Report for exercise 2 from group E

Tasks addressed:	5
Authors:	Minxuan He (03764584) Gaurav Vaibhav (03766416) Vatsal Sharma (03784922) Zhitao Xu (03750803) Çağatay Gültekin (03775999)
Last compiled:	2023–11–16
Source code:	https://gitlab.lrz.de/mlcms-ex-group-e/mlcms-ex-group-e

The work on tasks was divided in the following way:

Minxuan He (03764584)	Task 1	100%
	Task 2	100%
Gaurav Vaibhav (03766416)	Task 3	100%
	Task 4	20%
Vatsal Sharma (03784922) **Project lead**	Task 5	50%
Zhitao Xu (03750803)	Task 5	50%
Çağatay Gültekin (03775999)	Task 4	80%

Report on task 1, Setting up the Vadere environment

Vadere

Vadere is an open-source simulation framework primarily used for modeling crowd dynamics. It provides generic model classes and visualization and data analysis tools. It's designed to simulate pedestrian movement in various scenarios, such as public spaces, buildings, or evacuation situations.
Compared with the first exercise, Vadere's visual operation allows users to set up various scenarios very easily. It also allows each simulation to automatically save the results as output.

To use Vadere, we have two ways. One is to download the compressed package from the website and double-click "vadere-gui.jar" to directly open the GUI. The other is to open it through an IDE. We need to download the Vadere ("master branch" version) from the website and need to install Java 11 or higher. Open the project in the IDE and run the corresponding main function to open it.

For task 1, we need to use the Optimal Steps Model to simulate three scenarios: the RiMEA scenarios 1 and 6 and the "chicken test".

1.1 RiMEA scenarios 1 (straight line)

RiMEA scenario 1 stipulates a 40-meter-long and 2-meter-wide corridor, with the pedestrian starting point on one side and the target endpoint on the other. A pedestrian can complete this distance within the corresponding time period at a prescribed walking speed. In this scenario, we set the pedestrian's speed to 1.34m/sec, so the travel time should be in the range of 26 to 34 seconds.
Select the OSM model by clicking on the "Model" - "Load template" tab in the GUI.
By looking at the results of the simulation (Figure 2), results of the Vadere simulation are almost identical to our results in Exercise 1, and the pass times are as expected, between 26 and 34 seconds.One difference is the trajectory of the two simulations. According to the Vadere visualisation of the trajectory, it can be seen that the pedestrian's trajectory is almost a straight line, whereas in Exercise 1 the pedestrian tends to walk closer to the wall once he starts to walk (Figure 1), this is because we did not take into account the distance from the obstacles, which results in a curved trajectory.

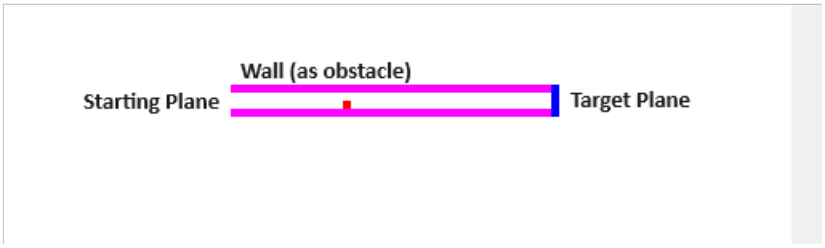
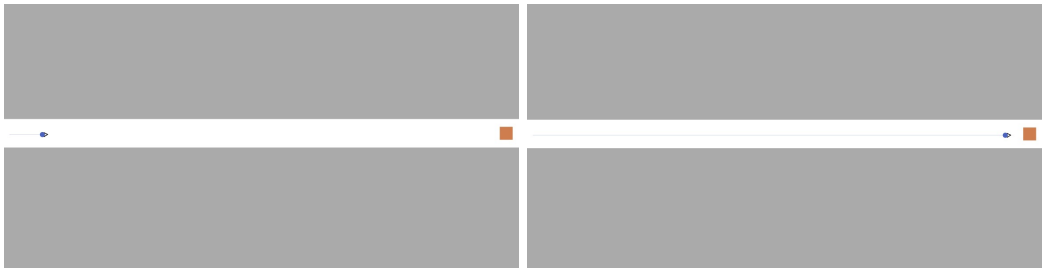


Figure 1: RiMEA scenarios 1 in Exercise 1

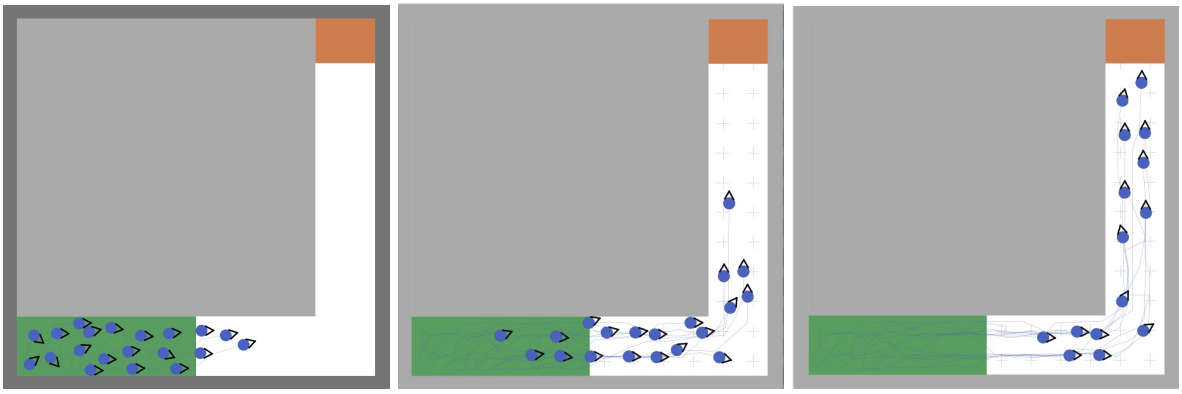


(a) It the beginning (b)Near the end

Figure 2: RiMEA scenarios 1 in Optimal Steps Model

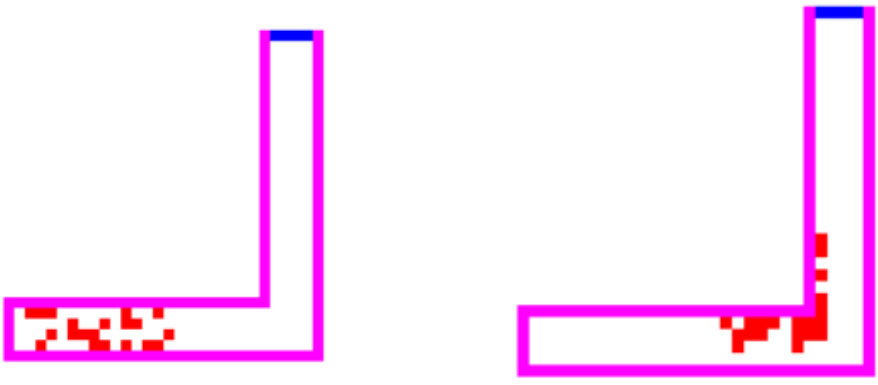
1.2 RiMEA scenarios 6 (corner)

RiMEA Scenario 6 requires 20 pedestrians to reach the target in the upper right corner without crossing the wall and turning left into the corner.
We used the lower and right edges of the canvas, as well as the square obstacle in the upper left corner, to construct the scene as shown in Figure 3. We set up 20 pedestrians equally divided in the green starting area of the corridor(This results in each pedestrian having a different distance to the target). It can be seen from the figure that all 20 pedestrians passed the test without passing through the wall.
The results of Vadere and Exercise 1 are still very similar, but the pedestrian movement is slightly different. We can see that the pedestrians in Figure 4 are not only walking against the wall but also seem to be turning towards the target in a crowded manner without caring about the distance between themselves and other pedestrians. In Vadere's OSM model, the pedestrians walk in an orderly fashion and seem to walk in pairs at most, without causing a crowded scene at the corners. Due to this mechanism, the simulation time of Vadere is longer than that of Exercise 1, and pedestrians will take a longer road.



(a) It the beginning (b) In the middle (c) Near the end

Figure 3: RiMEA scenarios 6 Optimal Steps Model



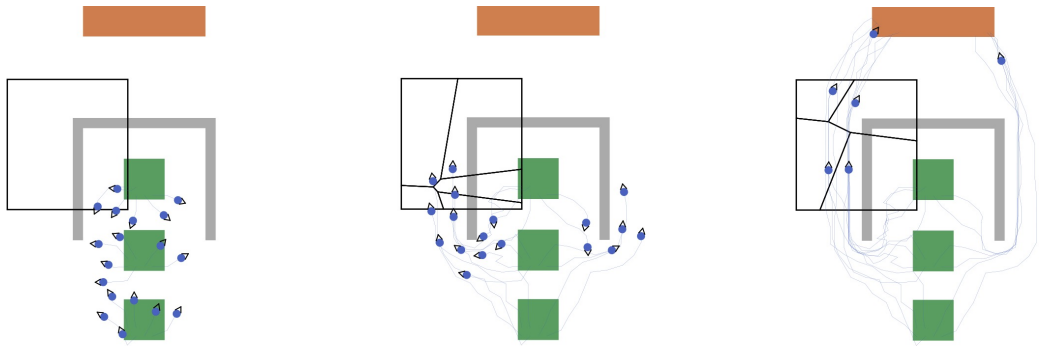
(a) It the beginning (b) Near the end

Figure 4: RiMEA scenarios 6 in Exercise 1

1.3 The "Chicken Test"

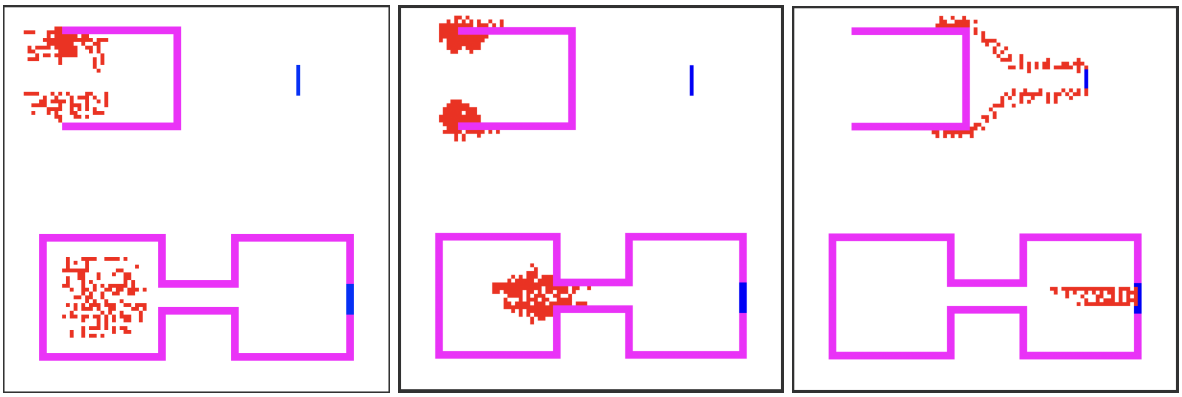
The scene of the "chicken test" is that there is a U-shaped obstacle blocking the straight line formed by a group of pedestrians and the target. The purpose of the obstacle is to trap the pedestrians so that they cannot reach their target.
We built a U-shaped obstacle in Vadere, and the target was placed above the center of the canvas. Three groups

of pedestrians were placed inside, in the middle, and outside of the U-shaped obstacle. This way we can better see the movement trajectories of the pedestrians.
As shown in Figure 5, all pedestrians successfully reached the target, which is consistent with the results of Exercise 1. However, what is obviously different from Exercise 1 is that in the simulation, pedestrians intentionally keep a distance from the U-shaped obstacle and move. Pedestrians inside avoid the wall and move toward the target in an orderly manner. Pedestrians in the centre and outside can also avoid obstacles rather than walking in a straight line leading to getting stuck. By maintaining distance between pedestrians and between pedestrians and obstacles, Vadere's simulation is closer to reality. But it also takes more time.
When the dijkstra or FMM algorithm is implemented in Exercise 1(Figure 6), you can see that the crowd is always crowded against the wall, especially at the corners. This simulation is more like the movement of a swarm of bees, or the movement of an extremely crowded crowd.



(a) It the beginning (b) In the middle (c) Near the end

Figure 5: Chicken Test in Optimal Steps Model



(a) It the beginning (b)In the middle (c) Near the end

Figure 6: "Chicken Test" and "Bottleneck Test" in Exercise 1

Summary of comparisons between cellular automaton implementation and optimal step models in Vadere

In terms of user interface, we only implemented simple buttons with basic functionality. In contrast, Vadere provides more functions. Under the function of user-defined scenes, Vadere's optimization is better and more intuitive.
In terms of model trajectory, as mentioned above, Vadere's simulation always considers the distance between pedestrians and the distance between pedestrians and obstacles. The trajectory is not consistent with our simulation.
The biggest difference between the two is the degree of visualization. Vadere provides multiple options such as

trajectory, movement direction, different groupings, step lengths, etc. But we only implement the movement of pedestrians. The visual information is far less than that of Vadere.

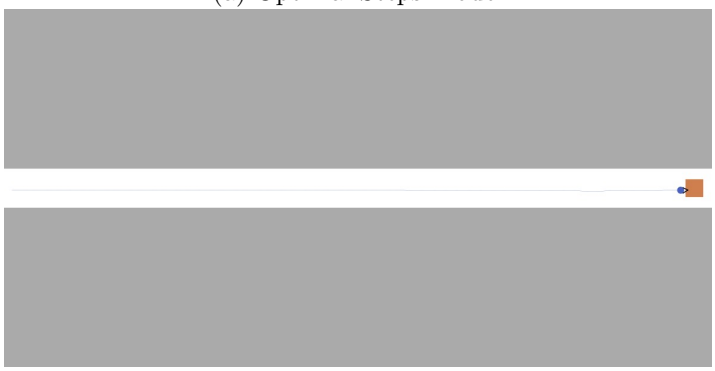
Report on task 2, Simulation of the scenario with a different model

In Task 1, we used optimal step models to simulate three scenarios, and in Task 2, we needed to use the Social Force Model and the Gradient Navigation Model to simulate the same three scenarios respectively and conduct a comparative analysis on them.
Select the SFM and GNM models by clicking the "Model" - "Load template" tab in the GUI.

2.1 RiMEA scenarios 1 (straight line)

For the simple scenario of a pedestrian crossing a straight corridor, the results of the three models are very similar(Figure 7), the difference being in the trajectory. OSM is a straight line, and SFM has a slightly inclined trajectory. The SFM trajectory is slightly inclined, whereas the GNM clearly shows the pedestrian slanting towards the wall, a complete diagonal.

(a) Optimal Steps Model



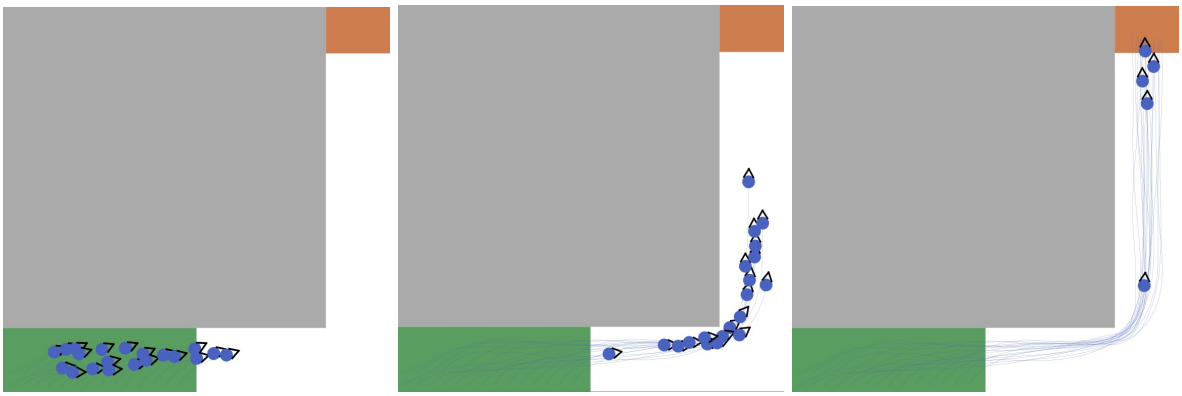
(b) Social Force Model

(c) Gradient Navigation Model

Figure 7: RiMEA scenarios 1

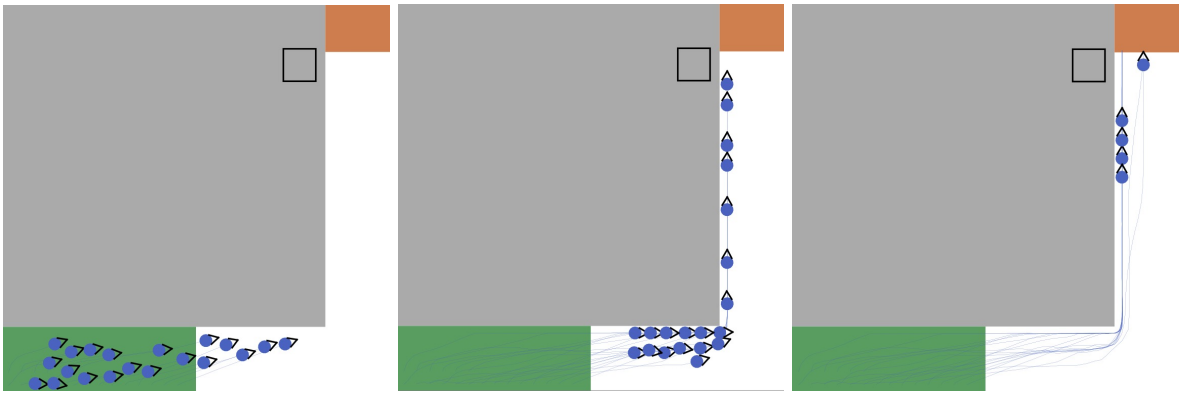
2.2 RiMEA scenarios 6 (corner)

For the Social Force Model(Figure 8), it can be seen that pedestrians are less sensitive to the distance between other people and the wall, two or three people can walk side by side, and it is the shortest time among the three models.
For the Gradient Navigation Model(Figure 9), we can see that pedestrians line up one by one, turning and moving forward near the wall in an orderly manner. When the person in front prevents the person behind from passing the curve, the person behind will reduce the speed and wait, following the trajectory of the pedestrian in front. The generated trajectory lines are few and clear, so it takes the longest time, 35.2 seconds.
Compared with the OSM of Figure 3, the trajectory of SFM has fewer folds, which means that the pedestrians' traveling distance is shorter. So OSM takes slightly more time than SFM. OSM is faster than GNM because pedestrians blocked by GNM will slow down and wait to pass, while pedestrians in OSM will choose a new road.



(a) It the beginning (b) In the middle (c) Near the end

Figure 8: RiMEA scenarios 6 in Social Force Model



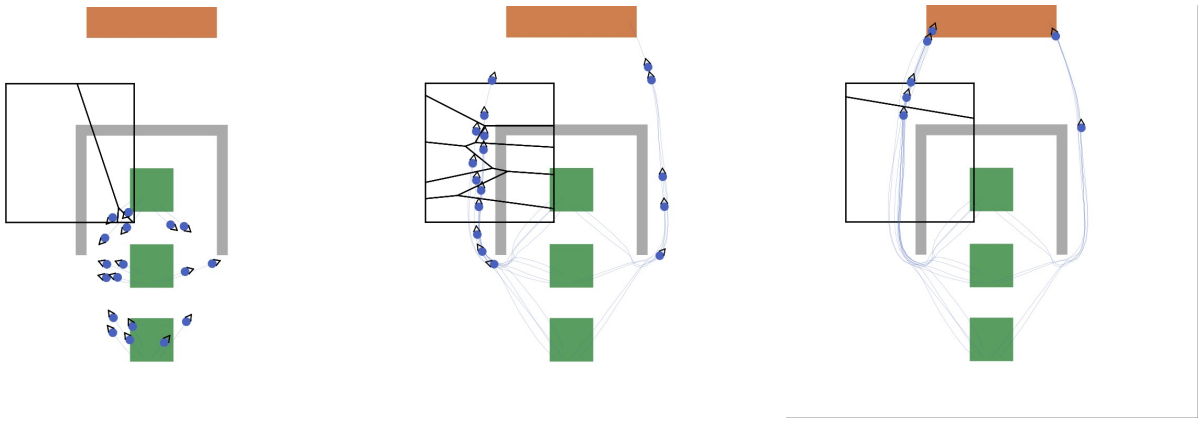
(a) It the beginning (b) In the middle (c) Near the end

Figure 9: RiMEA scenarios 6 in Gradient Navigation Model

2.3 The "Chicken Test"

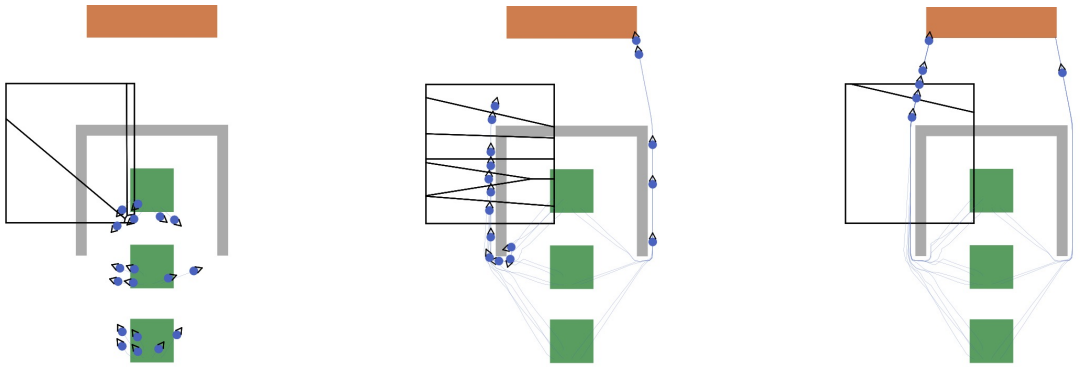
For the Social Force Model(Figure 10), which is still the shortest model, we can see that its trajectory is smooth without too many corners, and pedestrians take the shortest path. The distance between pedestrians and between pedestrians and walls is maintained.
For the Gradient Navigation Model(Figure 11), the trajectories are much less frequent and are straight lines, with pedestrians lining up one after the other to bypass the U-shaped obstacle and walk towards the target. The most obvious is that the trajectories between the obstacle and the target are almost two diagonal straight lines. The SFM and OSM models can still find different pedestrian trajectories.

Under the "chicken test", the times for SFM, GNM, and OSM were 22, 24.8, and 26.5 seconds, respectively. A closer look at figure 5 shows that the pedestrians of OSM, in order to keep their distance, stopped at the exit of the U-shaped obstacle and waited for the previous person to leave causing congestion, which we guess is the reason for its slowest result. Although OSM is the slowest, the fact is that all three models pass the "chicken test" very well, and compared to scenario 6, there is not much difference in the time spent among the three of them.



(a) It the beginning (b) In the middle (c) Near the end

Figure 10: "Chicken Test" in Social Force Model



(a) It the beginning (b) In the middle (c) Near the end

Figure 11: "Chicken Test" in Gradient Navigation Model

Comparative analysis

According to the above pictures and analysis, it can be concluded that models significantly affect the distribution of trajectories. The pedestrians of the three models seem to move in similar directions, but in fact, the trajectory curves and time consumption are different.
OSM and SFM are similar in terms of time spent and pedestrian movement. OSM is more sensitive to spacing. It would rather increase its time consumption and walk longer distances to get more space, while SFM focuses on reaching the target first. GNM enforces stricter path separation and does not allow pedestrians to choose new paths for avoidance purposes, which is more likely to cause congestion and takes slightly longer than the other two methods.

Report on task 3, Using the console interface from Vadere

3.1 Comparison of GUI and console outputs

The vadere-console.jar was used to run a single scenario file in the following way with the vadere-console.jar file in the working folder:

```
java -jar vadere-console.jar scenario-run --output-dir ./output --scenario-file ./scenario/scenario{\_}
```

The generated output file was comapred with the ones from GUI. Both exactly matches with each other. To illustrate this, the travel time of pedestrians of scenario 6 was plotted for both GUI and console. Figure 12 shows this.

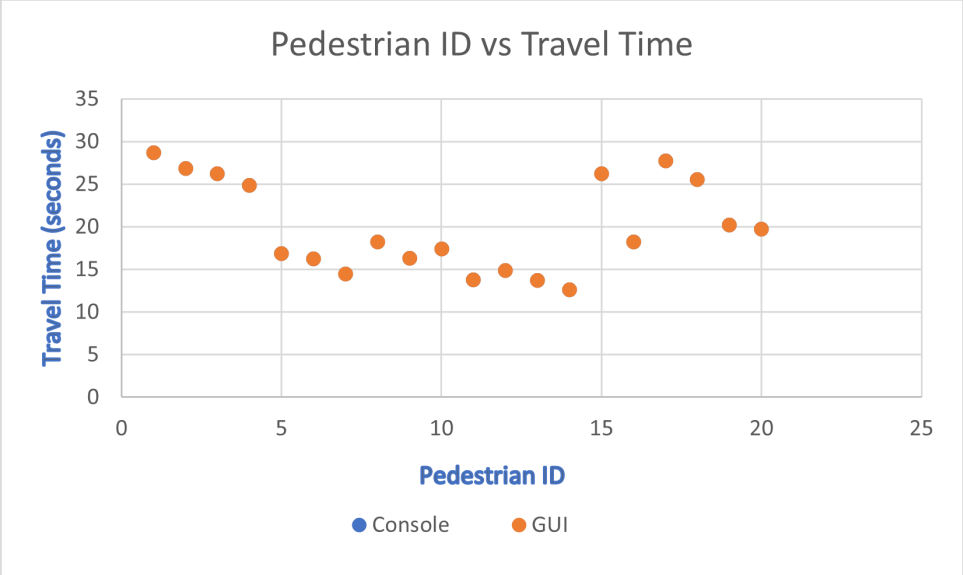


Figure 12: Travel time of pedestrians for Console Vs GUI

3.2 Adding pedestrians programmatically

In order to add a pedestrian, Pyhton program was used to modify the scenario file and run Vadere console using this modified file. Class Scenariofile was created with modules as:

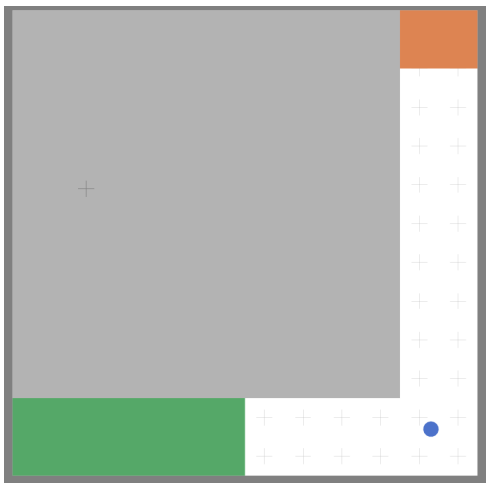


Figure 13: Added Pedestrian Scenario

- read_scenariofile(filename) Load the scenario json file

- write_scenariofile(filename,data): Modify the scenario file
- format_scenariofile(filename): Format the scenario file in consistent with Vadere requirements
- fix_attributes(target_id): Defining hard-coded (some) pedestrian attributes in dictionary format including required position coordinate as per this task

Jupter Notebook file containing main code was created. Program Workflow can be described as below:

- Reading the scenario file in dictionary variable 'add_pedestrian' using 'read_scenariofile()'
- Modifying the scenario name using key 'name' of 'add_pedestrian' dict
- Inheriting the 'attributesPedestrian' dict to append into the 'dynamicElements' list
- Reading target ID from 'add_pedestrian' dict to add it to pedestrian target ID
- Pedestrian ID is also updated using 'id' key which is then appended to 'attributesPedestrian' dict followed by again appending dict returned from 'fix_attributes module'
- Final appended dict is stored in "scenario" - "topography" - "dynamicElements" list of scenario file
- 'add_pedestrian' dict is dumped into the modified scenario file using 'write_scenariofile module'
- Modified scenario file is formatted using 'format_scenariofile module' for Vadere input file
- Running the modified scenario file through the console

```
os.system('java -jar vadere-console.jar scenario-run --output-dir ./output --scenario-file ./scena
```

3.3 Results and Discussion

The pedestrian was added at position coordinate x = 10.7 and y = 2.1. After running the modified scenario file through the program, new pedestrian took 7.21 seconds to reach the target while the mean travel time for the source pedestrians was around 20 seconds. This was expected as the added pedestrian was already ahead of the other source pedestrians in reaching the target and free from crowd effects.
There were no differences observed in the output files of the GUI and the simulation runs with the programmatically modified scenarion file as the final inputs to the software simulation are same.

REPORT TEXT.

Report on task 4, Integrating a new model

4.1 Integrate the SIR model in Vadere and Description of the SIR Model

First of all, we cloned the Vadere repository and installed IntelliJ. After transferring the files into the appropriate directories, we were able to successfully integrate the SIR model and execute Vadere from the IDE using the SIR model.

Description of the SIR Model

- Other than running circle, there are some initializer, getter, and setter functions as in every Java project. These are vital for project runs.
- In the model, preLoop will be called first then it will call "initializeGroupsOfInitialPedestrians" function. A list of DynamicElement-Containers of type Pedestrian is returned by this function once n pedestrians are randomly initialized.
- Every timestep, the "update" function is run to update the pedestrian's current status. This function first, checks the positions of all pedestrians and switches groups to INFECTED (or REMOVED). Then, it loops over neighbors and sets infected if they are close.
- The "getFreeGroupId" function adds a groupId to a dictionary called getGroupsById. Then, at the beginning of the simulation, increase the number of infected and recovered individuals.
- The "assignToGroup" function gets the pedestrian's own instance as a parameter and a new groupID is randomly assigned to it in accordance with the specified initial conditions if the pedestrian has no group assigned. If not, the pedestrian is just assigned to the groupID that is supplied as a function parameter and is taken out of its current group.

4.2 Output Processors

Processor 5 was utilized to record details about our SIR model and its characteristics. This also creates a SIRInformation.csv file, which aids in the visualization of the number of pedestrian groups over time. Then, we use the Python file on Moodle to visualize the CSV file.

4.3 Visualization of the Groups

The code was not working properly as stated in the problem description so we changed the code as follows:

Figure 14: Source Code for changing colors

So, it becomes as follows:

- 0. RED is for Infective
- 1. BLUE is for Susceptible
- 2. GREEN is for Recovered

As mentioned in the project description there are three ways of visualization, in our case, the visualization occurs during the simulation but does not occur in the after-simulation visualization phase.

4.4

Pseudocode for improving the efficiency of LinkedGridCells:

The modified approach can be a method which takes position of the pedestrian and the infection distance and returns the neighbours for that pedestrian in that radius. Following, the group of the pedestrian is determined. The pedestrian continues to be in infected or recovered state if they were already in that state else if they are from the susceptible group, the code iterates over its neighbours and if any neighbours are infected, the pedestrian is assigned to the infected group if the generated random number value is less than infected rate.

```
for each pedestrian do
    if pedestrian.group == "susceptible" then
        neighbours = get neighbours using position and infection radius
        for all neighbours do
            if neighbour is infected then
                num = generate a random number
                if num < infection rate then
                    set group of pedestrian to infected
                    break
                end if
            end if
        end for
    end if
end for
```

4.5 Some Test Cases

4.5.1 Static Scenario with Default Settings

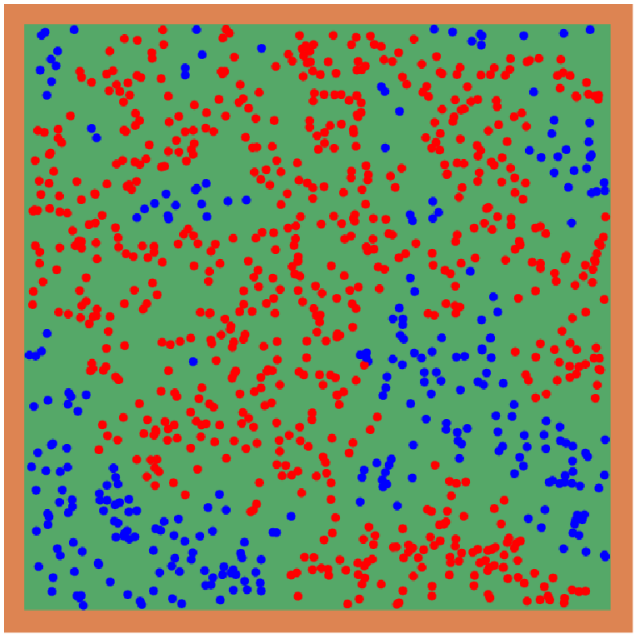
As required by the task instructions and resembling the figure provided in the exercise sheet, we generated a static scenario with 1000 pedestrians. The Boolean variable absorbing is set to "false" to indicate that the target is not absorbing. Its dimensions are 28 meters in height and width, and it is located at (1, 1). The source's parameters were set to "eventPositionRandom" to "true" and "eventPositionFreeSpace" to "false". It was positioned at (2, 2) and had a height and width of 26 meters, allowing for easy observation of the boundaries between the source and target. To generate a total of 1000 pedestrians, the "spawnNumber" was set to 1000.

On the other hand, the parameters set in the Model of the simulation are as follows:

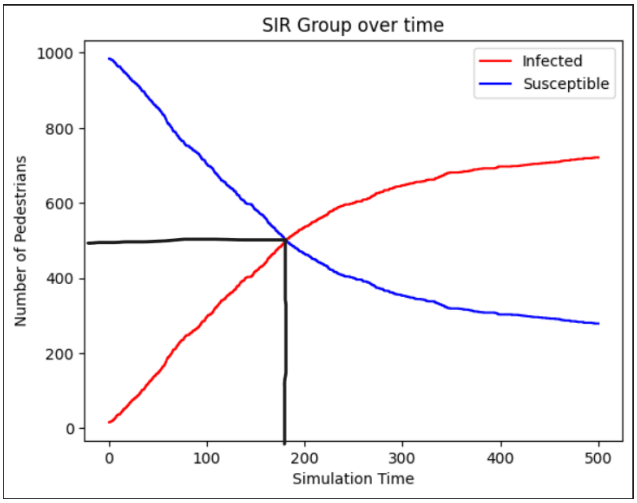
- "infectionsAtStart" : 10,
- "infectionRate" : 0.01,
- "infectionMaxDistance" : 1.0

This is how it looks at the end of the simulation and how the infected and susceptible pedestrians look like in simulation time15:

As you can see from the graph, in order to half of the pedestrian gets infected, 190 simulation time should be passed.



(a) Final Look



(b) Over-time Graph

Figure 15: Default Settings Test

4.5.2 Static Scenario with Increased Infection Rate

The only change here we did is we increased the infection rate from 0.01 to 0.02. However, significant changes happened in the time half of the pedestrians were infected. There is figure 16 and the final version of the area.

In order to infect half of the pedestrians, only 55 simulation times have become enough in this setting.

4.5.3 Corridor Test

In order to do this test, we finalized a 40mx20m area with two sources and two targets. Also, we did eventPositionFreeSpace:true as required. This is the plot 17 and in simulation look at the test:

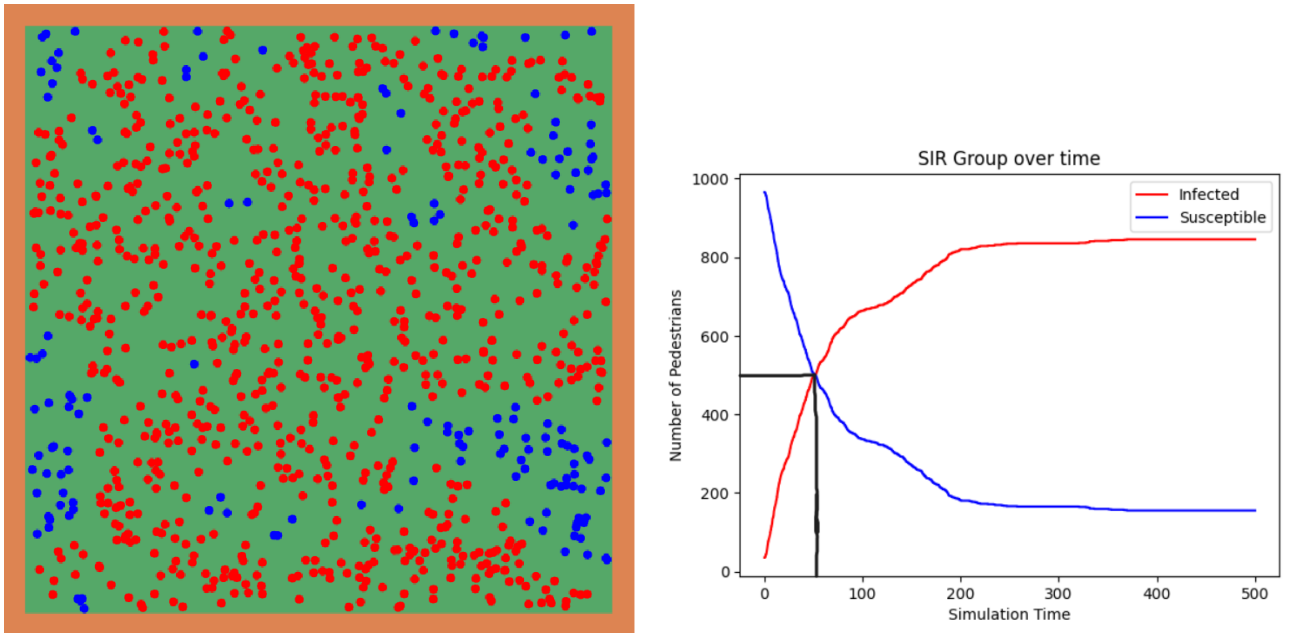
In this test, only 5 pedestrians became infected.

4.6 Decoupling the infection rate and the time step

As far as we can see in this application, the step size application causes each pedestrian in the simulation to change groups in certain situations, that is, to change the simulation image. Our suggestion is that we can fix the relation between infection rate and time step according to reel data in the reel simulations. We don't have any reel data of the reel simulations, so we decided to use default settings as the reference point. So, if we take the infection rate as 0.01 and the time step length as 0.4. In our suggestion, infection rate, and time step length multiplication should always give 0.04 so all the possible simulations can have a fixed way. The reason we are using multiplication is when the time step length gets less, the pedestrian's possibility of getting infected increases because it will divide the time of the simulation more and it will update the pedestrians' situation more in a way. On the other hand, when we decrease the possibility of infection, the possibility of the pedestrian getting infected is decreasing.

4.7 Possible Extensions

- For more realistic implementation, pedestrians can be grouped. For example, some of the pedestrians' possibility of being removed from the simulation can be higher because some people in real life have some diseases or are old so they are more likely to die when they are infected.



(a) Final Look (b) Over-time Graph

Figure 16: Increased Infection Rate Settings Test

- For again more realistic implementation, we can add the possibility of a person being vaccinated. In this simulation, the only possible way to be recovered is by being infected first but in real life, we can be recovered by the vaccine. Also, it is possible that recovery by a vaccine can be a function that getting more possible by time step.
- To add more possibilities, we can add more people who can be more responsible. We can add some buttons so it will change the infection rate when we click the button. The button can mean that people will wear more masks, and stand far from other people. This means in simulation our pedestrians will try to add distance to other pedestrians and the infection rate will be decreased.

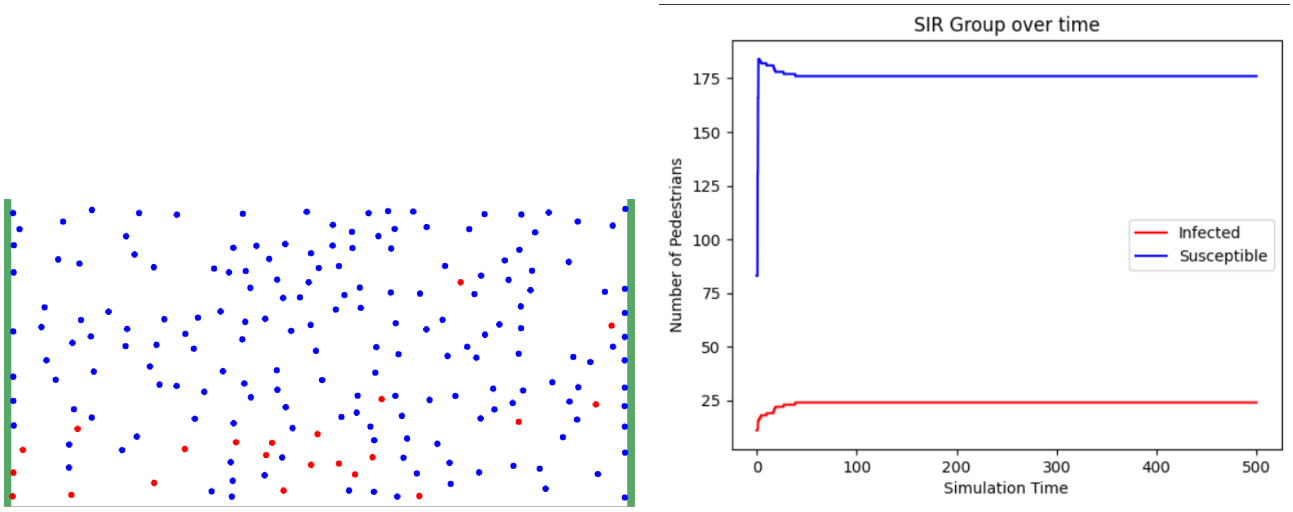
Report on task 5, Analysis and visualization of results

5.1 Adding recovered state

In the previous tasks, pedestrians have "infected" and "susceptible" states. In this task, we will add a "recovered" state to let the infected pedestrians have a chance to recover during the simulation. The change of the state to "recovered" can only happen to an "inffected" pedestrian, which means that a "susceptible" pedestrian cannot get "recovered" directly. In each time step, the "infected" pedestrians have a probability to recover, which we call `RecoveryRate`. This is indepedent of where it is and the neighbors around it. Once an infected pedestrian get "recovered", it cannot infect other pedestrians and it also cannot be re-infected. We added the following code in the outer `for` loop of `update` method in `org.vadere.simulator.models.groups.sir.SIRGroupModel.java`.

```
SIRGroup g = getGroup(p);
if (g.getID() == SIRType.ID_INFECTED.ordinal() &&
        this.random.nextDouble() < attributesSIRG.getRecoveryRate()) {
    elementRemoved(p);
    assignToGroup(p, SIRType.ID_RECOVERED.ordinal());
    this.totalInfected -= 1;
    this.totalRecovered += 1;
```

First, we obtain the group of the current pedestrian. Then we test two conditions, which are 1) if the group is "infected" and 2) if a generated random number (between 0 and 1) is less than the given `RecoveryRate`. If



(a) In-simulation Look (b) Over-time Graph

Figure 17: Corridor Test

these two conditions are satisfied simultaneously, then we change the state of the current pedestrian to the state "recovered". After that, we change the member variables `totalInfected` decreasing by 1 and `totalRecovered` increasing by 1, which are used to count the number of pedestrians from the corresponding group.

5.2 Visualization of the results

We decide to implement the visualization part by ourselves in Python with the package `Matplotlib`. As we know, if we are using `pyplot` to create a simple curve, we need to provide the function `plt.plot` with two 1-d arrays of the same shape. But we found that there are always pedestrians missing at each time step (except the first time step) in the output file `SIRinformation.csv` generated by the `FootStepGroupIDProcessor`. This would result in errors when we try to visaulize the number of pedestrians from each group over time. Therefore, we need to perform the data cleaning first. Since there are no missing pedestrians at the first time step, our strategy is to add the missing pedestrians at each time step and assign the group information to them based on the previous time step. Here we make use of the method `ffill` to efficiently achieve this goal. Then we calculate the number of pedestrians from each group by performing some operations on `Numpy` arrays, which gives us a 2-d array of the shape `(3, the number of all time steps)`, the 3 rows of whom represent the corresponding states respectively.

After the data cleaning, we define a function to create curves of the number of pedestrians from each group against all time steps. The "Infected", "Susceptible", and "Recovered" states are illustrated in red, blue, and green colors respectively, which are of the same colors as the circular pedestrians in the visualization of the scenario simulation.

The code of visualization is uploaded to the directory `TO ADD!` on GitHub for reference.

5.3 Test 1: Static scenario of 1000 pedestrians

In this test, we need to construct a large square scenario spawning 1000 static pedestrians. To achieve this scenario, we first set the width and the height of the "bounds" attributes in the `Topography` tab of the GUI to both 50 to create a fairly large background. Then in the `Topography creator` tab, we can see this white square background. On this background, we create a square `source` positioned at $(10, 10)$ with the shape 30×30 and a square `target` positioned at $(5, 5)$ with the shape 40×40. They are shown in Fig. 18, and the source is illustrated in green, while the target is shown in orange. As we can see, through this layout, the centers of the two squares coincide, and they are both at the center of the background.

After that, we set the parameter `eventElementCount` of the source to 1000 to create 1000 pedestrians, and in order to make the pedestrians do not vanish during the simulation, we also set the parameter `enabled` of

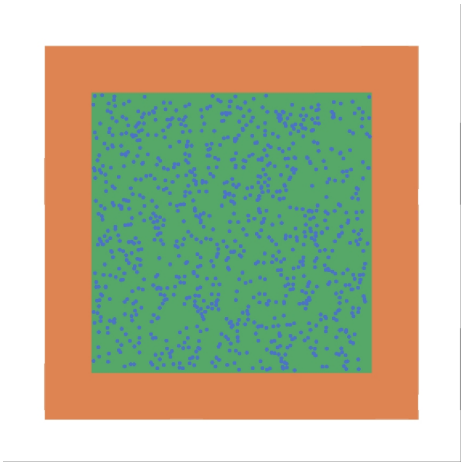


Figure 18: The static square scenario spawning 1000 pedestrians.

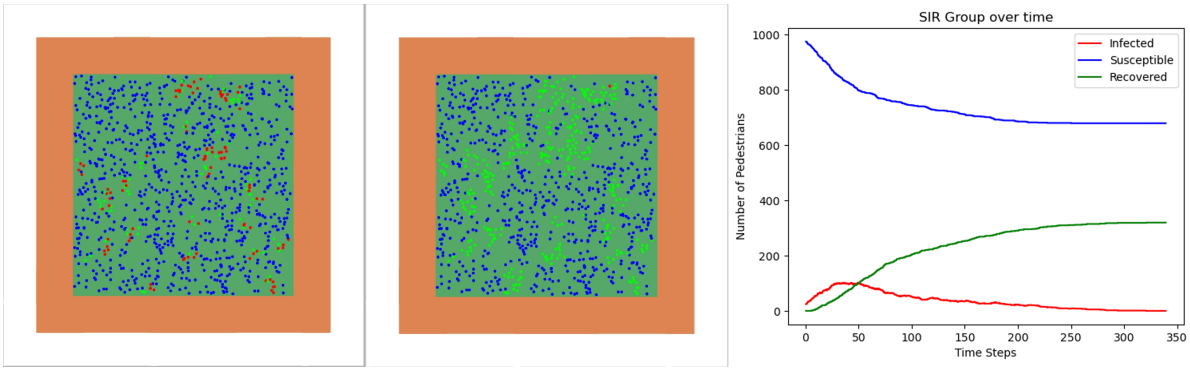


Figure 19: Source size: 30*30, infectionRate=0.01, recoveryRate=0.01. Left: during the simulation. Middle: end of the simulation. Right: SIR group information over time.

`absorber` of the target to false. Furthermore, we set the two parameters of source: the `eventPositionRandom` to true and the `eventPositionFreeSpace` to false to make these 1000 pedestrians distribute randomly and evenly in the source. We also set the `leavingSpeed` to -1.0 to make sure that the output processor continues writing the group IDs even though no pedestrians move. Finally, we add the following settings to the `Model` tab to start with 10 pedestrians infected.

```
  "org.vadere.state.attributes.models.AttributesSIRG" : {
"infectionsAtStart" : 10,
"infectionRate" : 0.01,
"recoveryRate" : 0.01,
"infectionMaxDistance" : 1.0
  },
```

The `infectionRate` and the `recoveryRate` are both set to 0.01 initially. The `infectionMaxDistance` is set to 1.0, which means for how far do we check if there is a infected neighbor around a pedestrian. For example, in the circular area with radius 1.0 around a susceptible pedestrian, if there exists an infected pedestrian, then this susceptible pedestrian can get infected with the probability 0.01. After it gets infected, it can get recovered with the probability 0.01 at each time step, no matter where it is and its neighbors' states. The visualization of the scenario under this settings and its group information are shown in Fig. 19.

5.4 Test 2: Experiment with the infection and recovery rate.

In this test, we need to simulate scenarios with different settings of infection and recovery rate. But before that, we first perform a pre-test to find out the suitable size of the source for the experiment. Based on the settings

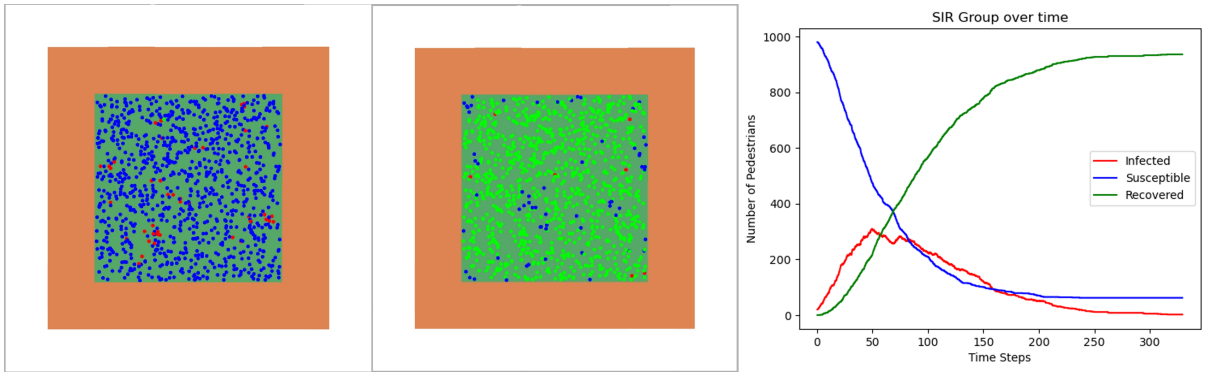


Figure 20: Source size: 20*20, infectionRate=0.01, recoveryRate=0.01. Left: during the simulation. Middle: end of the simulation. Right: SIR group information over time.

of the previous static scenario, we only change the sizes of the white background, source, and target decreasing by 10, respectively. The results are shown in Fig. 22.

In both scenarios, we stop the simulation at roughly the same time step, which is approximately 320 time steps, because till such time step, both simulations are stable. If we compare the Fig. 22 and the Fig. 19, we can find that, except the sizes, everything else is the same, but the results are quite different. Under this setting, almost all pedestrians get infected and then recovered, but under the previous setting, only a small part of pedestrians get infected and then recovered. This significant difference is caused by the change of the sizes. Since the pedestrians are evenly distributed, in the smaller source, generally the distances between pedestrians are relatively smaller than the ones in the larger source. As the `infectionMaxDistance` is fixed, in the smaller source, there are more likely to exist infected pedestrians around a susceptible pedestrian. Therefore, in the smaller square source, pedestrians have higher probability to get infected in general. For this test, we only care about how `infectionRate` and `recoveryRate` affect the result, so if we use the smaller source, we cannot tell how these two parameters work, because when the simulation becomes stable, always nearly all pedestrians get infected. As a result, we decide to select the previous scenario as a base for the experiment.

The results of the experiment is shown in Fig. 21. In this experiment, we simulate different scenarios with the following settings respectively:

- `infectionRate`=0.1, `recoveryRate`=0.01, for the first row scenario;
- `infectionRate`=0.01, `recoveryRate`=0.1, for the second row scenario;
- `infectionRate`=0.1, `recoveryRate`=0.1, for the third row scenario;

These scenarios get stable earlier than the base scenario. Nevertheless, we still stop the simulation at approximately 320 time steps for a fair and intuitive comparison with the base scenario. Now we will compare each of these scenarios with the base scenario, and analyze their differences.

1. If we increase the `infectionRate` to 0.1, comparing the first row of the Fig. 21 with the Fig. 19, we can find that, significantly more pedestrians get infected, and then eventually get recovered. This result is quite intuitive. As we can imagine, COVID-19 virus has been found to be more contagious than the common cold virus, which means its infection rate is relatively high, satisfying current scenario, so the COVID-19 virus spread rapidly among the population, resulting in many people becoming infected.

2. If we increase the `recoveryRate` to 0.1, in the second row of the Fig. 21, we can find that, only very few people get infected, and then at about 20 time steps, the scenario becomes stable. This is because the infected people can easily recover. Consequently, the virus has a very low chance to spread among the crowd.

3. If we increase both the `infectionRate` and the `recoveryRate` to 0.1, comparing the third row with the base scenario, we can find that, relatively more pedestrians get infected and then recovered. This is due to the higher `infectionRate`, so the virus spread faster than normal. Besides, this scenario also becomes stable much earlier, because of the higher `recoveryRate`, and the recovered people cannot get re-infected, and cannot infect others.

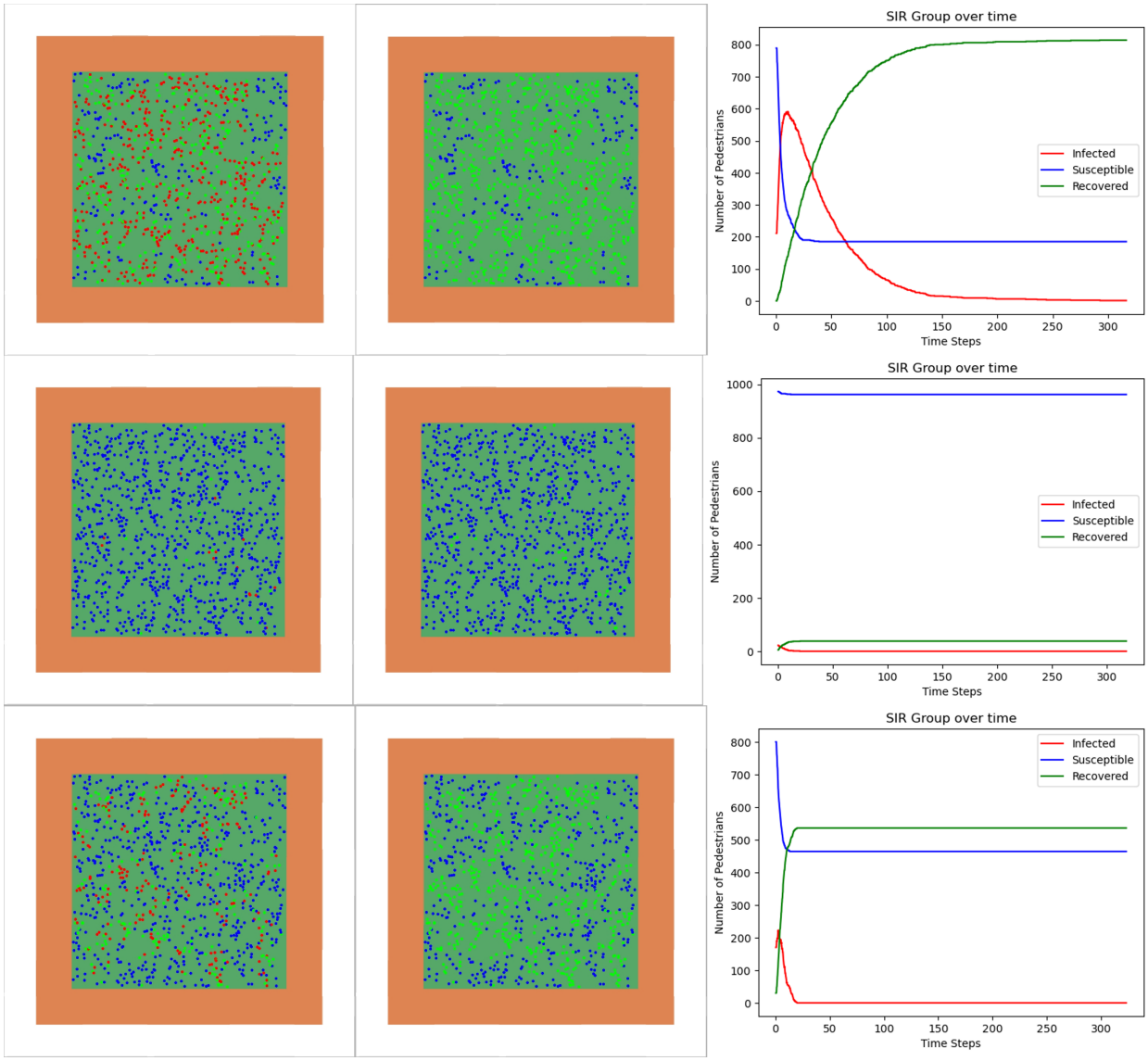


Figure 21: Left column: during the simulation. Middle column: end of the simulation. Right column: SIR group information over time. First row: `infectionRate`=0.1, `recoveryRate`=0.01. Second row: `infectionRate`=0.01, `recoveryRate`=0.1. Third row: `infectionRate`=0.1, `recoveryRate`=0.1.

5.5 Test 3: The Supermarket Scenario

In this, we evaluate the functionality of our previously constructed SIR Model through its implementation in a practical supermarket setting. Please see the detailed labelling and details of the scenario inf the figure below.

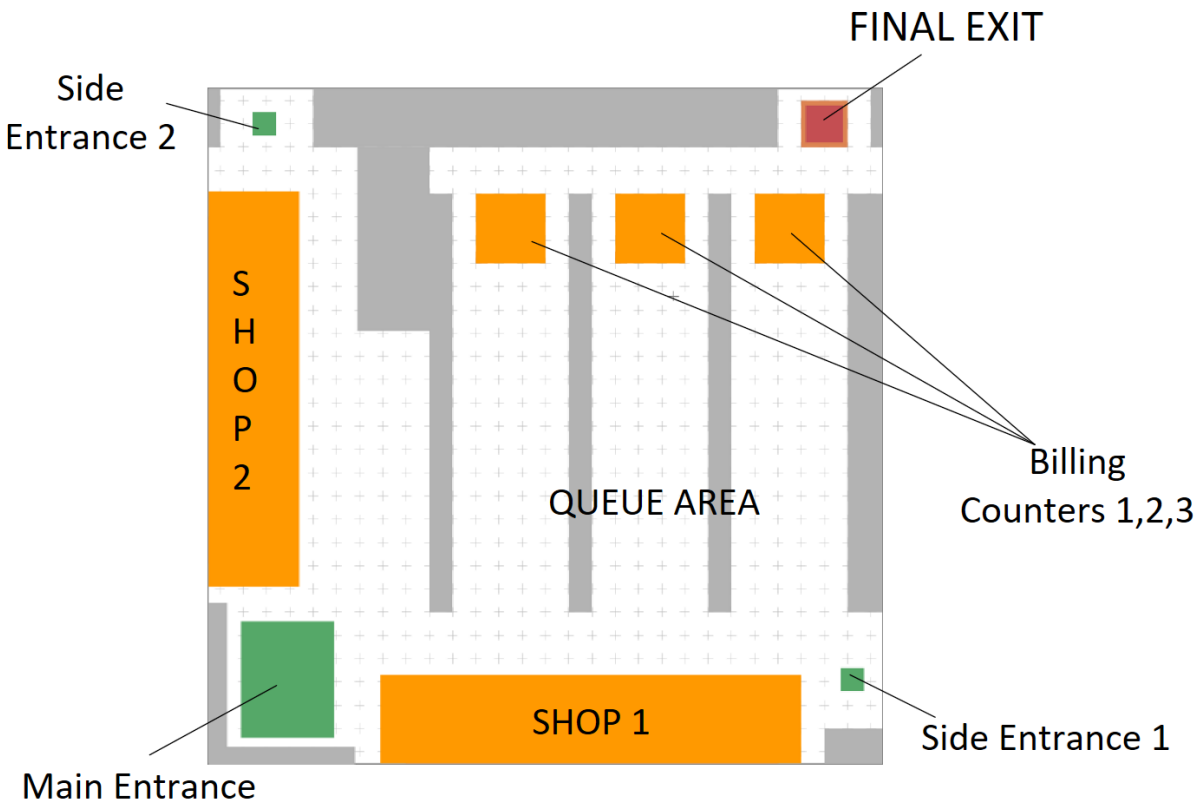


Figure 22: Supermarket Scenario

Details of the Scenario

- Dimensions - 30 x 30
- Number of customers to be spawned - 450 (150 from each source)
- Number of infected people at the start - 20
- Infection Rate - 0.01
- Recovery Rate - 0 [1]
- infectionMaxDistance - 1.0
- Number of absorbing areas - 1 (placed above the final exit)
- Number of sources - 3 (1 main entrance, 2 side entrances)

[1] We intend to show only the spread of infection and hence recovery parameter is disabled for further simulation and visualization purposes

- Number of targets - 6
  - 2 (SHOP1 and SHOP2)
  - 3 (Billing Counters)
  - 1 (Final Exit)
- Number of target changers - 5 (all placed above the targets)
- Number of obstacles - 12 (interspersed throughout the scenario)

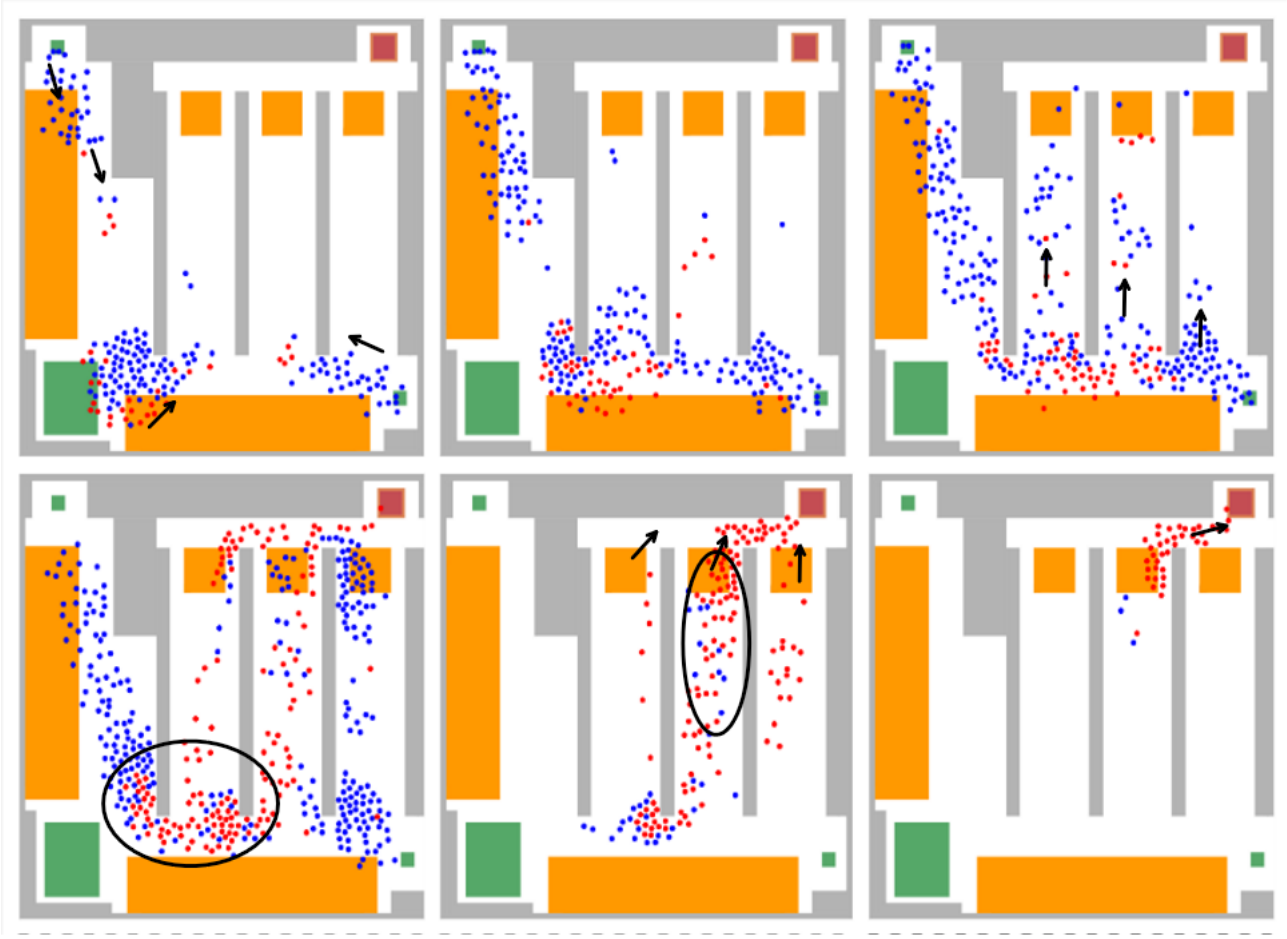


Figure 23: Simulation steps

Please see the flow of simulation in figures as

$$[1 \rightarrow 2 \rightarrow 3]$$
$$[4 \rightarrow 5 \rightarrow 6]$$

Arrows have been used to depict the customer flow.The significance of the ovals will be duly considered during the process of observation and inference generation in relation to this test. We can clearly see how the infection grows, spreads and affects a larger fraction of the crowd with every passing time step.

PLEASE NOTE - While explaining the simulation below, a number can be found at the end of certain points in brackets, which will point to the simulation snapshots in the above figure 23. Do consider it for references.

Explanation of Simulation

- The above configuration primarily has 3 entrances (sources) and 2 shops (targets) to depict customer flow.
- Main entrance spawns a bigger crowd than the remaining entrances, owing to its size. [1]
- Spawned customers have the shops as their targets and they start moving towards them. [1]
- Customers tend to move to towards their respective target shops from the three entrances and in the course hit the target changers, which redirects their path to the billing counter squares. [2] [3]
- A random billing counter is chosen out of a uniform distribution for the customer groups to evenly divide the crowd and put it into queues.
- This arrangement in turn gives out the effect, that the customers are done with their shopping and wish to complete their billing now. Also, it helps create a variety of trajectories for the different groups.
- The crowd heads to the queuing area and moves in paths created by obstacles placed around them. [4]
- As soon as they hit the billing counter squares, they encounter the target changers again and their target is re-modified to the "FINAL EXIT" square. [5]
- The "FINAL EXIT" square is equipped with an absorbing area, so as to avoid excessive crowding and efficiently eliminate pedestrians from the scenario. [6]

Parameters, Testing and Possible Solutions

In this section, we list down all the numerical and non-numerical factors that control the simulation results, what effects they cause, and how can we devise possible solutions out of them in order to contain the spread of infections in this supermarket scenario.

1. **Number of people**
   This is the simplest parameter. Nonetheless, decreasing the number of people in the crowd, definitely helps us contain the infection.

2. **Area of the scenario**
   Bigger the marketplace, lesser is the chance for people to contact each other and spread the infection.

3. **Obstructions by obstacles**
   This is a hidden but important parameter. We can clearly see in figure 24 that having longer obstacles causes congestion and particularly for a crowd (like in the simulation) chances of infection increase greatly. While shorter and smaller obstacles also constrain the crowd to move in certain trajectories, they do so in a relatively free and unrestricted manner, in the same area.
   SOLUTION - The imperative is not solely to expand spatial dimensions of public areas but also to ensure unimpeded accessibility, devoid of substantial obstructions that contribute to congestion in the end.

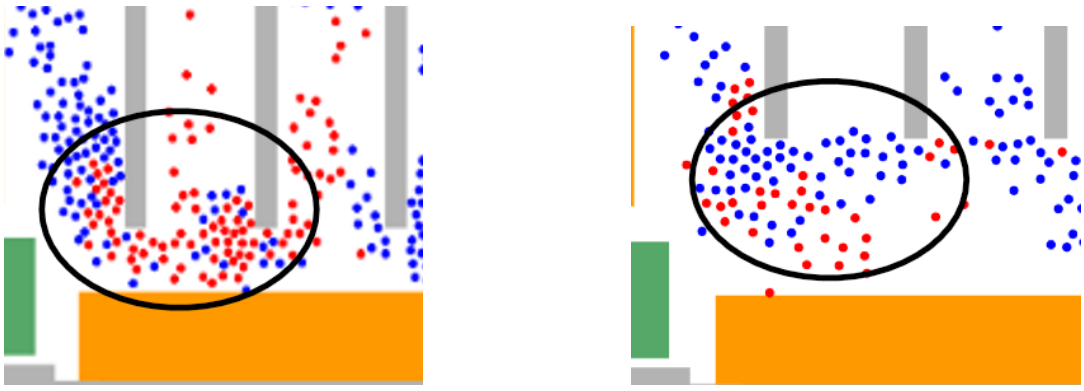


Figure 24: Left (Long continuous obstacles) Right(Shorter obstacles)

4. **Customer Personal Space width**
   Every individual has certain spaces defined around him/her. Personal and Intimate spaces are the closest ones and a contact with an infected person in these spaces, amplifies the chances of a potential infection. We try to tweak this parameter and observe its effect in infection control.
   In order to show the real effects of the 'social distancing' caused by this parameter, we reduce the passenger flux to 90 (30 by each source). See Fig25

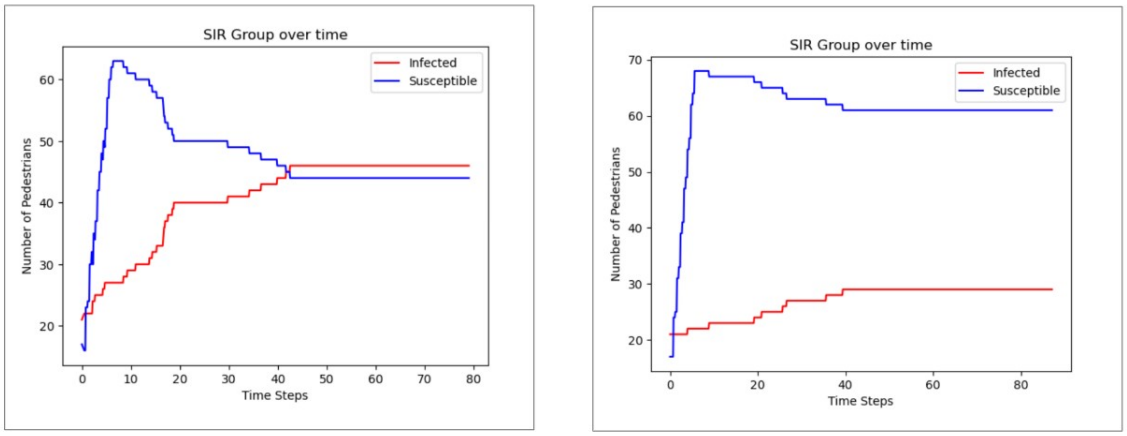


Figure 25: Left (pedPersonalSpaceWidth = 0.2) Right (pedPersonalSpaceWidth = 1.0)

Ignore the initial spike in blue plots (this is because spawn takes time). After it stabilizes, we can see that the scenario with a smaller value of pedPersonalSpaceWidth has a higher number of infected people. Close to 25 people changed their status from 'Susceptible' to 'Infected' (35% of the total uninfected crowd).

Whereas in the second case with pedPersonalSpaceWidth = 1.0, the situation is better with only 13% of the uninfected crowd catching the infection. This also implies that with the given speed of crowd, 90 is rather a huge number of customers inside the supermarket. This number can be further brought down to 55 or 60, so as to fully contain the infection.

SOLUTION - We can have distinguishable improvements in the scenario by limiting the number of people entering our intimate and personal space i.e. in other words by practicing "SOCIAL DISTANCING". In fact, this strategy emerged as one of the most efficient approaches for managing infections and 'flattening the curve' during the recent global onset of the COVID-19 pandemic. People were advised to distance themselves from other individuals by a gap of 2 meters (6 feet).

5. **Speed of the crowd**
   This is an important parameter to discuss. The higher the speed of crowd members, the faster they hit their respective targets and the earlier they leave the scenario. Adding to it, higher speeds leave very less time for the customers to actually interact with each other and contribute to the spread of infection. In our model, speeds are picked from a random distribution for which the min, max, mean and standard deviation is defined. We tweak these parameters by shifting the mean to the max side, increasing the max speed value and thus skewing the distribution so that more customers pick up larger speeds.

Initial Speed Parameters

```
 {
"speedDistributionMean" : 1.34,
"speedDistributionStandardDev" : 0.26,
"minimumSpeed" : 0.5,
"maximumSpeed" : 2.2
}
```

New Speed Parameters

```
 {
"speedDistributionMean" : 2.5,
"speedDistributionStandardDev" : 1.0,
"minimumSpeed" : 0.5,
"maximumSpeed" : 3.5
}
```

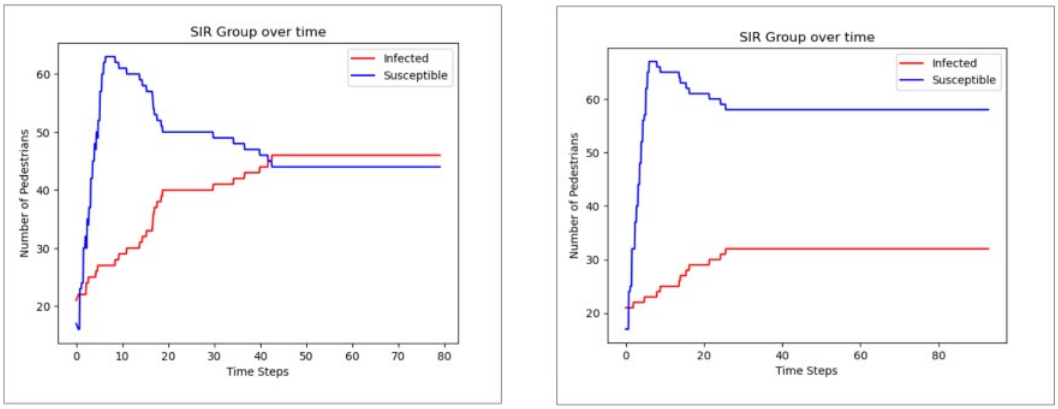


Figure 26: Left (low speeds) Right (high speeds)

We can see that the infection is relatively contained.
SOLUTION - Based on this insight, a practical measure entails the prompt evacuation of individuals from exposed or vulnerable locations. Implementation of designated entry and exit times for individuals frequenting places such as markets or malls could mitigate the formation of groups or crowds, thereby encouraging prioritized task completion and discouraging interpersonal communication during specified periods.

6. **Proper direction to the Crowd**
In the above simulation, customers entering from 'Side Entry 1' were allotted their target as adjacent SHOP1 and for customers entering from 'Side Entry 2', adjacent SHOP2 was allotted. This reduced intermixing between groups entering from different sources and thus reduced the infection rates a bit. This could have worsened in the opposite case where customers from one area would have to cross paths with customers from the other area.

SOLUTION - An effective resolution out of this involves the implementation of clear signage or directional indicators within densely populated areas. The provision of distinct guidance significantly reduces instances of individuals walking aimlessly and minimizes their exposure to potentially vulnerable locations.