduced by one until eventually the person is no longer ill and is cured.

At this stage you should be able to see many different people in the world, some coloured differently after being infected with a virus. If you wish, you could have different kinds of viruses with different colours.



Part C: Add the ability for people to transmit viruses

This part is worth 2 marks. Less detail is provided.

For this part, you will need to detect if one person has touched another (i.e., if a collision has occurred between the shapes). The simple version is functional, but inefficient and will cause the simulation to run slowly. You can get full marks for implementing the simple version.

Start by implementing the **collides** method of the **Person** class. Then implement the **collision_list** method of the **Person** class, which is passed a list of people and will determine which members of the list the person collides with. Finally, implement **update_infections_slow** in the **World** class which will check if an infected person has collided with any other person, and if so, will infect the other person. Make sure the simulate method calls the update infections method.

A Piazza discussion will be available for questions. An FAQ will be used to record the important points.

Marking information

The grading will be based the functional correctness (i.e., the code should perform the task as required), and also the style of the code (i.e., classes and methods should include well written docstrings, variable names are well chosen, the algorithms and code used to implement the functionality is easy to understand and well structured).

	Correctness	Style	Total
Part A	2	3	5
Part B	1	2	3
Part C	1	1	2
Total	4	6	10

Note: This assignment has elements that are deliberately vague and unspecific so that you can make some decisions and decide how to implement the functionality required.

Part D (optional, no marks): Make the collisions efficient

You may have noticed that the collision detection is slow because each person needs to check the distance to each other person (i.e. an O(n2) algorithm). When there are 500 people, the algorithm becomes too slow (try it!).

A more efficient approach uses what is known as a spatial hash table (dictionary) to perform collision detection.

Essentially, the idea is that we could divide the world up into a grid of cells (e.g., 20 x 20 grid). If we know that a person is located in a given grid cell, we only need to check the other people in the same cell for collisions, since other people will be too far away.

It isn't quite that easy, since a person that is right on the border of a cell may overlap into the next grid and collide with a person in the adjacent cell – but we can work around that.

Create a new class called EfficientCollision, which will be used to store a dictionary of locations, each of which has an (x, y) grid cell location as a key, and a list of People as the corresponding value.

- Define a method called hash, which takes a location (i.e. the location of a Person), and returns an (x, y) grid cell location
  (int(x / cell_size), int(y / cell_size))

- Define a method called add which will take a Person and store the Person in the cell corresponding to the location, but also add the Person to any adjacent cell locations that it overlaps, just in case it might collide with a Person in that cell.

- When all of the people are added to the dictionary, you can check for a collision by asking for all the people in the corresponding cells, and simply checking this much smaller list (which might be O(k^2), but the number of people k in the cells will be so small that it will be reasonably efficient – e.g. if there are 500 people and a grid of 20 x 20 cells, then most cells will have only 1 or 2 people.

Note that you have to clear the dictionary and add every person to the dictionary every time they move location, so every single update. It is *still* much faster to do this, since it is only an O(n) algorithm to create the dictionary.