6/1/2020

Coronavirus simulation

DSA assignment semester 1 2020

Documentation

Stephen den Boer

Curtin University

Table of Contents

1. Overview

The simulation uses a graph data type to represent a network of people. There are different types of connections managed by the connections class. Each connection has a correlating intimacy (probably spelt intamacy in my code) which indicates how often the people see each other and how likely they are to spread the virus to each other. There is different restrictions that can be put in place, which will decrease the intimacy of these relationships and so slow the spread of the virus. These restrictions are managed by the Government class. Each individual person also has a chance of imposing restrictions on themselves. These restrictions are managed by the Prevention class.

Each person is managed by a person class which keeps track of things like their name, age, health status, infection status, preventative measures taken, rebelliousness and dates infected. The name is used to identify the person and should be unique. The age and health status are used to determine a person's vulnerability to the virus while the infection status keeps track of their state of infection. There are 5 different infection stages, susceptible uninfected people, hidden carriers of the disease, infected people, recovered people and dead people. Its worth noting that recovered people can be re-infected albeit with a much lower chance. The rebelliousness of a person determines how likely they are to rebel against government restrictions and how likely they are to impose restrictions on themselves.

Each person object is added as a vertex to a graph object. Connections between people are then added as edges in order to simulate a network of people. This graph can be created by the functions in the CreateNetwork file. The CreateGraph file uses these functions and functions in the DSAGraph class to create and save a graph object to file. It takes command line arguments making it useful for running with a parameter sweep. To utilise a network, the Healthsim file must be run. This file can be run in either interactive or simulation mode, running it without command line parameters will give you more information on how to use it. The interactive menu is managed by the interactive file which allows a detailed view and interaction with the network, The simulation Is managed by the sim file which runs until everyone has been infected, there is no infectious people left or a full year has passed (26 fortnightly timesteps). Statistics for the overall population are managed by the Population class.

2. Traceability matrix

Note: Live demonstration means the requirement can be tested via the interactive menu.

Table 1. Traceability matrix

		Requirements	**Design/Code**	**Test**
1	**Driver/Menu & Modes**	1.1. System displays usage if called without arguments.	HealthSim.main()	UI Test: Live demonstration
		1.2. System displays interactive menu with "-i" argument.	Interactive.Menu()	UI Test: Live demonstration
		1.3. In interactive mode, user enters command and system responds.	Interactive.Menu()	UI Test: Live demonstration
		1.4. System enters simulation mode with "-s" argument.	Interactive.Menu()	UI Test: Live demonstration
		1.5. System returns appropriate error messages from wrong user input.	Interactive.Menu()	UI Test: Live demonstration

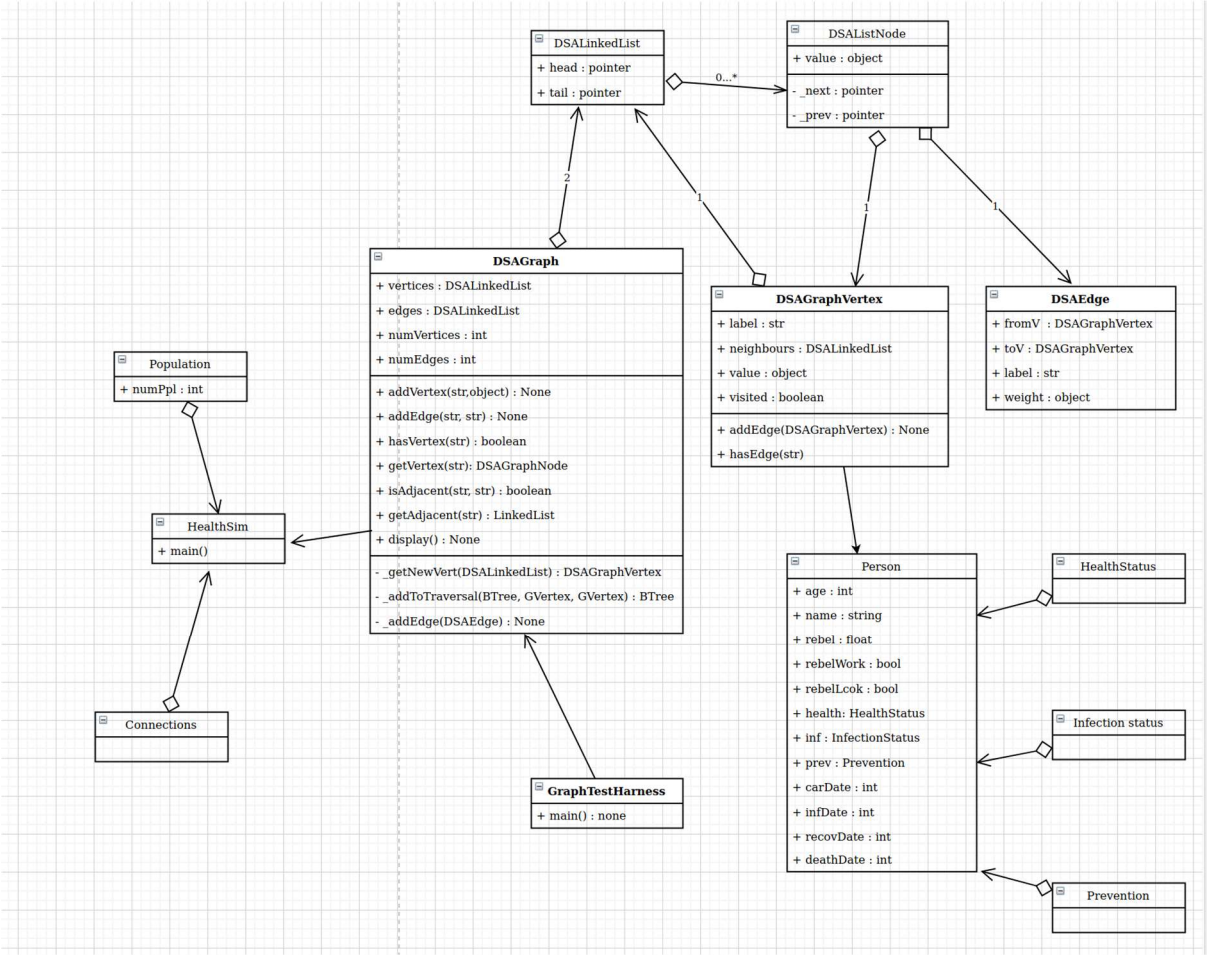
2	**FileIO**	2.1. To read file, user is prompted for filename.	Interactive.Menu()	UT Test: Live demonstration
		2.2. After file read, correct graph produced. (file must first be saved using the program before being loaded)	DSAGraph.loadGraph()	Live demonstration (display graph after loading)
		2.3 To save file the user is asked for filename	Interactive.Menu()	UT Test: Live demonstration
		2.4. Network is saved to file, and when loaded in, produces the correct graph	DSAGraph.saveGraph()	Live demonstration (load saved graph and display graph)
3	**Create a network**	3.1. The user is prompted for the number of people, and the initial percentage infected	Interactive.Menu()	UT Test: Live demonstration
		3.2. A random network is created with the correct number of people and percentage infected. (The percentage may be slightly off since only a whole number of people can be infected	CreateNetwork.init()	Live demonstration (view overall statistics)
4	**Set Probabilities**	4.1. The user is prompted to enter values within range	Interactive.Menu()	UI Test: Live demonstration
		4.2. The values are correctly updated	Interactive.Menu()	No test method currently exists, but by changing to dramatically different values, the differing results from running a timestep should be apparent
5	**Node operations**	5.1. The user is prompted whether they would like to find, insert or delete a person	Interactive.Menu()	UI Test: Live demonstration
		5.2. The user is prompted for the necessary information about the person	Interactive.Menu()	UI Test: Live demonstration
		5.3. The operation is executed and the changes are applied to the graph (if applicable)	DSAGraph.getVertex(), .addVertex, .removeVertex()	Live demonstration (display graph after operation)
6	**Edge operations**	6.1. The user is asked whether they would like to add or remove a connection.	Interactive.Menu()	UI Test: Live demonstraion
		6.2. The user is prompted to enter the connection type and the names of the people connected	Interactive.Menu()	UT Test: Live demonstration
		6.3. The relavent operation is carried out and the changes are reflected in the network.	DSAGraph.addEdge(), .removeEdge()	Live demonstration (display graph after operation)
7	**New infection**	7.1 Upon selecting the new infection option, a new infection is created in the network.	Interactive.Menu()	Live demonstration (display graph, or view statistics after operation)

8	**Display network**	8.1. Upon selecting the display network option the network is displayed	Interactive.Menu()	Live demonstration (view graph)
9	**Display statistics**	9.1. The user is asked whether they would like to view the overall statistics, the list of names in each category, or individual statistics.	Interactive.Menu()	UI Test: Live demonstration
		9.2. The overall statistics are correctly displayed	Population.generateStatistics()	Live demonstration
		9.3. The list of names are displayed in the correct category	Interactive.Menu()	Live demonstration (compare output to network display)
		9.4. the individual statistics are correctly displayed	Person.__str__()	UI Test: Live demonstration
10	**Update (timesteps)**	10.1. The simulation outputs the updated statistics	sim.timestep()	Live demonstration
		10.2. The simulation calculates the chance for each person's infection state to change, and updates it accordingly.	sim.timestep()	Live demonstration (view update statistics and display after timestep)
11	**Extra features**	11.1. A network can be randomly generated with people interconnected in groups who share common connections	CreateNetwork.init()	Live demonstration (test with a small population and observe that groups of people have the same number of connections for a given category)
		11.2. display prints in colour and shows the total number of connections in each category.	DSAGraph.display()	Live demonstration (display graph)
		11.3. Government restrictions are applied automatically depending on the governments level of precaution and the percentage of live people infected	Government.updateMeasures()	Live demonstration (run in simulation mode, or select the timestep option from interactive mode, government measures applied should be displayed on screen or in the text file produced)
		11.4. There is a chance for individuals to rebel against government restrictions, there is also a chance for them to apply preventative measures on their own.	sim.updateIndiv()	Live demonstration (run timesteps and check individual statistics to see preventative measures they employ and if they differ from the government measures)
		11.5. Individuals health and and age are taken into account in determining the chance of a person getting the virus.	Person.getRiskFactor()	No test method currently exists but it is used when running a timestep.
		11.6. There is an additional infection state, where a person can be a hidden carrier of the virus.	infectionStatus.py	Live demonstration (setting a relatively high chance for car_rate and running a

		The carriers have a chance to become a known infection, which is affected by the car_inf_rate. This effectively allows the rate of virus detection to be determined.		timestep, should show carriers of the virus in the network. This can be verified by the display and the statistics)

Source: This table had been adapted from the original template from Valerie Maxville (2019)

3. UML Class Diagram



4. Class Description

4.0. Overview

Object orientation was used substantially throughout the program; however no inheritance was implemented. Instead, the approach taken was to treat objects as a way to contain and manage information. As a result, aggregation was used to link objects together.

4.1. DSAGraph

The DSAGraph class holds all the information pertaining to the graph data structure, it features a list of vertices and a list of edges and keeps track of how many items are in each of those lists. It contains all the functions required to find, add, remove and update both vertices and edges as

well as the necessary functions to display, save and load the graph. This class was created to represent a network of people is a useful way to keep track of people and their connections

4.1.1. DSAGraphVertex

This class represents a node or vertex in the graph. It contains information about the node such as a list of connected nodes as well as a list of connections to other nodes. It also stores a label used to reference and find the vertex as well as a value that can be any object, in this simulation it is always a person object. This class is useful since it keeps the information about each vertex separate from the graph class itself.

4.1.2. SimEdge

This class represents a edge in the graph. It has been specifically modified for this simulation to represent a connection between people. It stores the names of the people the connection is between as well as a label to reference and find the edge. It also stores the type and intimacy of a connection. I decided to create this class since it was necessary to store more than the weight of a connection, as the original graphEdge class did. Alternatively a seperate object could have been aggregated to the edge to represent the type and intimacy of the connection, but since its only 2 pieces of information, it was easier to store it directly in the edge class. It was also useful to have the connection type for creating the edge labels so that different types of connections are distinguishable.

4.2. DSALinkedList

The linked list class holds the implementation of a linked list datatype. It is implemented as a doubly ended, doubly linked list. This means that each item stored in the list, references both the item before and the item after it and that a reference to the beginning and end of the list is stored. This makes the list faster for removing and inserting items at the end of the list since it doesn't have to go through the whole list to find the end. This implementation also contains an insertAny, removeAny and peekAny, which grant access to any index in the list. This is particularly useful when removing a random item from the list. This class is used to store things where we don't know the number of items that need to be stored, such as the number of vertices and connections in a graph.

4.2.1. DSAListNode

This class represents a node in the linked list. It contains the references to the previous and next nodes as well as the value stored in the node, which can be any object. This is helpful since it means the linked list doesn't have to store all the values and references between items.

4.3. Connections

This class represents the connections between people. It uses an array to store the intimacy of a connection, so that connection of type 0, has intimacy stored at index 0 in the array. It also stores an array of the default values for each connection, so that after overwriting values, they can be reset to their original value. It also provides functions to obtain the type of connection and the colour it should be displayed in. This class is decidedly useful, since it allows us to change the intimacy of a connection to reflect restrictions put in place, and to restore the original intimacy once the restriction has been lifted.

4.4. Government

The government class keeps track of which restrictive measures are in place by the government it does this by storing an array with nine elements, one for each measure. At each index in the array, there will be a value of either true or false to indicate whether a method is implemented or not. The methods are arranged in order of severity, so that measure 0 is the least severe and measure 8 is the most severe. This means that measures can be updated so that all the measures below a measure are active, while all the measures above the measure are inactive. This allows measures to be implemented based on the governments level of precaution and the percentage of live people infected by the virus. This class is particularly useful since it means only one command line parameter needs to be passed in order for multiple preventative measures to be implemented.

4.5. HealthStatus

This class represents the health status of a person. It keeps track of the health status with a linked list. This allows multiple health conditions to be prevalent in a person, such as high blood pressure and a lung condition. This functionality of multiple conditions is not currently exploited in the current simulation and the class could be simplified by instead keeping track of the status with a single variable. However, the class provides the opportunity for the simulation to be expanded to include people with multiple health conditions. Since there is five possible health states, it is useful to have this information stored separately from the person class

4.6. InfectionStatus

This class tracks a person's infection state. The different states are, susceptible, carrier, infected, recovered and dead. The carrier state refers to people who are infected but, their infection is hidden in that no one is aware they are infected. When updating a person's status, a susceptible person has a chance of being infected by connections to either a carrier or an infected person, and in turn has a chance to become either a carrier or infected. A carrier has a chance to become a known infection, or to recover. A carrier can't die from the virus since that requires the virus becoming more severe in which case the infection is known. A infected person has a chance both to die and to recover, however the calculation to die occurs first, meaning, for equal death and recovery rates, more infected people will die than recover, this is partially countered by the probability for carriers to recover. Recovered people can still become re-infected, however they only have 10% of the chance of being infected that susceptible people do. Dead people can no longer infect the living, and so when a person dies, their connections are removed from the network.

4.7. Person

The person class stores all the information about a person in the network. It contains their name, age, level of rebelliousness, health status, infection status, preventative measures, and date of infection/recover/death. This class was made in order to keep track of personal information in a contained way so that it could be stored as a vertex in the graph. It may have been useful to use inheritance to represent different types of people. This approach would have made it easier for different classes in society to be represented. Aggregation was chosen here as the preferred approach since it allowed more variability for each person and allowed each person to be more individual and unique.

4.8. Population

This class keeps track of the overall population. It keeps track od the number of people in each infection category in order to generate the percentage of people in each category. The class made it easy to keep track of the population statistics. Instead of passing multiple variables to keep track of the number of people in each category, one object can be passed. It also meant that when updating the graph the numbers in the population also needed to be updated, but this is quicker than having to search through the whole graph every time we needed the overall statistics.

4.9. Prevention

This class keeps track of the individual methods put in place to resist the virus. It is similar to the government class in that it keeps track of the which methods are in place with an array of Booleans. It also uses similar logic to update which measures are being implemented. Since these classes are quite similar, it could have been useful to inherit from a common base class. This didn't make much sense however since they both represent different things, the government class also has to implement a way to apply and remove the measures as they are updated. It may have been useful to implement the Boolean array as a datatype but since it was so simple it was decidedly easier to implement it directly in the class.

5. Justificiation

In the simulation the only datatypes used other than the graph, are linked lists and arrays. This is because they are easier and faster ways to store information. Since I was already using the graph datatype, I also didn't need additional complex datatypes such as trees. I also refrained from using the Linked list as a stack or a queue. This is because the base linked list allows greater functionality without restricting how I may want to use the list. This is particularly true given that the linked list implements functions to access any element in the list.

I decided to implement connections as groups of people when generating a network. This better represents real life, since people are often inter-connected as a group of friends, workmates or peers. I also decided to implement government restrictions based on the infection rate. This is more representative of reality, where governments apply and remove restrictions in response to the spread of the virus. However, this implementation doesn't model the social pressure that builds after being under restrictions for a period of time. It also doesn't model any political pressure to apply restrictions as a result of other nations. It was also decided to implement individual measures as this is more representative of how individuals also have a certain level of adherence to government restrictions and may be more or less cautious than the government.

The vertices and edges in the graph are inserted in alphabetical order. This can make searching for items in the graph quicker since a binary search can be implemented. This capability was formerly made use fo by the depth-first and breadth-first traversals of the graph. However, since the simulation doesn't use these methods, this functionality is unused. It does provide opportunity for future searching implementation to be added.

When creating a network, people are added in 'families' of people living together, with all these people being interconnected. This represents real life where people living in the same house typically have a high level of interaction between them. It is worth noting that there is possibility for

a person to have a ‘family’ of 1 person and so be living by themselves. Connections between people where added in a similar, way where a group of random people were connected. This isn’t necessarily the most realistic approach since it may result in instances such that people living in the same house have different neighbours. Enforcing this more realistic approach was considered more expensive in terms of the time it would take to perform the additional checks.

An array was used to keep track of the people in each group. This is since an array is faster then the other datatypes when the index and number of elements is known. It is also more memory efficient than other datatypes where the number of people isn’t known. When adding the people to the group a linked list was used to store a copy of the list of vertices in the graph. This was due to the fact that people were being removed from a random index in the list, and for an array, it was required to check that person hadn’t already been added, whereas with a linked list, the added person could simply be removed. This also turned out to be slightly faster when testing large groups of people.

Linked lists where also used by the graph to hold the vertices and edges in the graph. This was particularly useful since the number of vertices and edges wasn’t known beforehand. The same holds true for the graph vertex class, where linked lists where used to store the list of connections and neighbours.

The government, health status and connections class all implemented arrays as their main datatype. This is because an array allows them to use an index and a corresponding value. It also has the advantage of being faster than other datatypes such as linked lists and hash tables. The linked list wasn’t used, since it would need to store multiple values, so an extra object would need to be created. It could of course be inserted in order but accessing the middle of a list isn’t efficient. An array also uses less memory than a hash table or a linked list. A linked list needs to store extra information about which items are next to it as well as keeping track of where the list starts and ends. A hash table also stores an array bigger than the number of elements being stored. It would however allow the use as strings for keys, allowing us to use the actual type of connection as a parameter, this however makes it harder to loop over all the elements in the table.

The original display was an adjacency list. This quickly became a problem as more people were added to the network, it just wasn’t going to look good seeing the names of all the people a person was connected to. In order to combat this, I calculated the total number of connections in each category, as well as the total connections, and displayed that information instead. This has the benefit of a much cleaner and condensed display, and arguably produces more useful information. However, it does make it hard however to deduce exactly who is connected with who.