Aaron Fox
CECS 622
Dr. Elmaghraby

Test 1 (Rumors Simulation Project)

Introduction: A simulation program was written to simulate the Rumors Project assigned on Blackboard as Test 1. The program was written in Python using its native libraries with some imports. Following the instructions of the prompt, four main assumptions were written out and followed in addition to the explicit rules laid out in the prompt. The software was then explained at a high level in this report. The project itself involved an initial student spreading a rumor to his paired partner in the first minute of a hypothetical party. The group of even-integer N students at the party then paired off for every minute of the input number of minutes to run the project, with each student that has already heard the rumor having a 50% chance of spreading the rumor to another student. There were several experiments run on the Rumors Project following this situation on different numbers of students ranging from 100, 1,000, and 10,000 at different amounts of time. Observations were then made on the experiments and the questions from the prompt were also answered. The similarities and differences to real-world infectious disease modeling were also evaluated.

Assumptions: For clarity, the initial assumptions from the prompt's story were:

- N students are at a party (N is an even integer $2 \leq N \leq 10{,}000$)
- At some point, all students pair off at random and talk for exactly one minute.
- At the end of the minute, all students again pair off with another person at random.
- One student wants to start a rumor. He spreads the rumor to his conversation partner at noon.
- Every person who has knowledge of the rumor obeys these rules:
    - The likelihood of spreading a rumor to another person is 0.5
    - After a person has heard the rumor 2 times, he/she will assume everyone has heard the rumor and will no longer try to spread it further

The above were all implemented and followed in the simulation program written. Below are the assumptions that were taken in the creation of this simulation:

- Students can pair up with the same student again in different minutes (as often happens at parties in the real world.)
- If a student has shared the rumor once with a student already and meets up with them again, then that student cannot share the rumor again. (It won't count toward the 2 times that the receiving student has heard a rumor.)
- Likewise, students can keep track of who has told them the rumor so they can't "spread" the rumor to someone who has already told them the rumor.
- If both students in a pair have heard the rumor and haven't previously spread the rumor to the other, they each still have the possibility to tell the other student about the rumor (if

the random likelihood is true), and a randomly chosen student in the pair is chosen to first tell the rumor. If the first student's attempt is not successful, then the second student can try to spread the rumor to the first.
- The first student to spread the rumor is considered to have heard the rumor zero times at the party even though they know the rumor. While they technically may have "heard" it from their own brain, this is not counted in their personal count of the number of times that they have heard the rumor.

To elaborate, these assumptions mean that students are able to be matched up randomly each round (which is necessary anyway depending on the number of minutes to run the simulation and the number of students), that students cannot be told a rumor by one student twice nor can they tell a student twice, and that, if both students have heard a rumor and are paired, they may both attempt (in randomly set order) to spread the rumor to the other. But, once one student has spread the rumor to the other, they must follow the aforementioned assumptions above.

Approach (Software Design): The software was written in Python using its native libraries with one import module, `random`. This design took an object-oriented approach for this simulation. The core of the simulation rests in the Student class, which takes in the following parameters:

```
class Student:
    def __init__(self, student_id, times_heard_rumor=0, likelihood_of_spreading=0.5):
```

Each student is initialized with a unique student identification, `student_id`. Each student ID is used to identify each student, and it is used to allow students to keep track of which other student each instantiated student has already spread the rumor to and which student has spread the rumor to them. The other two parameters are always their default values as listed above (with `times_heard_rumor` = 0 and `likelihood_of_spreading` = 0.5) for the purposes of this experiment, but they were included for the purposes of future extensibility upon this experiment to test more variables for possible future research.

Each `Student` object also has a boolean variable called `has_heard_rumor`, which keeps track of who has heard the rumor. This is important because the very first student knows the rumor but has not heard it from anyone at the party yet, so technically their times_heard_rumor variable (at the party) is 0 although they themselves already know the rumor, following the last assumption listed in the "Assumptions" section above. When a student has heard a rumor, starting from the very first statement, `has_heard_rumor` changes from `False` to `True`.

Each student also has a variable called `times_heard_rumor`, which is always initially set to zero during the experiments of this situation. The number of times that the student has heard the rumor. If this value is 0, then the student has not heard the rumor. If this value is 1, then the student has heard the rumor once. If this value is 2, then the student has heard the rumor twice and stops attempting to spread the rumor, following the story prompt. Each student object also

has a `likelihood_of_spreading` variable, which is the chance of each student being able to spread the rumor to another student. For the purposes of this experiment, this is always 0.5.

The last variable that each student has is the list array called `students_spread_rumor_to`. This variable is a list of student IDs that each student has spread the rumor to already. This variable is used to prevent students from spreading the rumor to a student whom they have already told the rumor and, consequently, can also be used to prevent students from spreading the rumor to a student who has already spread the rumor to them.

Each `Student` has several methods as well, including getters, setters, and other necessary checks and methods. The methods are listed below:

`__repr__()`: Used for representing the `Student` object with its Student ID to the console and debugger.

`__str__()`: returns a string output of all the variables contained in the `Student` object.

`set_heard_rumor(boolean)`: Sets the `has_heard_rumor` variable of the `Student` object.

get_has_heard_rumor(): Returns the `has_heard_rumor` variable of the `Student` object.

get_id(): Returns the `student_id` variable of the `Student` object.

`append_students_spread_rumor_to(student_id)`: Appends a student ID to a `Student` object's `students_spread_rumor_to` list array.

`increment_times_heard_rumor()`: Increments the `times_heard_rumor` of the `Student` object and ensures that `set_heard_rumor` is set to true.

get_students_spread_to(): Returns the `students_spread_rumor_to` list array of the `Student` object. This is used to ensure that a student that has heard of a rumor from someone cannot tell them the same rumor again and vice versa.

`can_spread_rumor(student_id_to_tell_rumor_to, other_students_spread_rumor_to)`: This is the most important method of the class, and its logic looks like:

```
def can_spread_rumor(self, student_id_to_tell_rumor_to, other_students_spread_rumor_to):
        if self.has_heard_rumor == True:
            if self.times_heard_rumor < 2:
                if student_id_to_tell_rumor_to not in self.students_spread_rumor_to:
                    if self.get_id() not in other_students_spread_rumor_to:
                                            return True
        return False
```

This method determines if this student is capable of spreading the rumor to another student and makes sure that:

- this student indeed already knows the rumor,
- this student hasn’t heard the rumor more than one time,
- this student hasn't already told the other student the rumor,
- and that the other student likewise hasn't told this student the rumor.

If all these requirements are met in this function, then the student is allowed to spread the rumor to the other student. Note that this does not necessarily mean that this student *must* pass on the rumor to the other student, for there is still a 0.5 chance that the student does not successfully pass on the rumor to the student, per the input `likelihood_of_spreading` variable.

The function `run_rumor_simulation` runs the core of the actual simulation by using the Student class described above. Its header looks like:

```
run_rumor_simulation(likelihood_of_spreading_rumor=0.5, number_of_students=100, minutes_to_run=10)
```

with the input parameters allowing for experimenting with the number of students, *N*, and the minutes to run the simulation for, *t*. The method begins by making sure the `number_of_students` input parameter is an even number with a simple modulus two check. It then creates an array of students based off the Student class described above, like so:

```
students = []
for i in range(number_of_students):
    students.append(Student(student_id=i, times_heard_rumor=0,
                    likelihood_of_spreading=likelihood_of_spreading_rumor))
```

Notice how every student has its own unique `student_id`, starting from 0 to `number_of_students` - 1. A random student is then chosen to initially start the rumor by using Python’s `random.randint` module.

The core of the simulation then begins by using a for loop to iterate over every single minute of the simulation based on the input parameter `minutes_to_run`. At the beginning of every for loop, the array list of students is randomly shuffled. This ensures that every student will have a completely random partner (and possibly the same partner again, as specified in the “Assumptions” section above.)

Another for loop then iterates over every pair of students. In each loop, an initial first case is checked if the minute is 0. In this one-off case, the initial student spreads the rumor to his partner guaranteed. After this one-off case, three possible cases of rumor-spreading are checked during every loop. In the first case, the first student in the pair has heard the rumor and the second student in the pair hasn't. In this case, the first student has a 0.5 chance of spreading the rumor to the second. The second case is just like the first except the second student tries to spread the rumor to the first student of the pair.

The third and final possible case, wherein both students have heard the rumor, is the trickiest case. Here, there is a check to see if both students know the rumor already, they still have a random and equal chance of telling the rumor to the other student in the pair again if they haven't already told the other student the rumor or have been told the rumor by the other student. The one caveat is that if one student successfully tells the other student rumor first, then the student who just got told the rumor can't retell the rumor to the same person, as this would be very redundant during a real-world party. The first student to attempt to tell the rumor to the other in the pair is randomly chosen in the pair here. Note that there is still a 0.5 * 0.5 = 0.25 chance that neither student will succeed in telling the rumor to the other.

The final results of the method are then printed out and the total number of students that have heard the rumor is returned.

To find the average number of students that have heard the rumor for N students over t minutes, a for loop is run over `iterations_to_run` like so:

```
N = 1000
minutes = 10
iterations_to_run = 100
num_students_heard_rumor = []
for i in range(iterations_to_run):

num_students_heard_rumor.append(run_rumor_simulation(likelihood_of_spreading_rum
or=0.5, number_of_students=N, minutes_to_run=minutes))

print("num_students_heard_rumor: " + str(num_students_heard_rumor))
print("avg num students who have heard rumor " +
str(sum(num_students_heard_rumor)/len(num_students_heard_rumor)))
print("Average percentage of students who have heard rumor == " +
      str(sum(num_students_heard_rumor)/len(num_students_heard_rumor) / N))
```

This allows for rapid changing of the number of students and minutes to run to allow for experimenting

Questions Answered:

For N = {100, 1,000, 10,000} run your simulation several times to determine:

1. On average, what % of the attendees will have heard the rumor after 10 minutes?

The `run_rumor_simulation` was run 1000 times and averaged out to remove outlier percentages, and the average results are:

For N = 100: the results are:

After 1000 iterations over 10 minutes for 100 students:
Average number students who have heard rumor 45.639
Average percentage of students who have heard rumor == **0.45639 (45.639%)**

For N = 1,000, the results are:
After 1000 iterations over 10 minutes for 1000 students:
Average number students who have heard rumor 70.837
Average percentage of students who have heard rumor == **0.070837 (7.0837%)**

For N = 10,000, the results are:
After 1000 iterations over 10 minutes for 10000 students:
Average number students who have heard rumor 76.668
Average percentage of students who have heard rumor == **0.0076668 (0.76668%)**

2. On average, what % of the attendees will have heard the rumor after 20 minutes?

For N = 100: the results are:
After 1000 iterations over 20 minutes for 100 students:
Average number students who have heard rumor 93.935
Average percentage of students who have heard rumor == **0.93935 (93.935%)**

For N = 1,000: the results are:
After 1000 iterations over 20 minutes for 1000 students:
Average number students who have heard rumor 781.198
Average percentage of students who have heard rumor == **0.781198 (78.1198%)**

For N = 10,000: the results are:
After 1000 iterations over 20 minutes for 10000 students:
Average number students who have heard rumor 3054.988
Average percentage of students who have heard rumor == **0.30549879 (30.54988%)**

3. On average, what % of the attendees will have heard the rumor after 40 minutes?
For N = 100: the results are:
After 1000 iterations over 40 minutes for 100 students:
Average number students who have heard rumor 99.026
Average percentage of students who have heard rumor == **0.990259 (99.026%)**

For N = 1,000: the results are:
After 1000 iterations over 40 minutes for 1000 students:
Average number students who have heard rumor 984.47
Average percentage of students who have heard rumor == **0.98447 (98.447%)**

For N = 10,000: the results are:
After 1000 iterations over 40 minutes for 10000 students:
Average number students who have heard rumor 9761.044
Average percentage of students who have heard rumor == **0.9761044 (97.61044%)**

4. At what time, t, will 10% of the party have heard the rumor? N = 10,000.
After running a for loop over every minute for 10 iterations and averaging out the result, it can be seen that at time t=16, the average time is 7.994% and at time t=17, the average percentage of students who have heard the rumor is 14.094%. Thus, **sometime during t=16 minutes, 10% of the party has heard the rumor with N=10,000.** The graph below illustrates the average percentage of students having heard the rumor at different times t for N=10,000, and the time of around 16 seconds corresponds to the percentage of about 10 on the Y-axis.

5. At what time, t, will 50% of the party have heard the rumor? N = 10,000.
After running a for loop over every minute for 10 iterations and averaging out the result, it can be seen that at time t=22, the average percentage of students having heard the rumor is 49.498% and at time t=23, the average percentage of students who have heard the rumor is 53.071%. Thus, **sometime during t=22 minutes, 50% of the party has heard the rumor with N=10,000.** The graph below illustrates the average percentage of students having heard the rumor at different times t for N=10,000, and the time of around 16 seconds corresponds to the percentage of about 50 on the Y-axis.

Observations/Discussion: Smaller populations of students appear to spread the disease, as a percentage, much faster than larger populations. This is likely because there are fewer students that cluster in small groups and so the rumor spreads quickly in the smaller population, with N = 100 reaching 45% spread after just 10 minutes, N = 1,000 reaching 7.08% after 10 minutes, and N=10,000 reaching a mere 0.767% after 10 minutes. Larger populations thus appear more resilient as each student in the population is less likely to pair with someone who is infected over time. In fact, I found it very surprising that the smaller populations started to exponentially spread the rumor so much quicker than the larger population of students.

As expected, more students became infected over time as the minutes went on. This occurred, as expected, at an exponential rate, with the ten minutes from t=10 to t=20 leading to a sharp exponential jump for the N= 1,000 population from 7.08% students having heard the rumor to 78.12% students having heard the rumor. This shows the severity of the spread of rumor to be able to spread rapidly once it continues to get worse. This behavior was also expected as students cannot “unhear” rumors that they have already been told, so only an increase in the number of students that have heard the rumor was possible in the first place.

Despite the inherent randomness in the pairing of the students, which student starts the rumor first, and who, when two pairs of students both know the rumor, gets to spread the rumor first, the average of running `run_rumor_simulation` remained steady around the percentages they were given. Although the maximum and minimum percentages would vary often, the overall

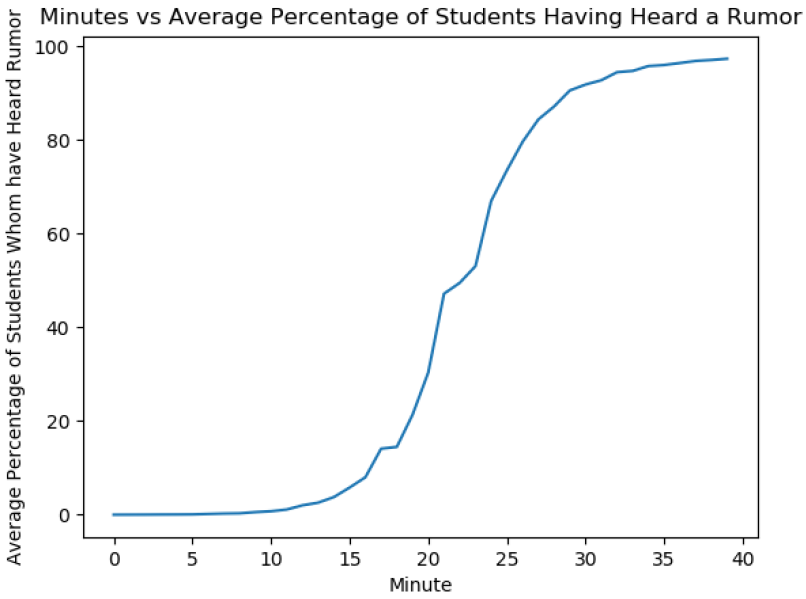
average of the experiments remained within a couple percentage points; the margin of consistency maintained even more consistency as more iterations were run.

It was initially surprising to see the percentage of students for the N = 10,000 rise so rapidly at time t=40 after its percentage initially did not rise very quickly at times t=10 and t=20 (at a mere 0.767% at t=10), but, after realizing that there were (10,000 / 2) * 40 = 200,000 pairings of each student after that 40 minutes, it became clear how easily the rumor could have spread throughout the population when compared to the mere (10,000 / 2) * 10 = 50,000 initial interactions of the students at t=10, given the exponential rate of spread of the rumor.

After finding an array for each minute and the average percentage of students that have heard the rumor at each minute for N = 10,000, the following observation was observed, with each array being equal to [minute, average percentage of students who have heard rumor]:

[[0, 0.01], [1, 0.02], [2, 0.03], [3, 0.049], [4, 0.059000000000000004], [5, 0.079], [6, 0.16699999999999998], [7, 0.255], [8, 0.303], [9, 0.551], [10, 0.75], [11, 1.111], [12, 2.006], [13, 2.5780000000000003], [14, 3.771], [15, 5.806], [16, 7.994], [17, 14.094000000000001], [18, 14.482999999999999], [19, 21.349], [20, 30.377], [21, 47.183], [22, 49.498999999999995], [23, 53.071], [24, 66.99199999999999], [25, 73.53], [26, 79.62199999999999], [27, 84.38799999999999], [28, 87.156], [29, 90.569], [30, 91.838], [31, 92.74700000000001], [32, 94.46700000000001], [33, 94.744], [34, 95.785], [35, 96.004], [36, 96.431], [37, 96.875], [38, 97.075], [39, 97.34799999999998]]

As a graph for N = 10,000 with 10 iterations over each minute averaged, it looks like:



Graph illustrating the average results of the simulation over time for N=10,000

The spreading of the rumor thus appears to follow a general exponential increase in the percentage of the spread of the rumor over time, bottling out near minute 28 when there are fewer and fewer students whom the rumor can be spread to. The curve thus follows a rough S-curve

Similarity and Contrast to Infectious Disease Modeling: With the jarring reality of the now-infamous COVID-19 virus striking the world at the time of writing this, carrying out the experiments of this case study was surprisingly relevant to the situations occurring in infectious disease models all over the world. During the planning of assumptions and following the story guidelines of this simulation, many similarities and differences began to manifest themself. The most obvious conclusion as soon as reading this story prompt is comparing the “rumor” being spread to a disease such as coronavirus and the possible spreading of the disease.

Just as infectious disease models use R_0 values that help determine how likely one person is to infect other people (with COVID-19 having an R_0 value between 2 and 2.5), this case study had a `likelihood_of_spreading` variable that determined how likely it was that one could spread it to other people. The two concepts differ in that R_0 is an estimate of how *many* people someone will infect over time while the `likelihood_of_spreading` variable determined how likely it was for one student to spread the rumor to one specific other student, however; the `likelihood_of_spreading` variable did not determine how many people someone may spread it to, although it could certainly shape the amount that it spread it too (with a high likelihood being equal to a higher R_0, although most likely not linearly.)

The concept of one student hearing the same rumor twice at a party and then not deciding to spread it further is very similar to the quarantine measures that infectious model diseases take into account and that are currently taking place due to the COVID-19 pandemic. Just as people suddenly start quarantining themself once they realize they are “infected” and don’t want to spread it anymore, the students stop trying to spread the rumor when they are aware of its spread already. Just as a student who has only heard the rumor once can still spread the rumor to more people, though, some people are asymptomatic and may not realize they are spreading something that they actually are. These are all concepts that must be taken into mind by infectious disease models.

The purposeful and continued pairing of students over time, even after the rumor has been spread and awareness of the disease could clearly be known, is unrealistic in the real world and infectious disease models would have to account for humans taking measures such as quarantining, finding a cure, or other prevention measures that would slow the increase of cases in a population. Some of the other assumptions, such as a student not being able to spread the rumor to someone who told them the rumor already or to be told the rumor by someone the student already spread the rumor too, are also accounted for in most infectious disease models because many diseases can not spread to a person over and over again.

Infectious disease models also have to account for people being cured and people dying off and not being able to spread the disease any more, which could eventually lead to a decrease in the amount of people who have disease. This is different from this case study, where students cannot unhear a rumor that they have already heard.

This case study was very revealing and striking with its similarities to infectious disease modeling currently taking place all over the world during the COVID-19 pandemic currently taking place in the world at the time of this writing, and it was very impactful and interesting to create my own case study during this time while quarantined at home.

Appendix

The source code is all contained in Fox_Test_1.py:

```
# Aaron Fox
# CECS 622-01
# Dr. Elmaghraby
# Test 1 (Rumors Simulation Project)

# Used for selecting a random starting student to start spreading the rumor
# and for shuffling students to later pair them
import random # Using randint and shuffle

## Story ##
# + N students are at a party (N is an even integer 2 ≤ N ≤ 10,000)
# + At some point, all students pair off at random and talk for exactly one minute.
# + At the end of the minute, all students again pair off with another person at random.
# + One student wants to start a rumor. He spreads the rumor to his conversation partner at noon.
# + Every person who has knowledge of the rumor obeys these rules:
#    1. The likelihood of spreading a rumor to another person is 0.5
#    2. After a person has heard the rumor 2 times, he/she will assume everyone has heard the
#       rumor and will no longer try to spread it further

## Assumptions
# + Students can pair up with the same student again as often happens at parties in the real world
# + If a student has shared the rumor once with a student already and meets up with them again,
#   then that student cannot share the rumor again and it won't count toward the 2 times
#   that the receiving student has heard a rumor.
# + Likewise, students can keep track of who has told them the rumor so they can't "spread" the rumor to someone
#   who has already told them the rumor.
# + If both students in a pair have heard the rumor and haven't previously spread the rumor to the other,
#   they each still have the possibility to tell the other student about the rumor (if the random likelihood is true),
#   and a randomly chosen student in the pair is chosen to first tell the rumor. If the first student's attempt is
#   not successful, then the second student can try to spread the rumor to the first.
# + The first student to spread the rumor is considered to have heard the rumor zero times at the party even though
#   they know the rumor. While they technically may have “heard” it from their own brain, this is not counted in their
#   personal count of the number of times that they have heard the rumor.


# The Student class keeps track of if the student has heard the rumor, the number of times
# that the student has heard the rumor, their unique student ID, the likelihood of each student
# spreading the rumor, and the students whom they have spread the rumor to
```

```
class Student:
    def __init__(self, student_id, times_heard_rumor=0, likelihood_of_spreading=0.5):
        # Unique student id to identify each student and keep track of which student
        # each student has talked to
        self.student_id = student_id

        # Keeps track of who has heard the rumor. This is important because the very
        # first student knows the rumor but has not heard it from anyone at the party yet, so technically
        # their times_heard_rumor is 0
        self.has_heard_rumor = False

        # The number of times that the student has heard the rumor.
        # If this value is 0, then the student has not heard the rumor.
        # If this value is 1, then the student has heard the rumor once.
        # If this value is 2, then the student has heard the rumor twice and stops spreading the rumor
        self.times_heard_rumor = times_heard_rumor

        # The chance of spreading a rumor to another student
        self.likelihood_of_spreading = likelihood_of_spreading

        # Maintain a list of unique student_ids that this student has spread the rumor
        # to to ensure that the student doesn't try to spread the rumor to one
        # person twice and increase another student's times_heard_rumor twice
        self.students_spread_rumor_to = []


    def __repr__(self):
        return "Student ID: " + str(self.student_id)

    def __str__(self):
        return str("Student ID: " + str(self.student_id) + ", Has heard rumor: "\
            + str(self.has_heard_rumor) + ", Number of times heard rumor: " + str(self.times_heard_rumor)\
            + ", Likelihood of spreading rumor: " + str(self.likelihood_of_spreading)\
            + ", Students spread rumor to: " + str(self.students_spread_rumor_to))

    def set_heard_rumor(self, has_heard_rumor):
        self.has_heard_rumor = has_heard_rumor

    def get_has_heard_rumor(self):
        return self.has_heard_rumor

    def get_id(self):
        return self.student_id

    def append_students_spread_rumor_to(self, other_student_id):
        self.students_spread_rumor_to.append(other_student_id)

    def increment_times_heard_rumor(self):
        # Ensure set heard rumor is true for the student
        self.set_heard_rumor(True)
        self.times_heard_rumor = self.times_heard_rumor + 1

    def get_times_heard_rumor(self):
        return self.times_heard_rumor

    # Returns the list of students that this class has spread to. This
    # is used to ensure that a student that has heard of a rumor from someone cannot tell them the same rumor again
```

```
    def get_students_spread_to(self):
        return self.students_spread_rumor_to

    # Determines if this student is capable of spreading the rumor
    # and makes sure that this student hasn't already told the other the rumor
    # and that the other student likewise hasn't told this student the rumor
    def can_spread_rumor(self, student_id_to_tell_rumor_to, other_students_spread_rumor_to):
        if self.has_heard_rumor == True:
            if self.times_heard_rumor < 2:  # Magic number 2 here represents the prompt stating that
                                            # "After a person has heard the rumor 2 times, he/she will
                                            # assume everyone has heard the rumor and will no longer try to spread it further"
                if student_id_to_tell_rumor_to not in self.students_spread_rumor_to:
                    if self.get_id() not in other_students_spread_rumor_to:
                        # print(self.__repr__() + " can spread to Student ID: " + str(student_id_to_tell_rumor_to) + "!")
                        return True

        # print(self.__repr__() + " CANNOT spread to Student ID: " + str(student_id_to_tell_rumor_to))
        return False

def run_rumor_simulation(likelihood_of_spreading_rumor=0.5, number_of_students=100, minutes_to_run=10):
    if number_of_students % 2 != 0:
        print("Error: Input parameter number_of_students must be even. " + str(number_of_students) + " is not an even number.")
        return

    students = []
    # First, create an array of students
    for i in range(number_of_students):
        students.append(Student(student_id=i, times_heard_rumor=0, likelihood_of_spreading=likelihood_of_spreading_rumor))

    # Randomly set one student to have heard the rumor by setting times
    student_to_start_rumor = random.randint(0,number_of_students - 1)
    students[student_to_start_rumor].set_heard_rumor(True)

    # Run over every minute needed in given parameter minutes_to_run
    for minute in range(minutes_to_run):
        # Get random pairs of each student by randomly shuffling everybody and then pairing them into twos
        random.shuffle(students)
        # print("students == " + str(students))
        # Iterate over every pair and evaluate results
        for i in range(0, number_of_students, 2):
            # print("Pairing " + repr(students[i]) + " with " + repr(students[i + 1]))
            # Since initial student initially spreads rumor to first conversion partner at minute 0, make sure this happens here
            if minute == 0 and (students[i].get_id() == student_to_start_rumor or students[i + 1].get_id() == student_to_start_rumor):
                # Check first case where current even number student in shuffled students list is initial rumor spreader
                if students[i].get_id() == student_to_start_rumor:
                    students[i].append_students_spread_rumor_to(students[i + 1].get_id())
                    students[i + 1].increment_times_heard_rumor()
                    # print("First spread from initial " + repr(students[i]) + " to " + repr(students[i + 1]))
                # Else, it's second case where current odd number student in shuffled students list is initial rumor spreader
                else:
                    students[i + 1].append_students_spread_rumor_to(students[i].get_id())
                    students[i].increment_times_heard_rumor()
                    # print("First spread from initial " + repr(students[i + 1]) + " to " + repr(students[i]))
            else:
                # First case: First student in pair has heard rumor and second hasn't
                if students[i].has_heard_rumor and not students[i + 1].has_heard_rumor:
                    # Make sure that this student can logically spread rumor to other student
```

```
            if students[i].can_spread_rumor(students[i + 1].get_id(), students[i + 1].get_students_spread_to()):
                if random.random() < likelihood_of_spreading_rumor:
                    students[i].append_students_spread_rumor_to(students[i + 1].get_id())
                    students[i + 1].increment_times_heard_rumor()
                    # print("Spread from " + repr(students[i]) + " to " + repr(students[i + 1]))

        # Second case: First student in pair hasn't heard rumor but second student has
        elif not students[i].has_heard_rumor and students[i + 1].has_heard_rumor:
            # Make sure that this student can logically spread rumor to other student
            if students[i + 1].can_spread_rumor(students[i].get_id(), students[i].get_students_spread_to()):
                if random.random() < likelihood_of_spreading_rumor:
                    students[i + 1].append_students_spread_rumor_to(students[i].get_id())
                    students[i].increment_times_heard_rumor()
                    # print("Spread from " + repr(students[i + 1]) + " to " + repr(students[i]))

        # Third case: Both students have heard rumor already. This is the tricky one
        elif students[i].has_heard_rumor and students[i + 1].has_heard_rumor:
            # Check if both students have heard rumor based on assumption that even if both students
            # know the rumor already, they still have a random and equal chance of telling the rumor to the other
            # student in the pair again.
            # Caveat: If one student successfully tells other student rumor first, then
            #         the student who just got told the rumor can't retell the rumor to the same person

            # Randomly see who gets to try to spread the rumor first
            curr_pair_students = [i, i + 1]
            index_of_first_student_in_pair_to_attempt = random.choice([0, 1])
            first_student_to_attempt = curr_pair_students[index_of_first_student_in_pair_to_attempt]
            second_student_to_attempt = ""
            if index_of_first_student_in_pair_to_attempt == 0:
                second_student_to_attempt = curr_pair_students[1]
            else:
                second_student_to_attempt = curr_pair_students[0]

            first_student_successfully_spread_rumor = False

            # Make sure that this student can logically spread rumor to other student
            if students[first_student_to_attempt].can_spread_rumor(students[second_student_to_attempt].get_id(),
students[second_student_to_attempt].get_students_spread_to()):
                # Check if randomly generated float is less than likelihood of spreading rumor. If so, spread rumor
                # If true, spread rumor to student who has not heard rumor
                if random.random() < likelihood_of_spreading_rumor:
                    first_student_successfully_spread_rumor = True

students[first_student_to_attempt].append_students_spread_rumor_to(students[second_student_to_attempt].get_id())
                    students[second_student_to_attempt].increment_times_heard_rumor()
                    # print("Spread from " + repr(students[first_student_to_attempt]) + " to " +
repr(students[second_student_to_attempt]))

            # If first student's attempt at spreading rumor failed, then second student tries to spread rumor
            if not first_student_successfully_spread_rumor:
                # Make sure that this student can logically spread rumor to other student
                if students[second_student_to_attempt].can_spread_rumor(students[first_student_to_attempt].get_id(),
students[first_student_to_attempt].get_students_spread_to()):
                    # Check if randomly generated float is less than likelihood of spreading rumor. If so, spread rumor
                    # If true, spread rumor to student who has not heard rumor
                    if random.random() < likelihood_of_spreading_rumor:
                        first_student_successfully_spread_rumor = True
```

```
students[second_student_to_attempt].append_students_spread_rumor_to(students[first_student_to_attempt].get_id())
                        students[first_student_to_attempt].increment_times_heard_rumor()
                        # print("Spread from " + repr(students[second_student_to_attempt]) + " to " +
repr(students[first_student_to_attempt]))


    # Print out final results
    num_students_heard_rumor = 0
    for s in students:
        # print(s)
        if s.get_has_heard_rumor():
            num_students_heard_rumor = num_students_heard_rumor + 1

    # print("Number of the " + str(number_of_students) + " students who have heard the rumor is " + str(num_students_heard_rumor)
+ " after " + str(minutes_to_run) + " minutes.")
    return num_students_heard_rumor

N = 1000
minutes = 40
iterations_to_run = 1000
num_students_heard_rumor = []
time_and_percentage = []
for minute in range(40):
    num_students_heard_rumor = []
    for i in range(iterations_to_run):
        if i % 100 == 0:
            print("On iteration " + str(i))
        num_students_heard_rumor.append(run_rumor_simulation(likelihood_of_spreading_rumor=0.5, number_of_students=N,
minutes_to_run=minute))
    print("On minute " + str(minute))
    time_and_percentage.append(
        [minute, 100 * (sum(num_students_heard_rumor)/len(num_students_heard_rumor) / N)])

print("time_and_percentage == " + str(time_and_percentage))
print("After " + str(iterations_to_run) + " iterations over " + str(minutes) + " minutes for " + str(N) + " students:")
# print("Num_students_heard_rumor: " + str(num_students_heard_rumor))
print("Average number students who have heard rumor " + str(sum(num_students_heard_rumor)/len(num_students_heard_rumor)))
print("Average percentage of students who have heard rumor == " +
      str(sum(num_students_heard_rumor)/len(num_students_heard_rumor) / N) + " (" + str(100 *
(sum(num_students_heard_rumor)/len(num_students_heard_rumor) / N)) + "%)")

# # Graphing Time and :
# 10 iterations
# time_and_percentage = [[0, 0.01], [1, 0.02], [2, 0.03], [3, 0.049], [4, 0.059000000000000004], [5, 0.079], [6,
0.16699999999999998], [7, 0.255], [8, 0.303], [9, 0.551], [10, 0.75], [11, 1.111], [12, 2.006], [13, 2.5780000000000003], [14, 3.771],
[15, 5.806], [16, 7.994], [17, 14.094000000000001], [18, 14.482999999999999], [19, 21.349], [20, 30.377], [
#     21, 47.183], [22, 49.498999999999995], [23, 53.071], [24, 66.99199999999999], [25, 73.53], [26, 79.62199999999999], [27,
84.38799999999999], [28, 87.156], [29, 90.569], [30, 91.838], [31, 92.74700000000001], [32, 94.46700000000001], [33, 94.744],
[34, 95.785], [35, 96.004], [36, 96.431], [37, 96.875], [38, 97.075], [39, 97.34799999999998]]
# 100 iterations
# minutes = [x for x, y in time_and_percentage]
# percentages = [y for x, y in time_and_percentage]
# import matplotlib.pyplot as plt
# plt.plot(minutes, percentages)
# plt.ylabel('Average Percentage of Students Whom have Heard Rumor')
# plt.xlabel('Minute')
```

```
# plt.title('Minutes vs Average Percentage of Students Having Heard a Rumor')
# plt.show()
```