Report for exercise 2 from group J

Tasks addressed:	5
Authors:	Wenbin Hu (03779096) Yilin Tang (03755346) **Team Leader** Mei Sun (03755382) Daniel Bamberger (03712890)
Last compiled:	2023–05–15
Source code:	https://github.com/HUWENBIN2024/TUMCrowdModelingGroupJ/tree/main/ex2

The work on tasks was divided in the following way:

Wenbin Hu (03779096)	Task 1	30%
	Task 2	30%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Yilin Tang (03755346) **Team Leader**	Task 1	30%
	Task 2	30%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Mei Sun (03755382)	Task 1	30%
	Task 2	30%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Daniel Bamberger (03712890)	Task 1	10%
	Task 2	10%
	Task 3	0%
	Task 4	0%
	Task 5	0%

Report on task 1, Setting up the Vadere environment

Overview of Vadere [4] In order to use *Vadere*, it was necessary to download Java 11. Then download the software from the Vadere releases website(specifically the "master branch" release instead of the stable one). Now, after installing the Vadere, it is possible to open the Vadere GUI by simply clicking on the file vadere-gui.jar. The interface of Vadere offers a wide variety of options that are very intuitive and makes the setup of any scenario much easier. The visualization is similar to the one of the cellular automaton since it consists of cells as well, with the plus of the elements being able to occupy larger sizes. Moreover, when it visualizes a simulation, it is really convenient to save the result as an output, so the user can check them again at any time. We can run a simulation with one of the built-in examples from Scenarios folder to test the software.

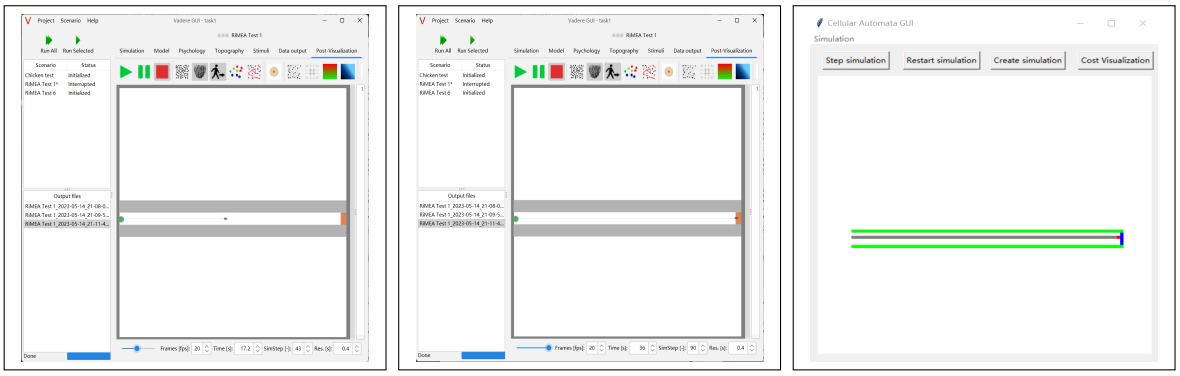
This task requires us to re-simulate RiMEA[5] scenarios 1 (straight line), 6 (corner) and "Chicken Test" through Vadere with *Optimal Steps Model(OSM)* [6, 7].

```
1 # ModelSet
2 "mainModel" : "org.vadere.simulator.models.osm.OptimalStepsModel"
```

RiMEA scenario 1 (straight line) The RiMEA scenario 1 in exercise 1 defined a 2 m wide and 40 m long corridor, with a pedestrian on one side and its target on the other side.

Compared with the results of the cellular automaton created in exercise 1, the behavior of the pedestrian is roughly the same.

- Vadere with OSM: This scenario has been set up with the Vadere as shown in Fig.1(a)Fig.1(b). For proper visualization purposes, both width and height have been set to 40 m, and two obstacles of dimensions 40x19 have been placed to leave a 2 m wide space in the middle. The velocity is obtained by sampling from a distribution that can be altered. The average speed is 1.34m/s, while the maximum and minimum speeds are 2.2m/s and 0.6m/s, respectively. And we could set the size of the time step, we also could set the ratio between simulation time and real-world time. The default time step in Vadere is 0.4s. The pedestrian walk in a straight line. So the pedestrian should reach the target in about 30 seconds. Additionally, the simulation of OSM is more adaptable. In cellular automata, each cell occupies a single grid, and the cell serves as the moving unit. Pedestrians are only able to move horizontally, vertically, or diagonally. In contrast, in OSM, pedestrians are smaller than the grid and can move in any direction.
- Exercise 1: As shown in Fig.1(c), the pedestrians only can set the closest target point as their own target. And we defined the time step as a constant which is 0.25s. The trajectory of Exercise 1 is a straight line.



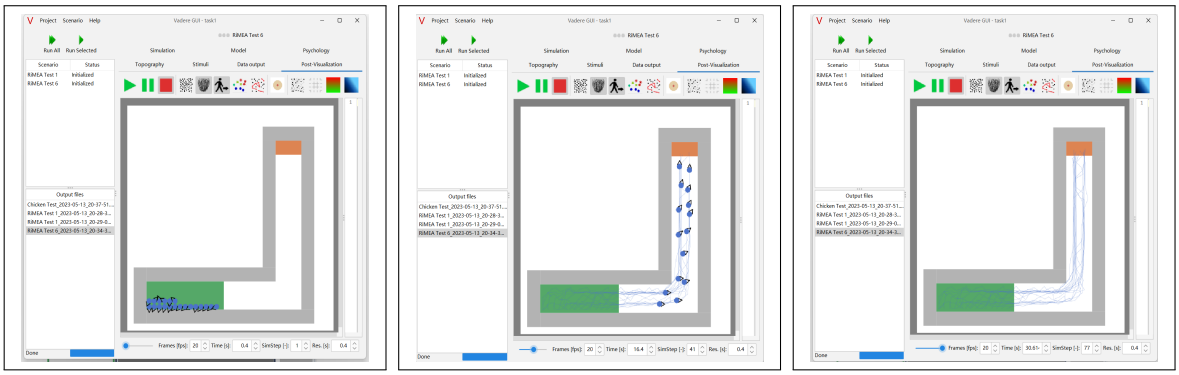
(a) (b) (c)

Figure 1: RiMEA scenario 1 [a] the beginning simulation in Vadere; [b] the simulation result in Vadere; [c] the simulation result in Exercise 1.

RiMEA scenario 6 (corner) The RiMEA scenario 6 in exercise 1 describes 20 pedestrians moving around a left corner, in order to show that they can go around the obstacle without passing through walls.

Compare to the results obtained from the cellular automaton in exercise 1, it appears that the pedestrians display similar behaviors. This is because the OSM was utilized in this simulation, causing the pedestrians always choose the most efficient step toward their goal while taking obstacles into consideration.

- Vadere with OSM: This scenario has been set up with the Vadere as shown in Fig.2. As requirement, both the height and width of the scenario have been set to 12 m. The obstacle has been placed in the top left part of the scenario and has dimensions of 10x10 m. And the corridor has a width of 2 m and both horizontal and vertical part has 12 m of length. The 20 pedestrians are uniformly distributed in the first 6 m of the horizontal part of the corridor (the green part) as demanded in the statement. The Pedestrians go around the corner without passing through walls. They passed the corner in an orderly and sparse way. There are at most two pedestrians walking abreast in the corridor. Furthermore, in this scenario, pedestrians are not only repulsed by obstacles but also repulsed by other pedestrians. Thus the trajectories lines are like two parallel lines in the corridor as shown in Fig.2(c). Because this source has multiple event elements, the simulation time depends on the finish time we set in the" Simulation" file. If we set the finish time long, we can observe the last pedestrian lingering at the target until the finish time.
- Exercise 1: As shown in Fig.4, 20 pedestrians are located randomly at the initial region of pedestrians. A source can be created, and we could set the pedestrians' distribution in the source. Because pedestrians only consider the neighbors' Dijkstra cost. The grids near the inner wall of the corner have the lower Dijkstra cost. There are more pedestrians walking toward the walls. So pedestrians prefer to walk toward the wall. After all pedestrians reach the target point, stop timing.



(a) (b) (c)

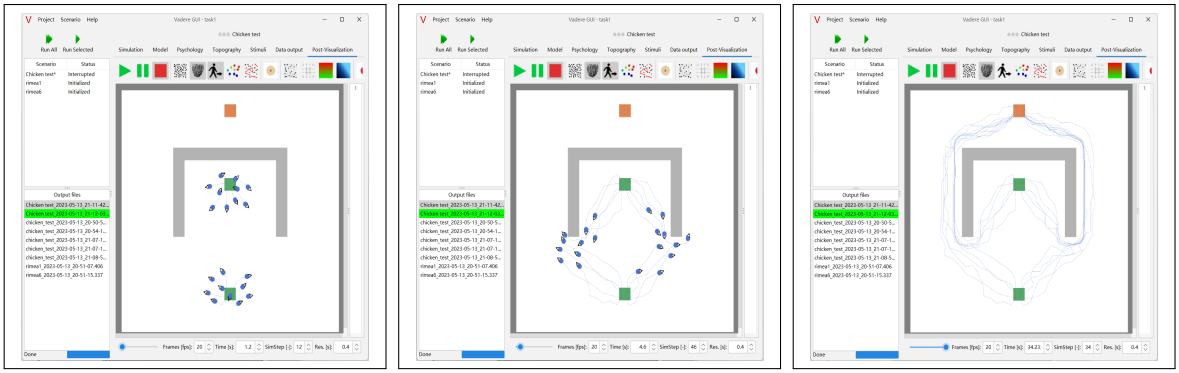
Figure 2: RiMEA scenario 6 [a] the beginning simulation in Vadere with OSM; [b] the middle simulation in Vadere with OSM; [c] the simulation result in Vadere with OSM.

(a) (b) (c)

Figure 3: RiMEA scenario 6 [a] the beginning simulation in Exercise 1; [b] the middle simulation in Exercise 1; [c] the simulation result in Exercise 1.

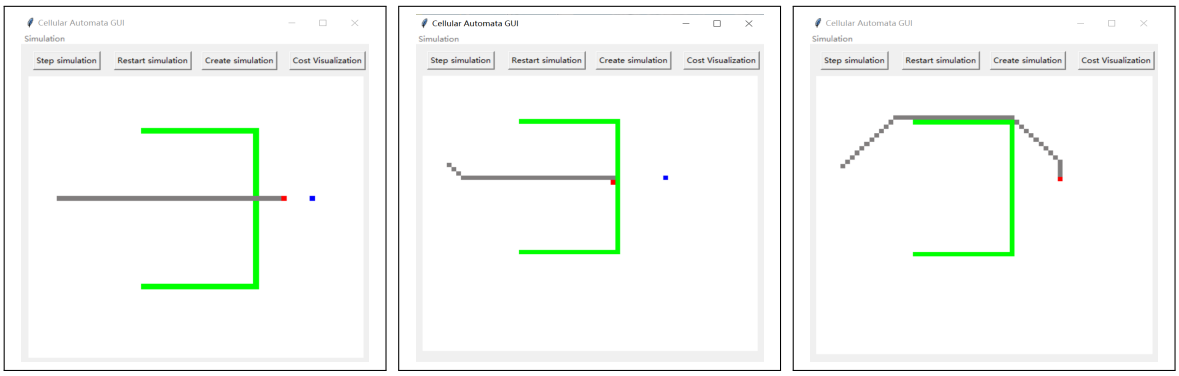
Chicken Test The chicken test consists of a scenario with pedestrians on one side and their target on the other, with a U-shaped obstacle in the middle obstructing the straight path between them. The point of this simulation is to test whether the pedestrians can avoid the obstacles and reach the target or whether they will be trapped at some point and cannot reach any target. Both implementations pass the test.

- Vadere with OSM: This scenario has been set up with the Vadere as shown in Fig.4, the scenario is composed of a U-form obstacle and the pedestrians placed in 2 different places: outside(10 pedestrians) and inside(10 pedestrians) of the U-form obstacles. The target is placed at the same level of the pedestrians, so in this case, the optimal path for the pedestrians to reach the target is a straight line, since there are obstacles, pedestrians cannot follow the straight line path so they have to flip the U form obstacles and then reach the target. As shown in Fig.2(c), all the pedestrians from the different places have reached the target turning around the U-shaped obstacle.
- Exercise 1: As shown in Fig.5, comparing the results with those obtained in the cellular automaton of exercise 1. The pedestrians have roughly the same behavior if the algorithm in exercise 1 is Dijkstra. If the algorithm is the basic two (Without Obstacle Avoidance & With Rudimentary Obstacle Avoidance) in exercise 1, the pedestrians do not perform in the same way, since the basic algorithms are not completely implemented obstacle avoidance, and therefore the pedestrians get stuck in the U-shaped obstacle.



(a) (b) (c)

Figure 4: Chicken Test [a] the beginning simulation in Vadere with OSM; [b] the middle simulation in Vadere with OSM; [c] the simulation result in Vadere with OSM.



(a) (b) (c)

Figure 5: Chicken Test [a] the simulation Without Obstacle Avoidance in Exercise 1; [b] the simulation With Rudimentary Obstacle Avoidance in Exercise 1; [c] the simulation with Dijkstra in Exercise 1.

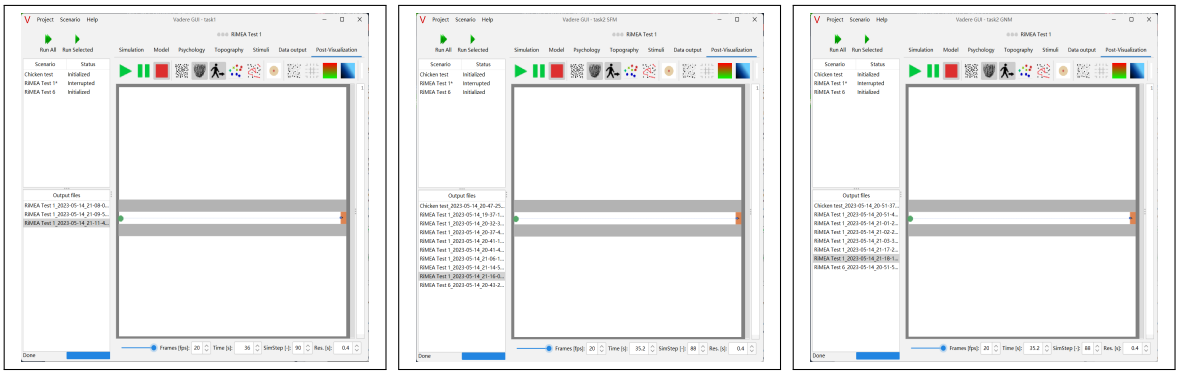
Report on task 2, Simulation of the scenario with a different model

To re-run all the scenarios in task 1 with *Social Force Model(SFM)* [3] and *Gradient Navigation Model(GNM)* [1] , we change the main model to SFM and GNM in Vadere. At first sight, the results, for RiMEA scenario 1 and the "Chicken Test" for single pedestrian don't show obvious differences, but running RiMEA scenario 6 shows that all pedestrians move in closer trajectories and their movement is not as constantly distributed as in Task 1 anymore. The behavior of the pedestrians, suggests that this simulation takes certain social behavior into account. It seems like generally pedestrians have a certain motivation to stay with their group or adjust their paths to the influences of other pedestrians. This explains why simulations with only one pedestrian, e.g. RiMEA Scenario 1 do not exhibit significant changes to the *Optimal Steps Model(OSM)* as there are no other pedestrians to influence individual decisions. To get a clearer idea of the OMS, SFM, and GNM we look into the individual scenarios in more detail.

```
# ModelSet
"mainModel" : "org.vadere.simulator.models.osm.OptimalStepsModel"
"mainModel" : "org.vadere.simulator.models.sfm.SocialForceModel"
"mainModel" : "org.vadere.simulator.models.gnm.GradientNavigationModel"
```

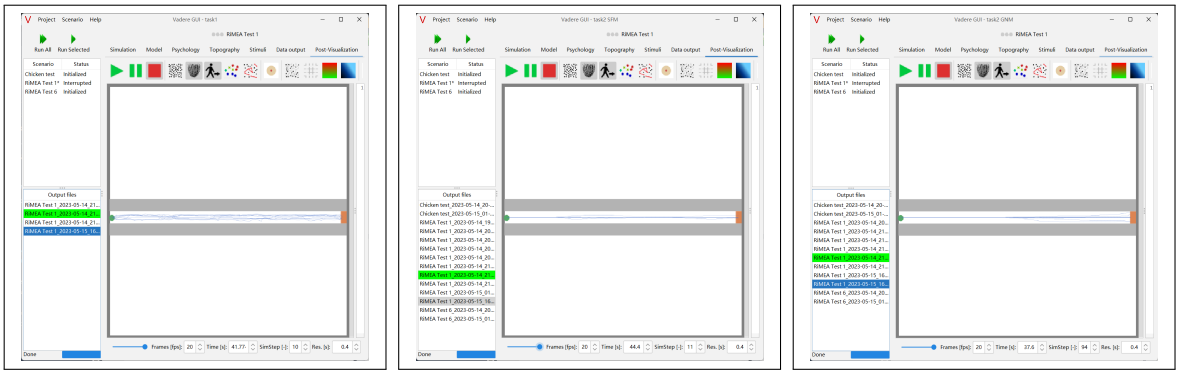
RiMEA Scenario 1 As shown in Fig.6, for single pedestrian no direct difference can be observed between the OSM, SFM, and GNM, the pedestrian follows a strait line and reach the target. Therefore we increased the number of Pedestrians to 10 for information about the behaviors in a simple straight path scenario. Then, some interesting observations can be seen there. As seen in Figure 7 the trajectories of the pedestrians are straighter that in OSM, and also their movement speed seems to be a bit higher on average. These differences can be explained by the social component in the SFM, as pedestrians try to follow their peers, making them choose more direct paths to the goal following their group, and increasing the average speed, as slower pedestrians try to keep up and follow faster individuals.

- OSM: As shown in Fig.7(a), with OSM the 10 pedestrians does not follow a straight line path anymore, instead, as we can see in the Figure . These pedestrians avoid each other so that they cannot follow a straight trajectory. As a result, their trajectories are more chaotic and their distributions more discrete.
- SFM: As shown in Fig.7(b), with SFM the trajectory is closer to a straight line in the 1/4 of the path. The obstacles always evoke a repulsive force pushing 10 pedestrians away in this model. Thus, pedestrians prefer to walk in the middle of the corridor. Therefore, pedestrians will waste some time waiting in line to walk in the middle, resulting in a longer travel time for SFM than for OSM (44.4 seconds with SFM and 41.77 seconds with OSM).
- GNM: As shown in Fig.7(c), with GNM, he trajectory of 10 pedestrians is smoother and tends to be more in a straight line. The GNM is not a force-based model, and pedestrians don't be repulsed by obstacles. It uses a superposition of gradients of distance functions to directly change the direction of the velocity vector. So, in this simulation, the pedestrian doesn't keep distance from obstacles and walks straight. Thus the pedestrians reach the target a little faster than OSM (37.6 seconds with GNM and 41.77 seconds with OSM).



(a) (b) (c)

Figure 6: RiMEA scenario 1 for single pedestrian [a] the simulation result with OMS; [b] the simulation result with SFM; [c] the simulation result in GNM.

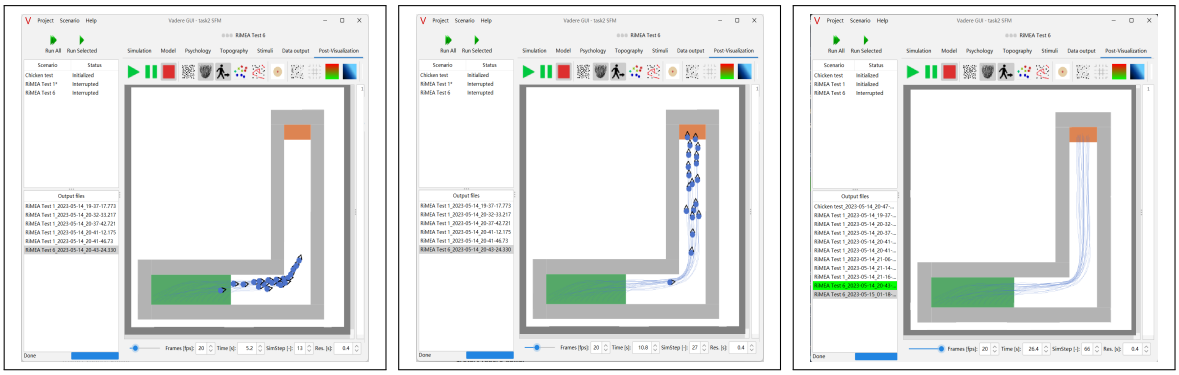


(a) (b) (c)

Figure 7: RiMEA scenario 1 for 10 pedestrians [a] the simulation result with OMS; [b] the simulation result with SFM; [c] the simulation result in GNM.

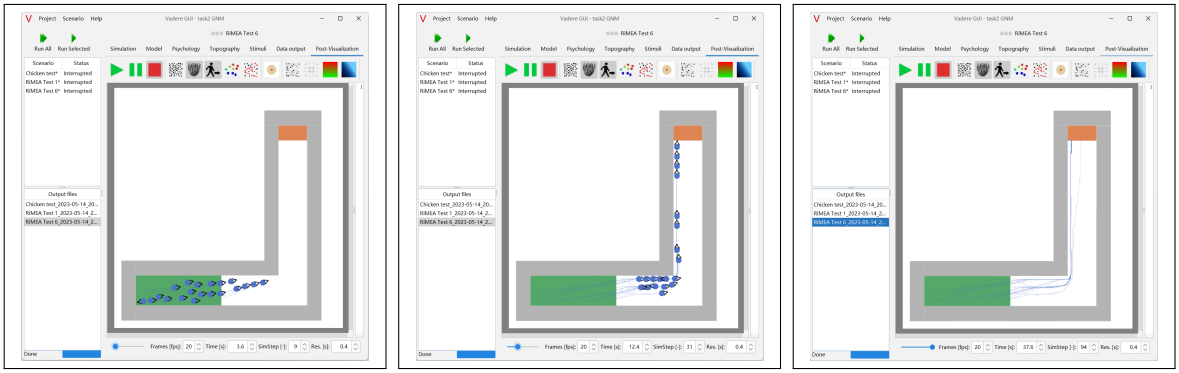
RiMEA Scenario 6

- SFM: As shown in Fig.8, comparing the behaviors of the pedestrians using the SFM and the OSM, the SFM yields more natural and orderly movements. In SFM, all pedestrians tend to follow the shortest and optimal path to reach the target, resulting in smoother trajectories compared to the hard angles and squared-like stretches in OSM. The crowd movement is also more fluid in SFM as there is less excessive giving way to others, unlike in OSM where pedestrians tend to stop unnecessarily. In Fig.8(c), the overall time for all pedestrians to reach the target is 26.4 seconds with SFM, which is faster than the 30.61 seconds with OSM.
- GNM: As shown in Fig.9, comparing the behaviors of pedestrians using GNM and OSM in this scenario, it was observed that all but two pedestrians followed the same uniform path as the first pedestrian who reached the target, without passing through walls. This suggested that the pedestrians were organized and had already known the path they should take. In Fig.9(c), the result of GNM showed that all pedestrians walked toward the wall, passing the corner one by one. Moreover, when a pedestrian encountered an obstacle in the form of another pedestrian, their behavior was to reduce their speed and wait until they could continue, instead of choosing an alternate path, which was different from the behavior of the pedestrians in the simulation using OSM. This behavior caused the travel time to increase to 37.6 seconds.



(a) (b) (c)

Figure 8: RiMEA scenario 6 with SFM [a] the beginning simulation in Vadere with OSM; [b] the middle simulation in Vadere with OSM; [c] the simulation result in Vadere with OSM.

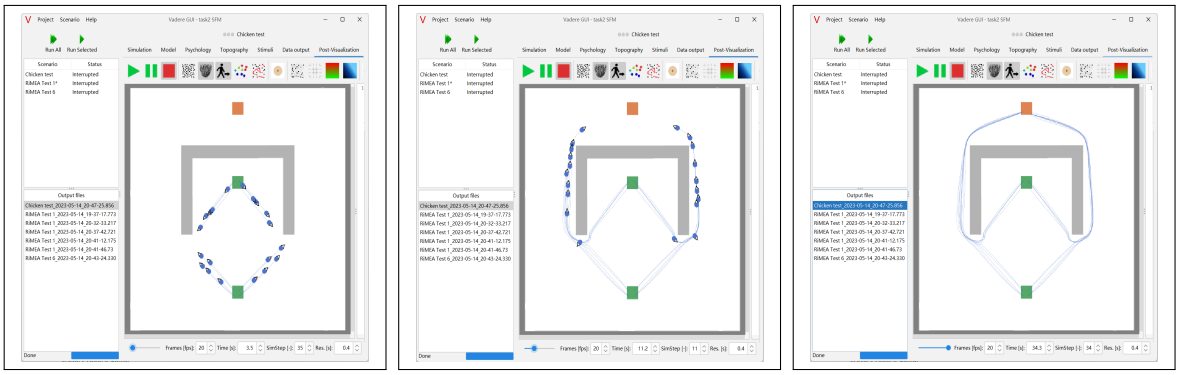


(a) (b) (c)

Figure 9: RiMEA scenario 6 with GNM [a] the beginning simulation; [b] the middle simulation; [c] the simulation result.

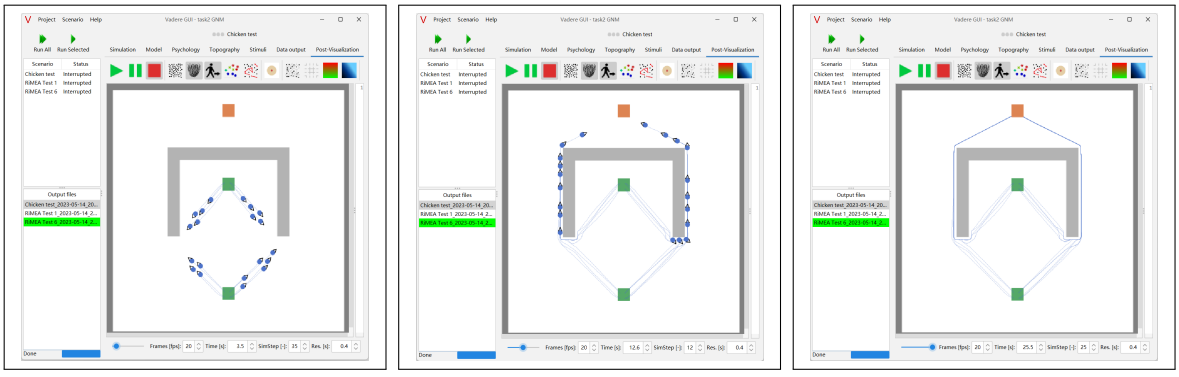
Chicken Test

- SFM: As shown in Fig.10, comparing the behaviors of pedestrians using SFM and OSM for the chicken test scenario, both models allowed the pedestrians to avoid obstacles and reach the target. However, the behavior exhibited using SFM appeared to be more direct, ordered, and natural than that of OSM. With SFM, it seemed like the pedestrians knew from the beginning where to go to reach the target, resulting in smoother and more direct paths as shown in Fig.10(c). On the other hand, although all of the pedestrians in OSM could avoid obstacles and reach the target, some of them collided with each other and obstacles along the way, resulting in less smooth paths with larger changes in direction. The travel time is very close, with SFM is 34.3 seconds, and OSM is 34.23 seconds.
- GNM: As shown in Fig.11, in the chicken test, all pedestrians were able to turn around the U-shaped obstacle, by taking uniform paths as well. The GNM takes a very directed trajectory to the target, it looks close to the shortest path that Dijkstra's algorithm could find. It is possible to observe this in Fig.4(c) and Fig.11(c). Their behaviors are similar to RiMEA scenario 6, but this time different yet symmetrical paths were taken, depending on the starting point of each pedestrian inside the source (either slightly upper or lower). There were some exceptions as well, some pedestrians chose to avoid the obstructing pedestrians rather than waiting. This time, however, the travel time was reduced by around 6 seconds, most probably because in the simulation using OSM, the scattering of so many pedestrians in such a narrow space for taking other paths caused a blockage, something that does not happen with GNM. So it is faster than both OSM and SFM, which OSM is 34.23 seconds, SFM is 34.3 seconds, and GNM is 25.5 seconds.



(a) (b) (c)

Figure 10: Chicken Test with SFM [a] the beginning simulation; [b] the middle simulation; [c] the simulation result.



(a) (b) (c)

Figure 11: Chicken Test with GNM [a] the beginning simulation; [b] the middle simulation; [c] the simulation result.

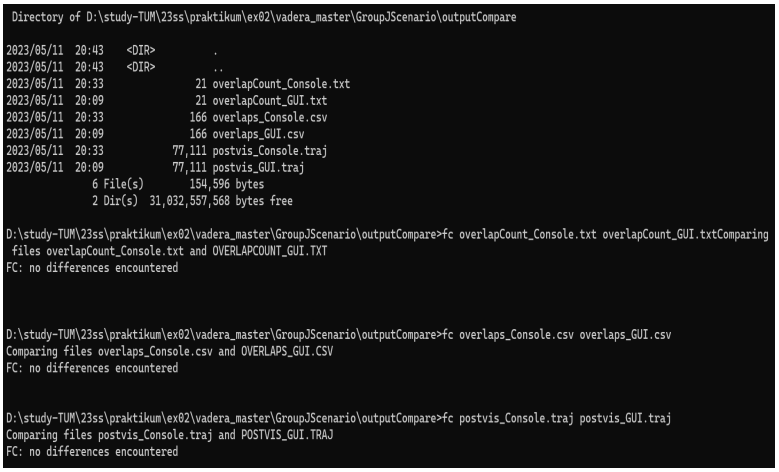
Report on task 3, Using the console interface from Vadere

Running Vadere through the console with command line

After running Vadere from the console with the required command line shown in the following.

```
# Simulation
java –jar vadere–console.jar scenario–run
––scenario–file "/path/to/the/file/scenariofilename.scenario"
––output–dir="/path/to/output/folders"
```

We noticed that the output files are, as expected, the same as the output files from GUI. The compared result could be done by using **fc** command in Windows. As shown in Fig.12.



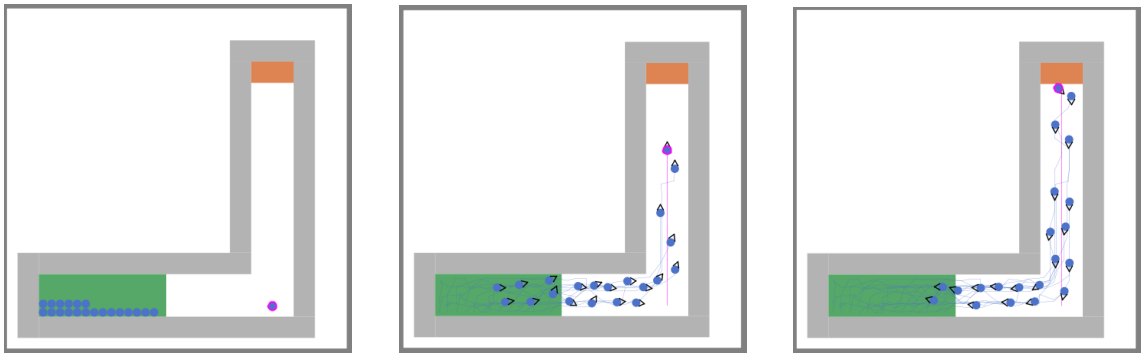
(a)

Figure 12: Comparison result of output files between Console and GUI.

Adding pedestrians programmatically

In this part, we create a Python script **add_pedestrians.py** to implement the function of adding pedestrians. This script first loads the scenario file and changes the scenario by adding a pedestrian, and finally saves the modified file in JSON format. Thus, there are 3 functions in our script:

- **load_scenario(path)**: This function allows us to access all the information of the input JSON scenario file. Then stores the data in a dictionary called "attributes". This dictionary actually has the same structure which is shown in the JSON scenario file under the list *dynamicElements*. Finally, return it.
- **add_pedestrian(attributes)**: This is the major function, it takes the "attributes" obtained from the *load_scenario*(*path*) as the parameter and adds a pedestrian with the same attributes as the other pedestrians at position (13, 2.5). As shown in Fig.13(a) and returns the updated scenario with the added pedestrian.
- **save_scenario(path, attributes)**: This function saves the updated scenario to the specified path so that it can be loaded into Vadere.



(a) (b) (c)

Figure 13: [a] Initial scenario. t=0; [b] Middle moment. t=8; [c] First one hits the target.t=11.6

We perform the new scenario both in GUI and Console, there is no difference by comparing the output files. By analyzing the output, we found that the new pedestrian takes 11.6 seconds to reach the destination Fig.13(c). Obviously, he is the fastest one because of the nearest distance. The second fastest which from the source takes 12 seconds. The last one takes 30.4 s seconds(with OSM).

Report on task 4, Integrating a new model

Introduction to SIR [2] Model

Notations: S: Susceptible, I: Infectious, and R: Recovered.

Motivation: The SIR model is based on the assumption that the population is well-mixed, meaning that any individual is equally likely to come into contact with any other individual. It also assumes that the disease is transmitted through direct contact between infectious and susceptible individuals.

Model: The model is represented by a set of differential equations that describe the change in the number of individuals in each state over time. The equations are as follows:

$$\frac{dS}{dt} = -\beta SI$$

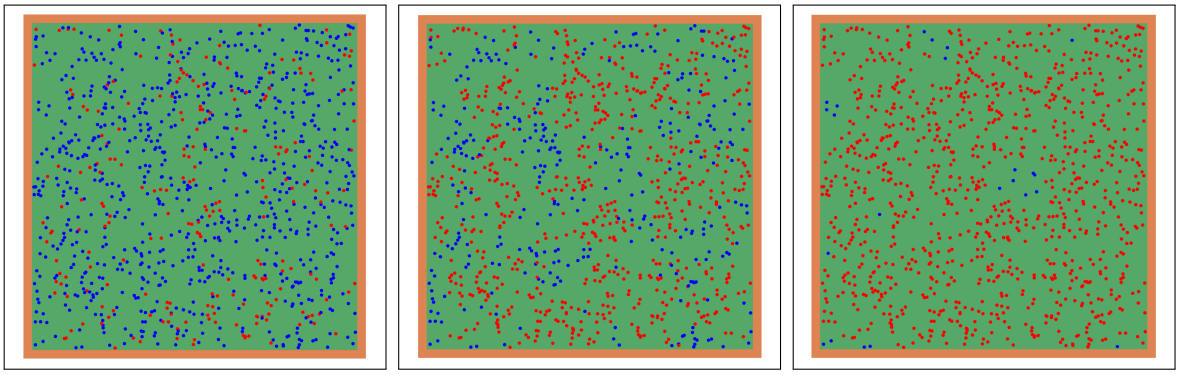
$$\frac{dI}{dt} = \beta SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

, where β is the transmission rate, γ is the recovery rate, $S(t)$ is the number of susceptible individuals at time t, $I(t)$ is the number of infectious individuals at time t, and $R(t)$ is the number of recovered individuals at time t.

Analysis: The first equation represents the rate of change of susceptible individuals over time, which decreases as individuals become infected (the $-\beta SI$ term). The second equation represents the rate of change of infectious individuals over time, which increases as individuals become infected and decrease as they recover (the βSI term and the $-\gamma I$ term). The third equation represents the rate of change of recovered individuals over time, which increases as individuals recover from the disease (the γI term).

Conclusion: By simulating the SIR model, researchers and public health officials can estimate the potential impact of interventions such as vaccination, social distancing, and quarantine on the spread of infectious diseases within a population.



(a) (b) (c)

Figure 14: [a] shows the initialization of the scenario; [b] shows the half of population is infected; [c] shows the end of the simulation.

Subtask 1: Integrate the SIR model in Vadere As instructed by the exercise sheet and the Readme file, we place files compressed in SIRGroupModel.zip to the correct places in Vadare source code to functionalize the SIR model. Those places/directories are indicated by the first line in each file. Specifically:

1) SIRGroup.java, SIRGroupModel.java, and SIRType.java under the folder sir/ should be placed in directory: VadereSimulator/src/org/vadere/simulator/models/groups/sir

2) AbstractGroupModel.java should be at VadereSimulator/src/org/vadere/simulator/models/groups

3) AttributesSIRG.java should be at VadereState/src/org/vadere/state/attributes/models

4) FootStepGroupIDProcessor.java should be at VadereSimulator/src/org/vadere/simulator/projects/dataprocessing/processor

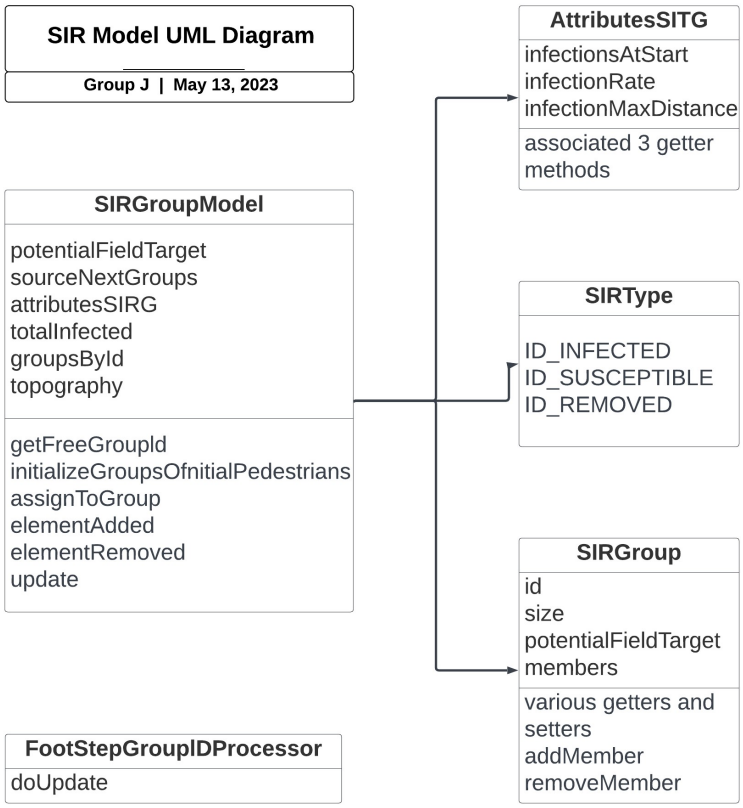


Figure 15: A UML diagram that shows the architecture of SIR implementation.

The detailed functionalities are shown below:

1) The class "AttributesSIRG" is used to set general parameters related to the infection behavior in a simulation. These parameters include the number of initially infected pedestrians ("infectionsAtStart"), the probability of a susceptible pedestrian becoming infected when in contact with an infected pedestrian ("infectionRate"), and the range within which a susceptible pedestrian can become infected if an infected pedestrian is within that range ("infectionMaxDistance").

2) The "SIRGroup" class is responsible for maintaining an active count and membership for different pedestrian groups, namely Infected, Susceptible, and Recovered. Each group is identified by a unique ID and contains a list of members. The "SIRType" enum is used to conveniently store the possible pedestrian states.

3) The "SIRGroupModel" class is the main class responsible for dividing pedestrians into groups, updating them, and performing infections and recoveries. It contains a LinkedHashMap to store the present groups, an instance of "AttributesSIRG" to hold the simulation parameters, and a counter to keep track of the overall number of currently infected pedestrians. It also includes methods for creating groups, initializing groups with the initial pedestrians, and assigning pedestrians to groups. The "update" method implements the model core by checking for infected pedestrians within the maximum infective range and infecting susceptible pedestrians based on a probability coin toss.

4) The "FootStepGroupIDProcessor" class is an output processor that maintains the infection status in the resulting file for later analysis and plotting. Its "doUpdate" method writes a new output row at each

time step composed of the current time step, the pedestrian ID, and its group ID. It is essential to use this processor as the desired output processor to ensure that the infection status information reaches the post-simulation analysis.

Subtask 2: Simulation Setup As instructed by the exercise sheet, we renew the processor FootStepGroupIDProcessor. We configure DataOutput by adding SIRInformation.csv to the Files session and linking it to FootStepGroupIDProcessor. In addition, under Model session, we add a submodel: org.vadere.simulator.models.groups.sir. SIRGroupModel. By default, we are keeping using OptimalStepsModel mode for our simulation. Once we obtain the output files from Vadere, we use Dash/Ploty app for sketching the result and analyze the SIR information by running app.py.

Subtask 3: Color Correction for Different Groups Visualization is significant and crucial to crowd modeling and simulation. So as to achieve this goal, to distinguish different groups of pedestrians, we set them to different colors. In this way, we can observe the crowd/epidemic flow easily. The colors for S, I, and R groups are blue, red, and green respectively. We added several lines in SimulationModel.java for implementation. Details are in our provided code pushed on GitHub. Thankfully, since all the simulation setup and visualization are ready, we can continue our study and further improvement.

Subtask 4: Improvement of the Infection Functionality The original update function is the method update() in SIRGroupModel class. Our improvement mainly lies in 2 aspects:

1) The original update function does a 2-nested for loop to find neighbor-infected people for each person, and then the function use the distance result to update infected people. This implementation has drawbacks both in the program running efficiency and infection broadcast efficacy. As for efficiency, it is obviously low by the nested for loop. For efficacy problem, it is very likely that a person who should not be infected is infected by a just updated infected person. Instead, inspired by reverse thinking, we do a loop for every infected people to infect their surrounding susceptible people.

2) It is not efficient to traverse all the pedestrians to compute the distances for checking neighborhood. We can use the getObjects() method in LinkedCellsGrid class to find neighbors, which is provided by the Vadere source code. We just simply add one line of code:

```
List<Pedestrian> nb = c.getCellsElements().getObjects(position, infectionRadius);
```

Subtask 5: Simulation of Two Scenarios We have already corrected the color of SIR groups for visualization and infection functionality. Now, we conduct two experiments of simulation for a static scenario and a dynamic scenario. Here are some analyze:

1) **Static Scenario** The experiment involves a fixed area of 50 meters by 50 meters, where a source generates 1000 pedestrians at random places. The target is located directly above the source but is not absorbing. Parameters are:

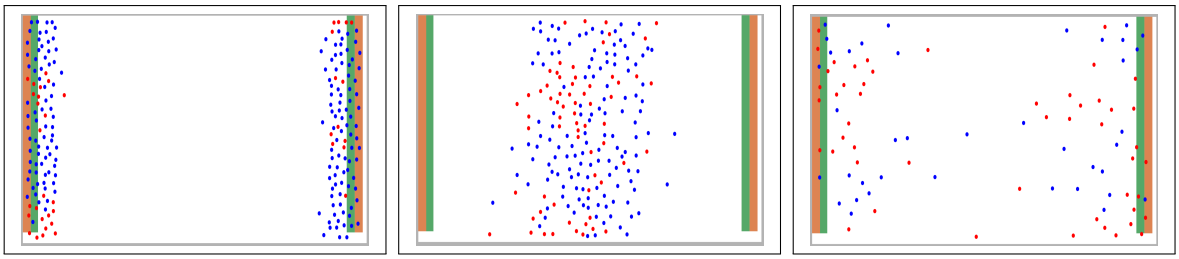
```
— infectionRate: 0.07 or 0.14
— infectionAtStart: 10
— infectionMaxDistance: 1.5
— spawnAtRandomPositions: True
— useFreeSpaceOnly: False
```

The simulation of the scene is shown in Fig.14. When infectionRate = 0.07, half of the population is infected in 5.0s; when infectionRate = 0.14, half of the population is infected in 2.6s. The visualization of infection results are shown in Fig.17(a) and Fig.17(b) respectively.

2) **Dynamic Scenario** We construct a corridor scenario of $40m \times 20m$, where one group of 100 susceptible people moves from left to right, and another group of the same number of people moves from right to left. Our parameters are selected as:

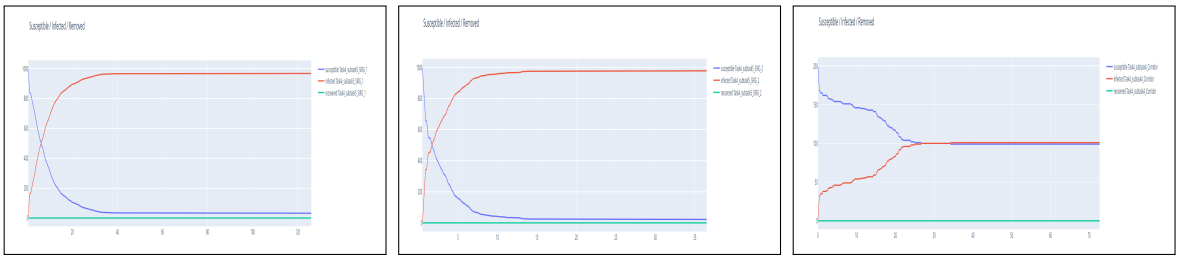
```
— infectionRate: 0.04
— infectionAtStart: 10
— infectionMaxDistance: 1
— spawnAtRandomPositions: True
— useFreeSpaceOnly: False
```

The scene is shown in Fig.16. UseFreeSpaceOnly is set as true to create pedestrians at running time. 98 pedestrians are infected during the process. The visualization of the infection result is shown in Fig.17(c).



(a) (b) (c)

Figure 16: [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.



(a) (b) (c)

Figure 17: [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.

Subtask 6: Decouple the Infection Rate and Time Step The instruction in the exercise sheet point out that it is not natural and proper to associate the infection rate and time step. The time step varies in different computers and even in the same computer with different using times. This will cause instability for crowd/infection modeling, which is not bearable for researchers and engineers. To tackle this problem, we associate the infection rate with real-world time. To be specific, the infection rate α means the original α times per second. If a α is smaller than 1, the program will accumulate the update; if greater than 1, the program will do a while loop to complete a proper number of update steps. Details are described in our pseudocode [1].

Algorithm 1 Decouple the Infection Rate and Time Step

$t \leftarrow t + \Delta t$
if $t \geq 1$ **then**
  while $t \geq \frac{1}{\alpha}$ **do**
    update()
    $t \leftarrow t - \frac{1}{\alpha}$
  end
end

Subtask 7: Possible Extensions Though we already have a comprehensive infection modeling system, it is still far from the real-world situation. There exists lots of features we need to add to the program. Here are some examples:

1) Immune state: Genetic factors can make some individuals more resistant to diseases than others. This state would be assigned at the start of the simulation, with a certain probability rate. Once in the immune state, the pedestrian would be incapable of becoming susceptible, infected, or recovered.

2) Pedestrian age: In real life, people of different ages have varying levels of susceptibility to illnesses. Implementing an age distribution in the model could act as a weight on the infection, recovery, and death rates. Additionally, an aging mechanism could be added so that pedestrians get older over time, dynamically altering their infection, recovery, and death rates.

3) Vaccinated status: Vaccinated individuals are likely to recover faster and have a lower risk of being infected or dying. However, the effectiveness of a vaccine may vary over time.

Report on task 5, Analysis and visualization of results

Implement Recovered State In previous tasks, we implement a SIR model to simulate infection among susceptible, infectious, and not infected people without a recovered state. Naturally, an infected person will turn to a recovered state. Thus, in this task, we implement recovered states for the program. We add IDRECOVERED to SIRType class and add recoveryRate to AttributeSIRG class. We utilize a system-generated random number to control the recovery rate, which is described in algorithm 2.

Algorithm 2 Recovered State

```
for p ← infectedGroup do
    r ← random(min=0, max=1)
    if r ≤ recoveredRate then
        p.state ← 'recovered'
    end
end
```

To visualize the changes for recovered pedestrians in the plot, here we add an additional " if " condition in utils.py located in the SIRVisualization file. If a pedestrian gets recovered from the infected state, we set the current state to recover, then for each future time step, increases the number of recovered and decrease the infected. Once a pedestrian is recovered, it can not change its state, so we break.

Simulation with Recovered State Now that we have implemented the recovered state for SIR model, we can conduct some experience on the efficacy of our implementation.

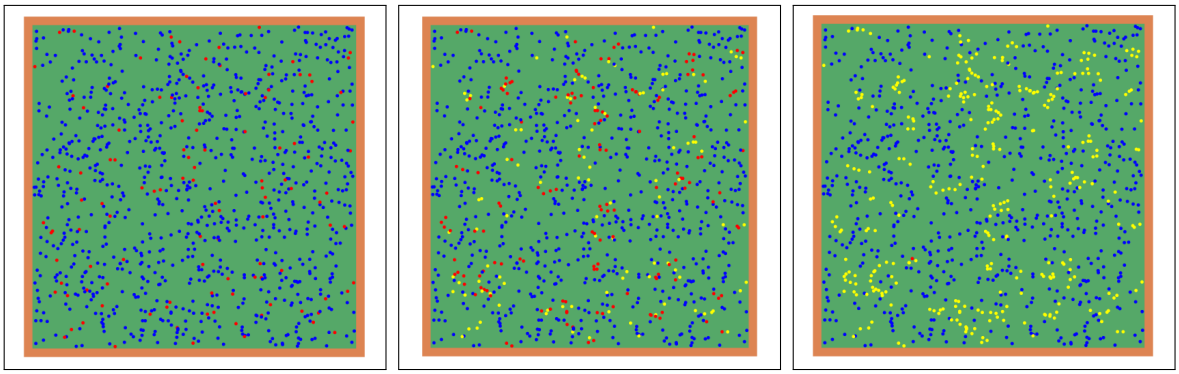
1. A Basic Test For the test I, we need to construct a large scenario with 1000 random-placed pedestrians. We start with 10 infective and 990 susceptible as instructed. The scene size is 50×50. The parameters we use are shown below:

```
— infectionRate: 0.1
— recoveredRate: 0.1
— infectionAtStart: 10
— infectionMaxDistance: 1
— spawnAtRandomPositions: True
— useFreeSpaceOnly: False
```

The setup simulation scene is shown in Fig.18(a), and after it turn to Fig.18(b) and Fig.18(c). The visualization result of the number of infective/susceptible/recovered people is shown in Fig.18(a). As we can see, compared to the result in Task 4, the infection speed is slower, since an infected person can turn into a recovered state to block the spread of the pandemic. It's worth noting that it seems the number of infected people will not increase at certain status during the simulation. It doesn't mean that no pedestrian will get infected anymore. The truth is that the number of newly infected people is equal to the number of newly recovered people.

2. Contrast Experiments To show the efficacy of our implementation, we have conducted contrast experiments to comprehensively show our results. We measure the result with the variation of infectionRate and recoveryRate.

1) Experimental Setting: infectionRate = 0.1, recoveryRate = 0.1. The simulation is shown in Fig.18. The visualization of the result can be shown in Fig.19. This setting provides a baseline for the following settings, to investigate the infectionRate/recoveryRate.



(a) (b) (c)

Figure 18: Shows the process of the simulation of infectionRate=0.1 and recoveredRate=0.1 [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.

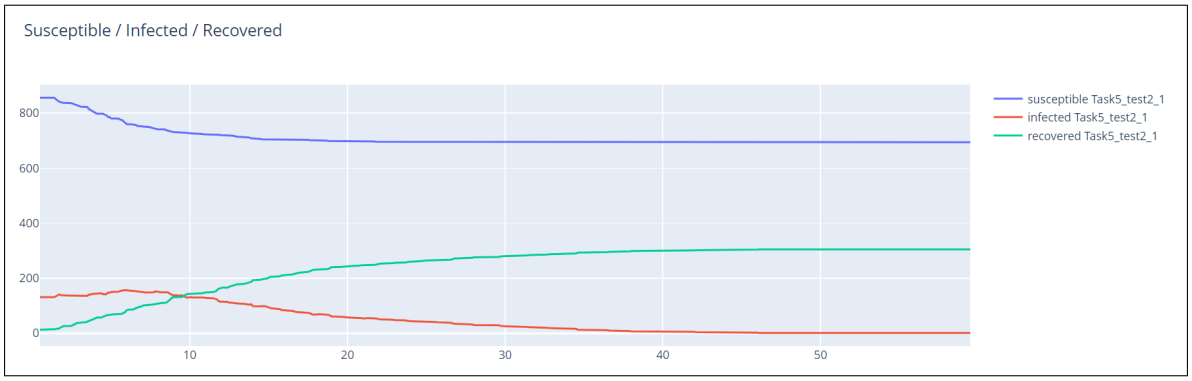
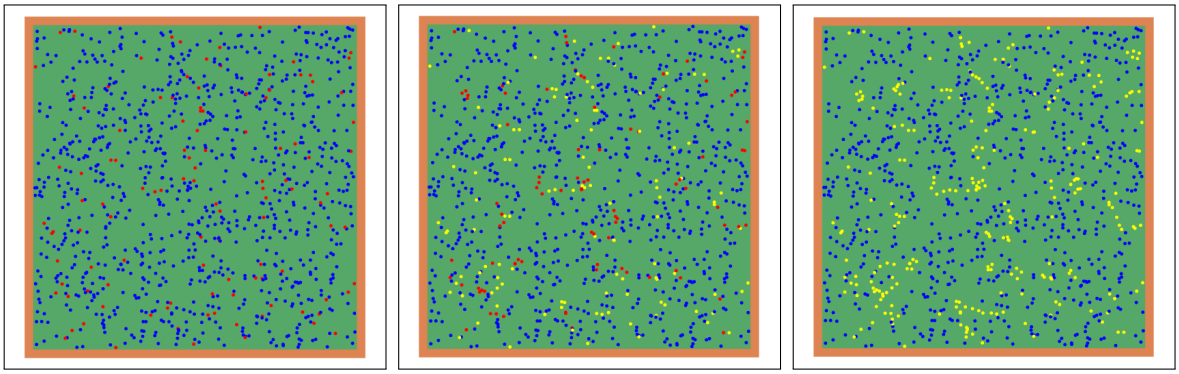


Figure 19: Shows the visualization of the result of infectionRate=0.1 and recoveredRate=0.1

2) Experimental Setting: infectionRate = 0.1, recoveryRate = 0.2. The simulation is shown in Fig.20. The visualization of the result can be shown in Fig.21. Compared to the first setting, recoveryRate is larger. As we can see in Fig.21, less people are infected, and the increase speed of the number of infected people is slower. We can conclude that increasing recoveryRate can suppress the infection, and make less people infected in the end.



(a) (b) (c)

Figure 20: Shows the process of the simulation of infectionRate=0.1 and recoveredRate=0.2 [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.

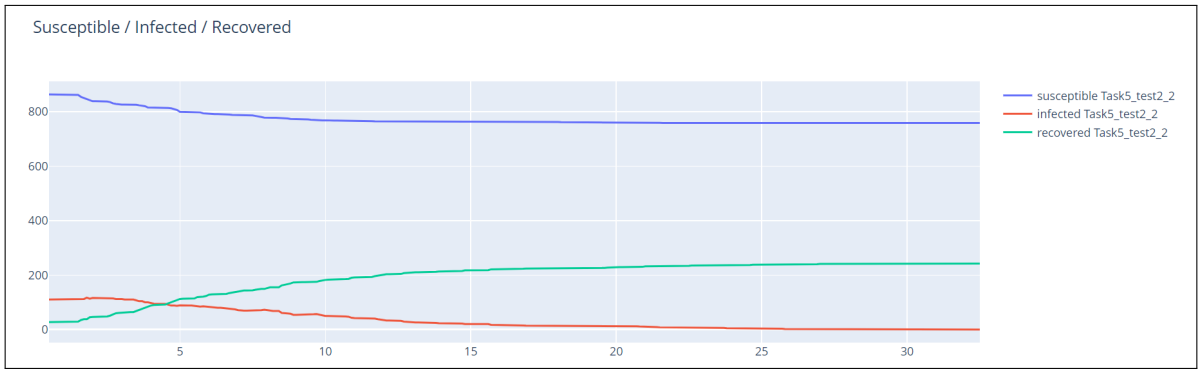
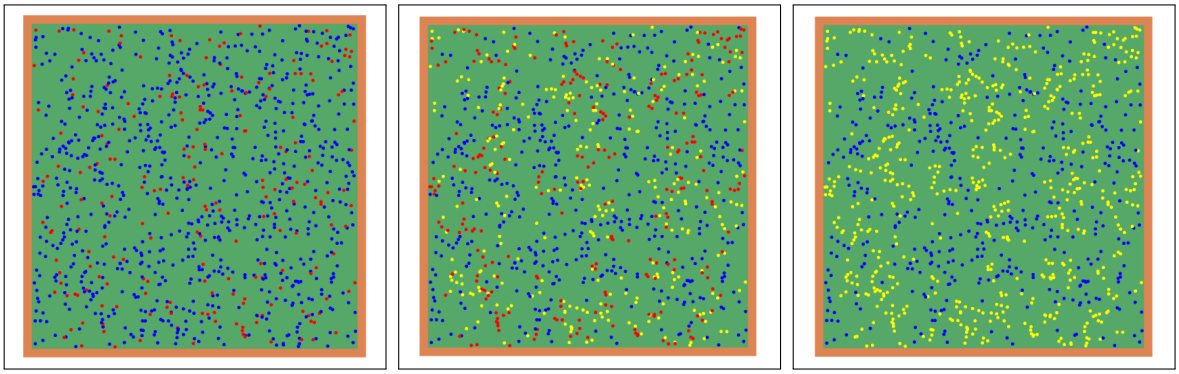


Figure 21: Shows the visualization of the result of infectionRate=0.1 and recoveredRate=0.2

3) Experimental Setting: infectionRate = 0.2, recoveryRate = 0.1. The simulation is shown in Fig.22. The visualization of the result can be shown in Fig.23. As we can see in Fig.22, more people are infected, and the increase speed of the number of infected people is faster. We can conclude that increasing recoveryRate can exacerbate the infection, and make more people infected in the end.



(a) (b) (c)

Figure 22: Shows the process of the simulation of infectionRate=0.2 and recoveredRate=0.1 [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.

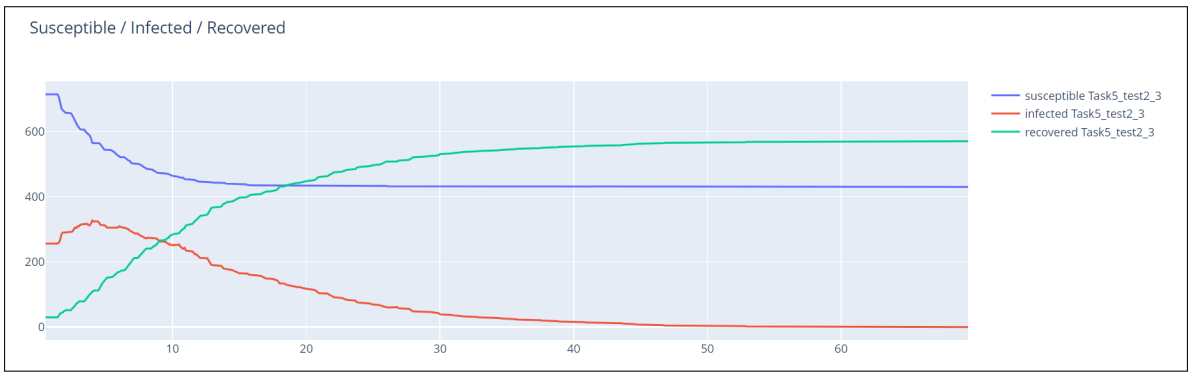
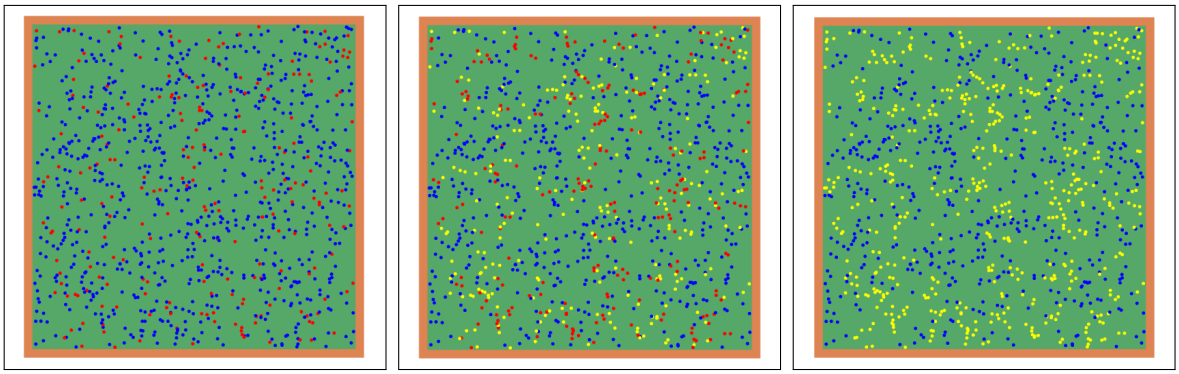


Figure 23: Shows the visualization of the result of infectionRate=0.2 and recoveredRate=0.1.

4) Experimental Setting: infectionRate = 0.2, recoveryRate = 0.2. The simulation is shown in Fig.26. The visualization of the result can be shown in Fig.25. The infection is more serious compared to the one in setting 2; the infection is less serious compared to the one in setting 3. This setting is used to further confirm our conclusion in setting 2, and 3.



(a) (b) (c)

Figure 24: Shows the process of the simulation of infectionRate=0.2 and recoveredRate=0.2 [a] shows the initialization of the scenario; [b] shows in the middle of the simulation; [c] shows the end of the simulation.

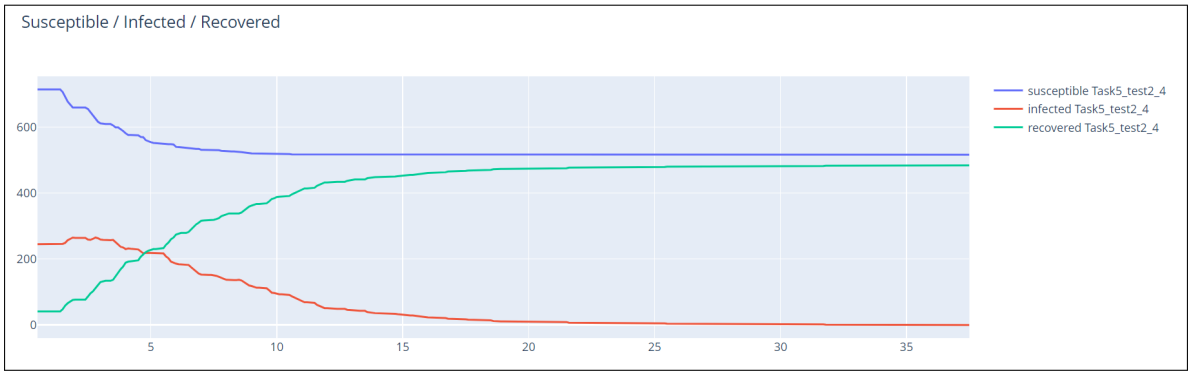
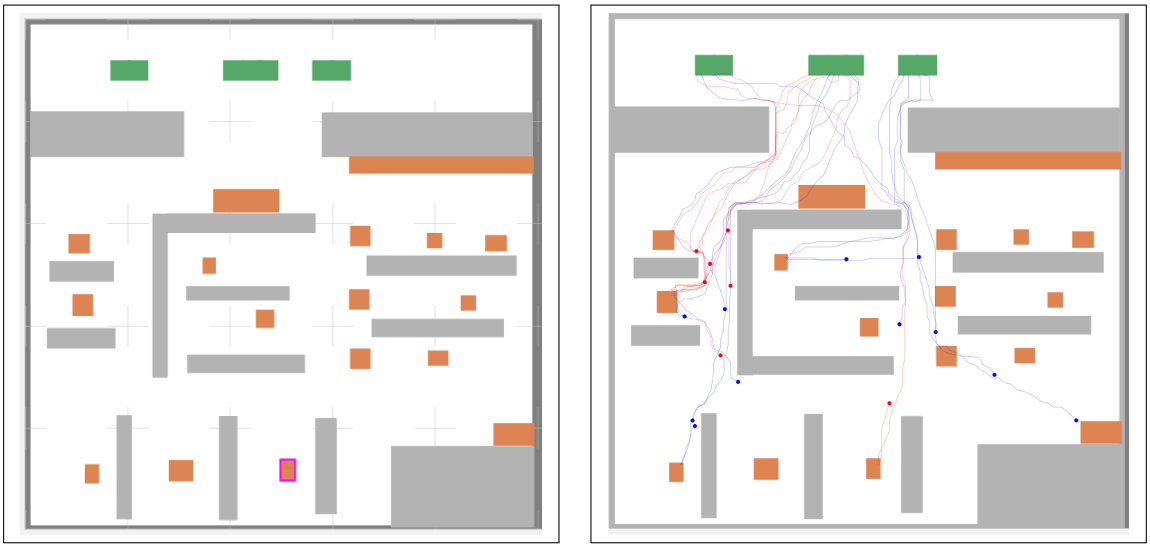


Figure 25: Shows the visualization of the result of infectionRate=0.2 and recoveredRate=0.2

3. Supermarket A final test was conducted to assess the model's performance in a real-world scenario of infection: a supermarket. Given that supermarkets are potential sites for disease transmission, it is essential to take precautionary measures to prevent large gatherings and ensure adequate interpersonal space. We construct a scenario of the supermarket, which can be seen in Fig. 26(a). A possible simulation is shown in Fig. 26(b).

```
— infectionRate: 0.1
— recoveredRate: 0.0
— infectionAtStart: 1
— infectionMaxDistance: 1
```



(a) (b)

Figure 26: 26(a) is our scenario of our supermarket; 26(b) is a possible simulation.

The following Figure27 is the plot for supermarket SIR simulation result.

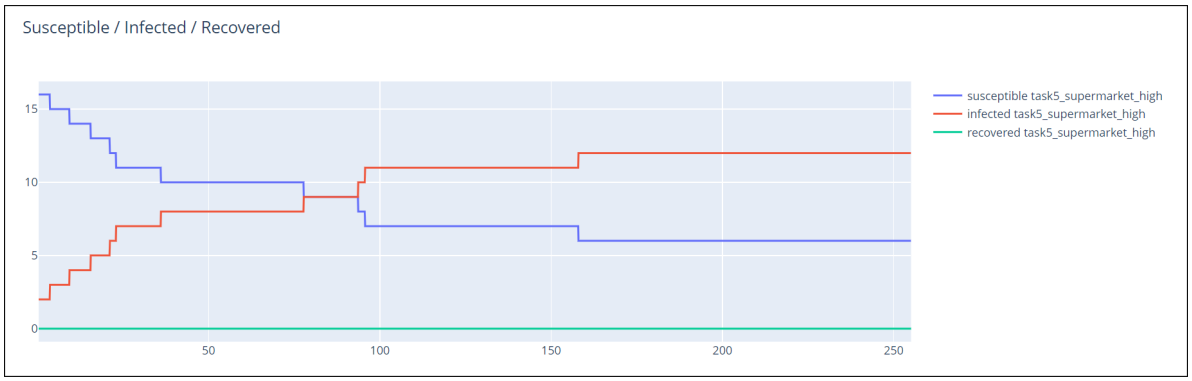


Figure 27: Shows the visualization of the result for supermarket simulation

It is evident that areas with relatively narrow spaces, such as the left side of the picture, can contribute significantly to the transmission of the virus. For instance, the passage of just six customers through this aisle led to their infection. Therefore, it is a wise and necessary decision to regulate the number of people entering places like supermarkets, shopping malls, and other public areas during the epidemic period to minimize the risk of transmission.

References

[1] Felix Dietrich and Gerta Köster. Gradient navigation model for pedestrian dynamics. *Physical Review E*, 89(6):062801, 2014.

[2] Dirk Helbing, Illés Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[3] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[4] Benedikt Kleinmeier, Benedikt Zönnchen, Marion Gödel, and Gerta Köster. Vadere: An open-source simulation framework to promote interdisciplinary understanding. *Collective Dynamics*, 4, 2019.

[5] G Rimea. Richtlinie für mikroskopische entfluchtungsanalysen [guidelinefor microscopic evacuation analysis] version 3.0, 2016.

[6] Michael J Seitz and Gerta Köster. Natural discretization of pedestrian movement in continuous space. *Physical Review E*, 86(4):046108, 2012.

[7] Isabella Von Sivers and Gerta Köster. Dynamic stride length adaptation according to utility and personal space. *Transportation Research Part B: Methodological*, 74:104–117, 2015.