Spreading News

Random walk simulations are among the most popular simulations used to model various phenomena. One of these phenomena is the simulation of the spread of news in a population of people, which can be used by journalists to optimize the content of articles, or to combat misinformation and alarmist messages. For the third assignment, you will use a random walk approach to explore the dynamics of news dissemination. In doing so, we will of course introduce simplifications to make the task more manageable.

Our simulation will work with the following computational models (each represented by its own class):
- news - an example of an article that is forwarded by people within the population; the news will have a content category and will be described by the excitement rate (how much it interests the reader), the time it remains current, and the time of creation.
- person - a member of the population who has interests, reads and forwards messages to their friends; in addition to interests, each person will have an excitement threshold, i.e. how exciting the news piece must be to be read and forwarded.
- population - the group of people in which the piece of news is spread.

Using our simulations, we will seek to answer the questions:
1. How do different news pieces spread in a given population?
2. How many people, on average, receive a news piece with some degree of excitement?
3. What level of excitement does it take for a news piece to reach a certain proportion of all people interested in the topic of the news piece within a population?
4. What level of excitement does a news piece need to reach some proportion of all people interested in the topic of the news piece within a population where everyone is interested in the topic?

The simulation will work with discrete time, so it will consist of several rounds, where in one round the people who received the news can read it and send it on, but the recipients will only read the news in the next round.

Note: Individual classes and simulations build on previous concepts, so we recommend following the order of tasks described in this document when developing your solution. You need to implement the minimum methods described in this guide, but if you want to split some functionality into different methods, or if you need auxiliary methods, you can define them, but they will not be evaluated.

Task 1 – `News` class

The `News` class defines the representation of news in our simulation. Each news item is described by the following values:

- `category` - the category of the news content; in the `news.py` file you will also find the constant `CATEGORIES`, which defines the list of allowed categories in the solution
- `excitement_rate` - the surprise rate; a decimal number between 0 and 1 that expresses how much the news piece is able to surprise the reader: the higher this value is, the more likely it is that a person will read it and then forward it to their friends
- `validity_length` - expresses the time in number of rounds that the news piece is relevant and up-to-date; after this time, people will not be interested in the news any more and therefore will not read it (regardless of the level of excitement)
- `created` - when the news piece was created. This is an integer value that expresses in which round of the simulation the news was created.

The class already contains a ready-made constructor, which first calls a method to check if the parameter values are correct, and then sets the member variables of the class. You need to implement the following methods in the class:

`check_data(category, excitement_rate, validity_length, created)` – 0.5p

This method is used to check the validity of parameter values when creating a new object. In this method, you need to perform various checks and generate various exceptions if a prerequisite is not met. Parameters must have the following values:

- `category` - a string from the `CATEGORIES` list - you don't need to deal with the type of the value, just check if it is one of the allowed values;
- `excitement_rate` - decimal number (`float`) from the interval <0, 1> - if the type is not correct, a `TypeError` is generated, if the value does not match, a `ValueError` is generated;
- `validity_length` - integer from the interval <1, 10> - if the type is not correct, a `TypeError` is generated, if the value does not match, a `ValueError` is generated;
- `created` - integer not less than 1 - if the type is not correct, a `TypeError` is generated, if the value does not match, a `ValueError` is generated.

`get_excitement(time_step)` – 0.5p

The method calculates and returns the current excitement rate of the news at a given point in time, which it receives as the `time_step` parameter. The rate is calculated based on the elapsed time (number of rounds elapsed) at the current instant since the time the message was created. The following rules apply:

- if more time has elapsed than the `validity_length` of the message, the method returns 0
- Otherwise, the current surprise rate is calculated following the formula $ER^{\Delta t}$, where ER is the initial surprise rate of the news (`excitement_rate`), and Δt is the elapsed time since the news was created, expressed as the number of rounds since the news was created. The formula expresses the decrease in recency and hence in the surprise rate of a news piece over time.

Task 2 – `Person` class

The `Person` class represents one person from our group of potential readers. Each person is described by the following characteristics:

- `threshold` - a threshold that expresses how hard it is to get a person's attention - a news piece must have an actual excitement rate higher than this value for a person to read it
- `interested_in` - a list of categories in which the person is interested - only news pieces that fit into one of these categories will be read, other news will never be read, regardless of their excitement level
- `friends_list` - a list of friends of the person - when initialized, this will be an empty list, to which we will later add people (objects of type `Person`) to whom the person can forward the news
- `has_read` - a list of news pieces that the person has already read - when initialized, this will be an empty list to which we will later add news pieces (objects of type `News`) that the person will read; we need the list to prevent the person from reading (and forwarding) the same news piece multiple times
- `patience` - the number of news pieces a person is willing to read in one round of simulation.

The class contains a ready-made constructor, but you need to add the remaining methods:

`is_interested_in(category)` – 0.1p

Returns information about whether the person is interested in the category the method receives as the `category` parameter. The return value of the method is therefore `True` or `False`.

`has_read_news(news)` – 0.1p

Returns information about whether the person has read the news piece they receive as parameter `news` (object of type `News`). The return value of the method is `True` or `False`.

`make_friends(population, n)` – 0.3p

The method populates the list of friends of the given person, receiving two parameters:

- `population` - a list of people from which it can select friends
- `n` - the number of the person's friends.

After the method is executed, the person's friends list (`friends_list`) will be populated with *n* objects of type `Person` from the population list. The selection of friends is completely random, but the friends must be unique values (each object will be in the friends list at most once). Of course, a person cannot be a friend of themselves.

`process_news(news, time_step)` – 1p

The method is a simulation of what a person will do if they receive a news piece. The method always returns a list of people to whom the person forwards the news. The method has two parameters:

- `news` - an object of type `News` - the news piece the person received
- `time_step` - `integer` - the point in time at which the message was received.

After receiving the news piece, the person behaves as follows:

- if a person has read at least as many news pieces as his/her patience in a given simulation round allows, he/she will not read the next news piece and will not forward it to anyone

- if he/she has already read the news piece before, he/she will not read it again and therefore will not forward it to anyone
- if the news piece has a category that does not interest the person, he will not read and forward it to anyone
- if the news piece does not interest him or her with its current level of excitement, he or she does not read it and does not forward it to anyone
- Otherwise, they will read the news piece and forward it to all their friends who are interested in the topic of the news piece.

Note: If the person reads the message, be sure to update the corresponding member variable of the `Person` object.

Task 3 – Classes `Population` and `HomogeneousPopulation`

The `Population` class represents the group of people who read and send news to each other. The class constructor is already finished, the following member variables are set in it:

- `people` - a list of people in the population - after initialization an empty list, populated in the `generate_population` method
- `active_news` - list of news pieces that are spread in the population - empty after initialization, the content is updated in the methods `introduce_news` and `update_news`
- `patience_limit` - a pair of integer values that express the lower and upper limits of the readers' patience. The real value for each member of the population is generated from this interval (including the limit values).

In addition, the constructor includes a call to the `generate_population` method, which you must implement along with the other methods:

`generate_population(n, friends_count, patience_limit)` – 1p

The method will generate a list of people in the population and create friendships between them. The method takes two parameters, and has no return value - it directly updates the list of people in the population. The parameters of the method have the following meaning:

- `n` - `integer` - the number of people in the population - the number of objects you need to create and add to the list of people
- `friends_count` - `integer` - each member of the population will have this many friends after the population is generated
- `patience_limit` - a pair of integer values that express the lower and upper limits of the readers' patience.

Generate people with a random threshold value from the interval <0, 1) and four random regions of interest (from the list of allowed categories). Generate patience randomly from the `patience_limit` interval (including limit values).

`introduce_news(news)` – 0.5p

The method represents the point at which a news piece begins to spread in the population. The `news` parameter is an object of type `News`, so it is the news that we want to circulate in the population. The method returns a list of the first five people to receive the message, which can only be people who will be interested in the topic.

Note: for proper functionality, don't forget to update the `active_news` list.

`update_news(time_step)` – 0.5p

This method updates the list of active news and is necessary in order not to try to send news pieces that are no longer up-to-date (they have expired) unnecessarily. The method has one parameter - `time_step` (`integer`) - which represents the point in time or round in which we want to update the list of active news. The method has no return value, it just deletes from the list of active news those pieces that no longer have a chance to be read by the population members (their current excitement rate is zero).

`count_readers(news)` – 0.25p
The method calculates and returns the number of people in the population who have read the `news` message (object of type `News`, method parameter). In the method, treat also the case if `news` is not an active news piece, but could have been active in the past. The return type of the method is an integer.

`get_number_of_interested(category)` – 0.25p
The method returns the number of people who are interested in the news category given as a parameter of the `category` method. The return type of the method is an integer.

The `population.py` file also contains the definition of the `HomogeneousPopulation` class, which is a closer specification (and subclass) of the `Population` class, and represents a population in which each member is interested in a certain category of messages. The interface and functionality of the class are unchanged, but the class constructor is extended with an additional parameter `category`, which represents the category that everyone is interested in. In the constructor, this value is then stored in the member variable `self.category`.

In the class, you need to redefine the **`generate_population`** method (grading **0.5p**), and you can build on the implementation in the `Population` class, but make sure that each member of the population has a common category of interest for that population in the list of topics of interest to it.

Task 4 – simulations

The function definitions implementing the various simulations can be found in the `simulation.py` file. Your task is to implement these functions.

`simulate_spread(all_news, population)` – 2p

This function is used to examine the dynamics of news dissemination (in the `all_news` list) in a population (`population`). The parameters of the function have the following form and meaning:

- `all_news` - list of `News` instances that will be propagated in the population
- `population` - an object of type `Population` (or `HomogeneousPopulation`), i.e. a group of people.

The function tries to answer the question how many people will read the news from the list of news in the population after each step (`time_step`) of the simulation. In doing so, the simulation will have the following structure:

1. at the beginning of the simulation, add messages to the population and get the list of people who will read the news piece first
2. simulate one step of the simulation repeatedly, with the following operations to be performed in one step:
    - record the number of people who read the news for each news piece from the `all_news` list
    - simulate the moment when the people who received the news piece read it and get the list of people to whom the news piece is forwarded
    - remember the list of people to whom each news piece is forwarded in the next round of the simulation
    - update the list of news pieces in the population (delete those that are no longer relevant)
3. the simulation continues as long as a message is active in the population

The return value of the function is the information about the change in the number of people who read each news piece, in the following structure:

- the return value is a `dictionary`
- the keys of the dictionary are objects of type `News` from the list `all_news`
- under the keys are stored lists of integers that represent the number of people who have read the given news piece

You can see an example of the return value below, with the first news piece having an excitement rate of *0.9* and the second news piece having a surprise rate of *0.5* - which is why fewer people read it:

```
{
<news.News object at 0x000002BCFA430A58>: [0, 5, 36, 190, 439,
511, 513, 513, 513, 513, 513, 513],
<news.News object at 0x000002BCFA477898>: [0, 5, 17, 41, 59, 65,
65, 65, 65, 65, 65, 65]
}
```

`average_spread_with_excitement_rate(excitement_rate, pop_size, friends_count , patience_limit, test_count)` – 1p

Another function looks for the average number of people who read a news piece with a given excitement rate during the simulation. The parameters of the function are as follows:

- `excitement_rate` - `float` - the excitement rate of the news piece
- `pop_size` - `int` - number of people in the population
- `friends_count` - `int` - number of friends of each member of the population
- `patience_limit` - `(int, int)` - lower and upper limit of the possible patience value of a population member
- `test_count` - `int` - optional parameter, expresses the number of simulations from which the average number of readers is calculated.

In the function, perform repeated simulations (the number given by the `test_count` parameter), requiring in each iteration:

1. create a news piece with a random category, a specified excitement rate, a recency length of 10, and a creation time of 1
2. create a population of a given size and number of friends per member (each iteration will work with a different population)
3. simulate the propagation of the generated news piece and get the final number of people who have read the news piece.

The function has two return values:

- the first value is a list of the final number of readers for each test iteration (the length of the list will correspond to the parameter `test_count`)
- the average value of the first return value.

`excitement_to_reach_percentage(percentage, pop_size, friends_count , patience_limit)` – 1p

The function seeks to answer the question of at least what level of extiement a news item needs to have to reach a certain proportion of readers interested in its category. The parameters of the function are as follows:

- `percentage` - a `float` between 0 and 1 - the proportion of people who may be interested in the news item and we want to reach them, e.g. if it has a value of 0.5, it means that we want to reach at least half of all people who are interested in the topic
- `pop_size` - `integer` - population size (number of people)
- `friends_count` - `integer` - number of friends of one member of the population
- `patience_limit` - `(int, int)` - lower and upper limit of possible patience value of a population member.

In the function, you need to successively test different excitement rates of news pieces and see if the necessary proportion of readers is reached with a given value. Due to the randomness of the simulations, this value will vary a bit with different executions of the function, but it is enough if you do one simulation with each rate, the deviation will only be minimal.

An experiment with some news excitement rate will have the structure:

1. create a population with the given parameters
2. create a news piece with a random category, the given excitement rate, a recency length of 10 and a creation time of 1
3. get the number of people who will read the news piece by the end of one simulation

4. get the number of people in the population who are interested in the topic of the generated message
5. find out whether the message has been read by the desired subset of people
6. if the proportion has been reached, the function returns the excitement rate; if the desired proportion is not reached with any excitement rate, the function returns `None`.

Note: When trying different excitement rates, increment the value incrementally by *0.01*; we recommend using the numpy representation of `floats` for higher precision and easier handling of the counter.

`excitement_to_reach_percentage_special_interest(percentage, pop_size, friends_count , patience_limit, news_category)` – 0,5p
The last function solves a similar issue as the `excitement_to_reach_percentage` function, but does so in a population of type `HomogeneousPopulation` and for a news with the specified topic (defined in the `news_category` parameter).

Thus, the differences from the previous function are as follows:
- generate a population of type `HomogeneousPopulation`, where each member is interested in the topic defined by the `news_category` parameter
- the generated news must have a category according to the `news_category` parameter.

The structure of the simulation and the return value of the function are unchanged.

The `simulation.py` file also contains a main function that you can use for arbitrary experiments. You can further test your solutions by using the sample tests that are created for each solution project file.