Report for exercise 2 from group I

Tasks addressed: 5
Authors: Yizhi Liu (03779947)
Kejia Gao (03779844)
Jingyi Zhang (03785924)
Last compiled: 2024–05–13

The work on tasks was divided in the following way:

Yizhi Liu (03779947)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%
Kejia Gao (03779844)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%
Jingyi Zhang (03785924)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%

0 Introduction

Vadere is an open-source tool for the simulation and visualization of human crowds. In this report, we'll not only describe how to use its graphical user interface, but also set up, run, and modify simulation scenarios, and integrate a new SIR model into the software. Compared to Exercise 1, Vadere makes it possible to design various scenarios simply in GUI with mouse operation and instant visual feedback rather than writing Python codes. Besides, it enables users to save simulations as results.

1 Task 1

Report on task TASK 1, Setting up the Vadere environment

1.1 Setup

Here is an example of environmental setup in Windows.

First of all, it is required to install OpenJDK (17 or above) and Maven 3.9.6 and set the environment variables. Open the page of Environment Variables, add new system variables with the variable name "JAVA_HOME" and variable value "`...\Java\jdk-21`". Double click the system variable "Path" and add two new variables: `...\apache-maven-3.9.6\bin` and `%JAVA_HOME%\bin`.

Afterwards, it is supposed to install IntelliJ IDEA Community Edition, which serves as the IDE for following development tasks.

Thirdly, download the Master Branch version of Vadere from `https://gitlab.lrz.de/vadere/vadere`.

Fourthly, open the PowerShell, switch to `...\vadere` as the working directory and input following commands:

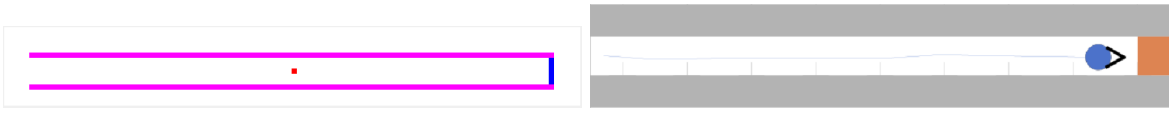
1. mvn clean (Clean up the code from the last compilation.)
2. mvn -Dmaven.test.skip=true package OR mvn "-Dmaven.test.skip=true" package (Build the code.)
3. java -jar VadereGui/target/vadere-gui.jar (Start the GUI.)

After taking these steps, the setup is finished and Vadere GUI is launched.

1.2 RiMEA scenario 1 (straight line)

In RiMEA scenario 1 [1], a $40m$ long and $2m$ wide corridor is modelled and a pedestrian placed on one end of the corridor is supposed to move in a straight line with correct speed towards the target on the other end. No obstacle is set along the path. The initial speed of the pedestrian is $1.33m/s$ and the travel time should range from 26 to 34 seconds. Optimal Steps Model is selected by clicking "Model" and "Load template" in the menu.

Figure 1b depicts the simulation of RiMEA Scenario 1 with Optimal Steps Model. The pedestrian moves nearly straight towards the target. According to Figure 1a, the simulation with OSM in Vadere is almost identical to that in Exercise 1, where the trajectory is a straight line.



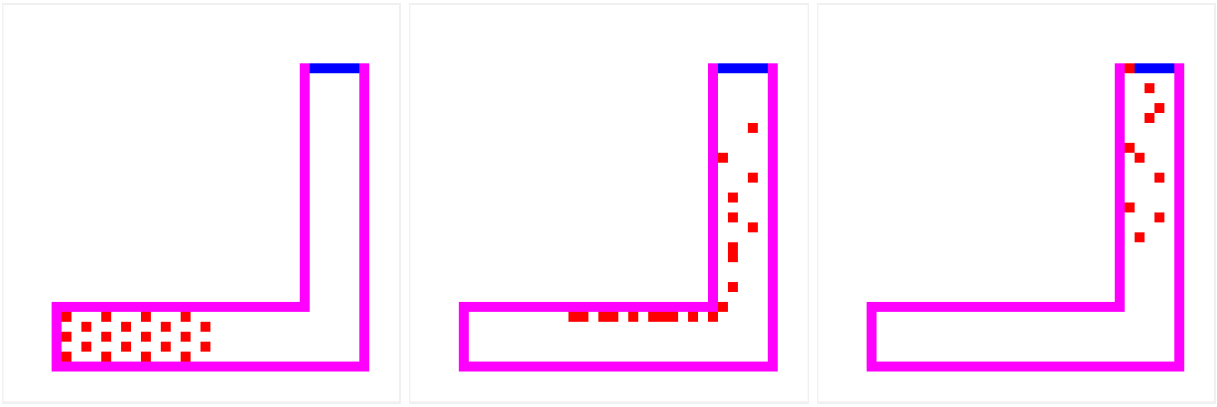
(a) Exercise 1 (b) Exercise 2, with OSM model

Figure 1: Visualization of RiMEA Scenario 1 in Exercise 1 and 2

1.3 RiMEA scenario 6 (corner)

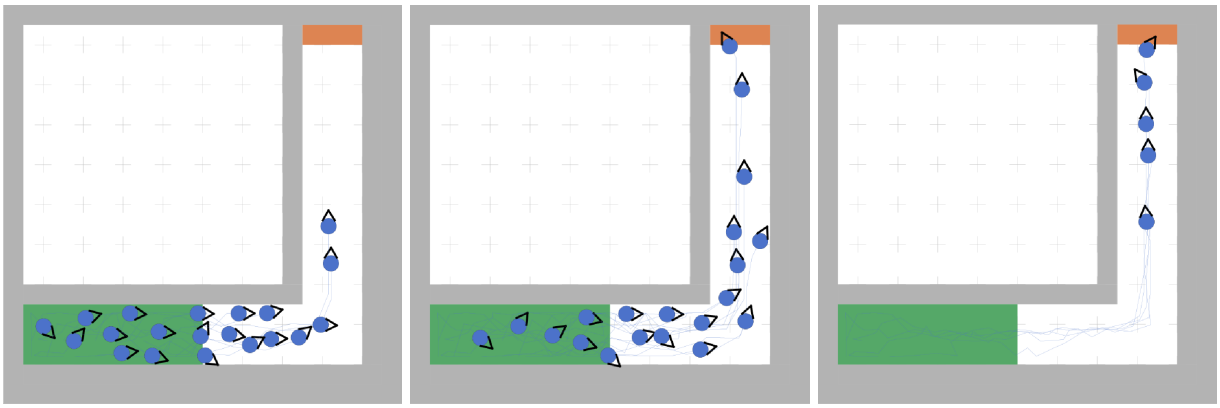
In RiMEA scenario 6 [1], a group of twenty pedestrians uniformly distributed at bottom left, heading toward a left-turning corner will navigate around it successfully without intersecting any walls. The target is at top right.

Figure 2 and Figure 3 demonstrate the comparison of results of simulations in Exercise 1 and 2 (OSM). The two simulations are similar to each other because both crowds of the pedestrians manage to arrive at the target without intersecting the obstacles. But there exist slight differences. In Exercise 1, the pedestrians stand in a line before turning left at the corner one by one. In contrast, the crowd in the simulation with OSM is more realistic, where 2 pedestrians can pass the corner simultaneously and the space at the corner is better utilized.



(a) At the start (b) In the middle (c) Near the end

Figure 2: Visualization of RiMEA scenario 6 in Exercise 1



(a) At the start (b) In the middle (c) Near the end

Figure 3: Visualization of RiMEA scenario 6 with OSM in Exercise 2

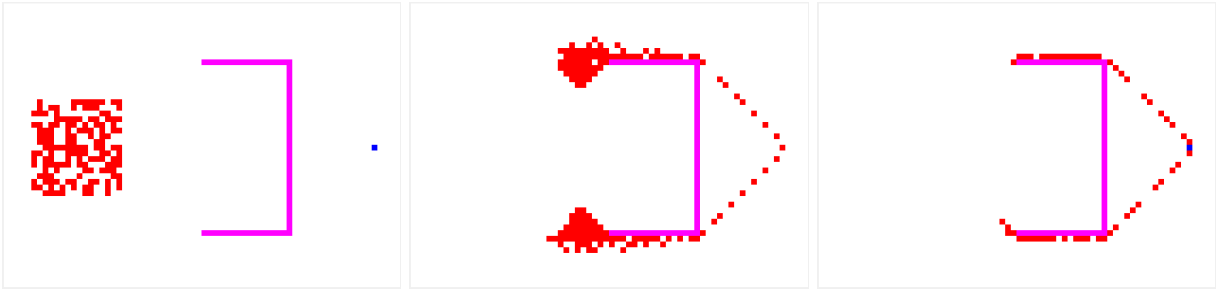
1.4 "Chicken test"

In "chicken test", the crowd should bypass a U-shaped obstacle without getting trapped inside the walls. They are facing the entrance of the obstacle, while the target is located to the right of the obstacle.

Figure 4 and Figure 5 show the visualization of "chicken test' in both exercises. In both simulations, the pedestrians bypass the obstacle successfully without getting stuck, although some of them get in the obstacle temporarily and cause congestion at the two ends of the obstacle. However, the difference lies in the crowd behaviour near the end of the simulation process. Figure 4c shows that the pedestrians in simulation of Exercise 1 go straight in two groups towards the target and arrive at it one by one, while Figure 5c indicates that the pedestrians in simulation with OSM surround the target and get in it in all directions.

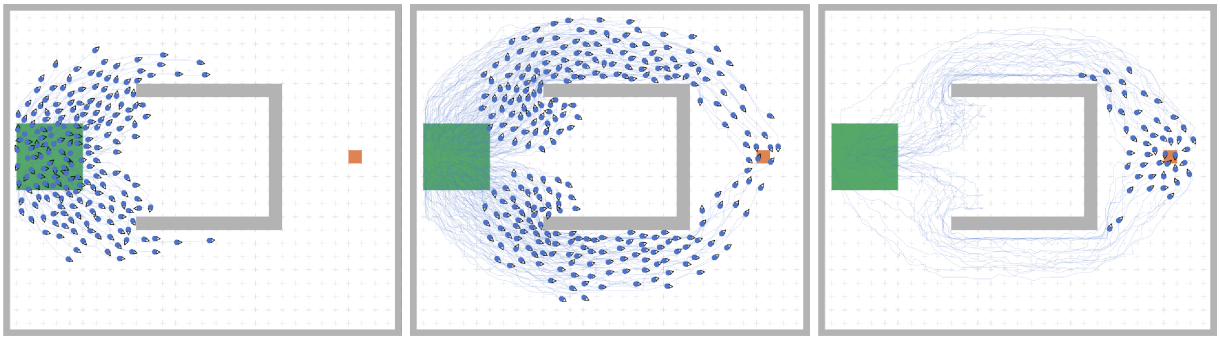
1.5 "Comparison of user interface and visualization"

The GUI of cellular automaton in Exercise 1 only contains several buttons like "load simulation", "run/stop simulation", "show/hide distances", etc. The users can't create or modify the scenario in the GUI. Instead, they



(a) At the start (b) In the middle (c) Near the end

Figure 4: Visualization of chicken test in Exercise 1



(a) At the start (b) In the middle (c) Near the end

Figure 5: Visualization of chicken test with OSM in Exercise 2

have to turn to the python code to generate json files. When running the simulation, only the cellular automatons are visualized, without any other information. In comparison, Vadere offers users more possibilities. It enables users to generate new scenarios via the GUI by clicking and dragging instead of writing codes. Moreover, users are able to show or hide pedestrians, trajectories, walking directions and distances. Different groups can be classified by assigning different colors. Recording the simulation as a video is also available for saving the visualization of the whole process.

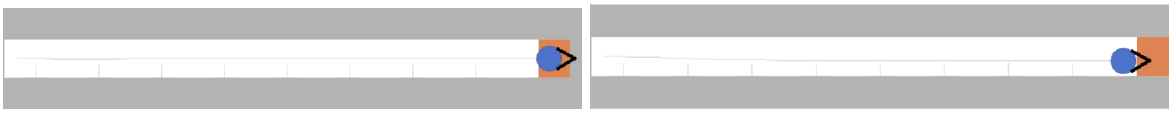
2 Task 2

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

In Task 2, we are required to implement two other models named Social Force Model (SFM) and Gradient Navigation Model (GNM) to simulate the three scenarios in Task 1. Like OSM, SFM and GNM are also available by clicking "Load templates" in the menu. With the results, we'll analyze the differences and similarities among OSM, SFM and GNM and discuss possible reasons.

2.1 RiMEA scenario 1 (straight line)

The trajectories of the 3 models are approximately straight lines but there are slight differences. In the OSM (Figure 1b), the pedestrian walks slightly to the right of forward at first, then slightly to the left of forward to compensate for the vertical distance, and finally walks slightly to the right of forward again. In the SFM (Figure 6a), the pedestrian moves slightly to the right of forward at first , then compensates for the vertical movement and moves along the axis of the corridor. In the GNM (Figure 6b), the pedestrian walks in a forward but slightly rightward direction until getting to the target.

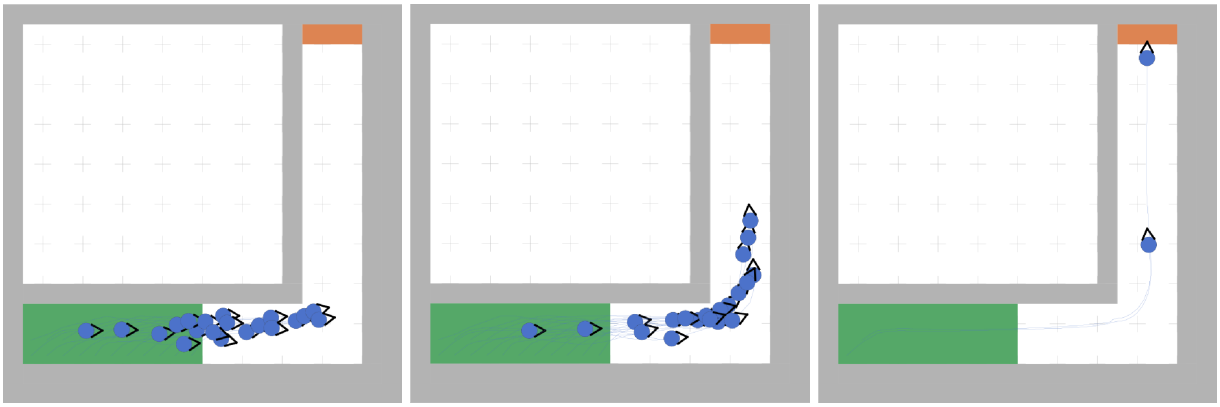


(a) SFM (b) GNM

Figure 6: Visualization of RiMEA Scenario 1 in Exercise 2

2.2 RiMEA scenario 6 (corner)

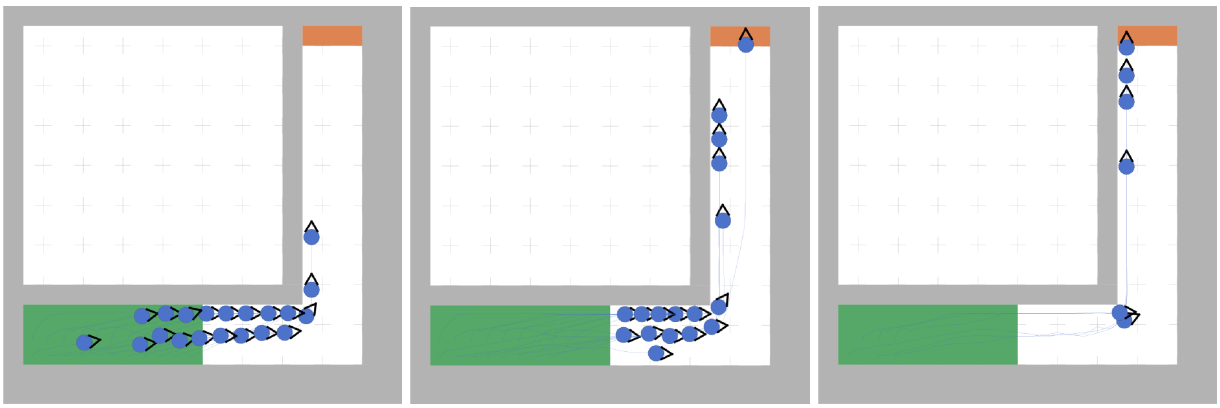
Figure 7 presents the simulation of SFM in RiMEA scenario 6. In comparison to OSM (Figure 3), the crowd in SFM are less sensitive to the mutual avoidance because the pedestrians can get very close to each other when passing the corner, while the people in OSM change the directions very frequently in order to avoid overlapping at the corner. Therefore, the trajectories in SFM are smoother, while those in OSM are zigzag, which means that the travel distances and travel time of SFM should be slightly smaller than those of OSM.



(a) At the start (b) In the middle (c) Near the end

Figure 7: Visualization of RiMEA scenario 6 with SFM in Exercise 2

Figure 8 reveals the simulation of GNM in RiMEA scenario 6. Compared with OSM (Figure 3), the pedestrians in GNM prefer to move orderly because they slow down at the corner and wait those in the front to pass the corner one by one, while the people in OSM maintain the speed, look for new positions available and move very flexibly. Therefore, the trajectories in GNM are much smoother than those in SFM. However, the drawback of good order is lower efficiency, since the pedestrians in GNM wait to pass one after another.

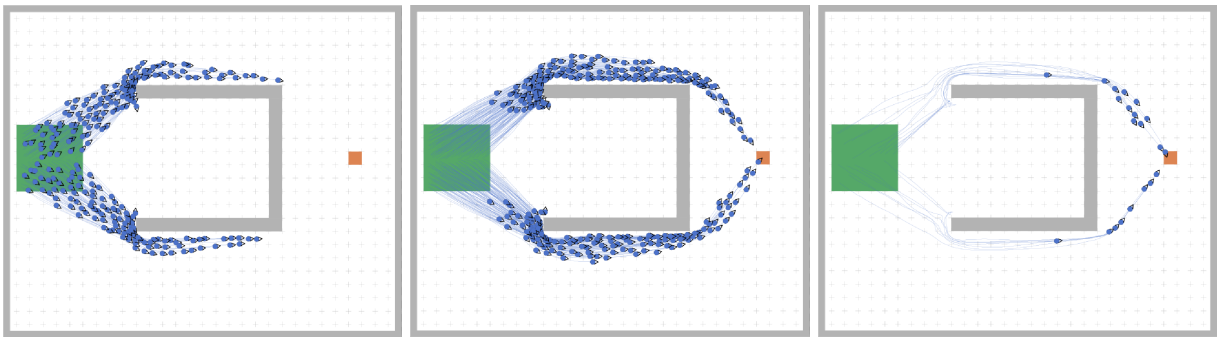


(a) At the start (b) In the middle (c) Near the end

Figure 8: Visualization of RiMEA scenario 6 with GNM in Exercise 2

2.3 "Chicken test"

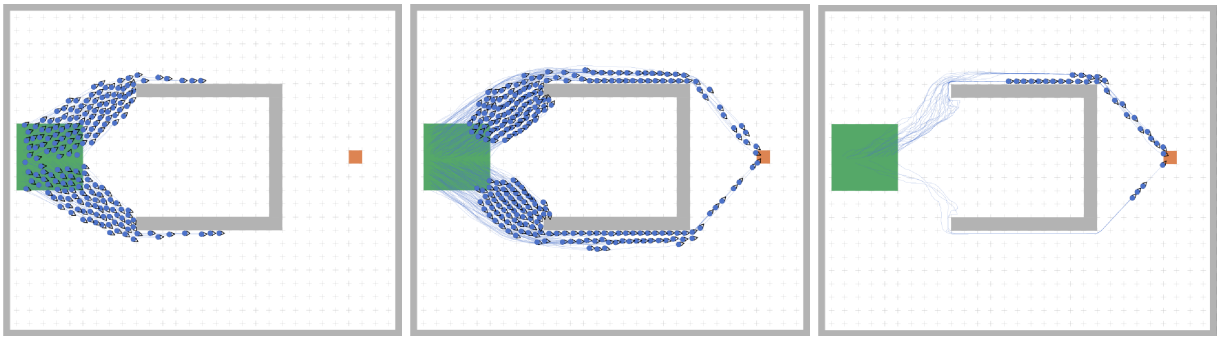
Figure 9 conveys the simulation of SFM in "chicken test". In comparison to OSM (Figure 5), the crowd in SFM prefer to seek for shortest path regardless of getting close together, while the people in OSM move flexibly to keep mutual distances. After bypassing the obstacle, the crowd in SFM move straight towards the target, while the crowd in OSM surround the target and get in from all directions. Therefore, the trajectories in SFM are smooth and have overlapping, while those in OSM are zigzag.



(a) At the start (b) In the middle (c) Near the end

Figure 9: Visualization of chicken test with SFM in Exercise 2

Figure 10 illustrates the simulation of GNM in "chicken test". Unlike OSM (Figure 5), the crowd in GNM tend to stand in a line and bypass the obstacle one after another. Afterwards, the crowd in GNM move in a straight line towards the target, while the crowd in OSM enter the target from all sides. Therefore, the people in GNM spend more time getting to the target than those in OSM in this scenario.



(a) At the start (b) In the middle (c) Near the end

Figure 10: Visualization of chicken test with GNM in Exercise 2

2.4 Summary of 3 models in 3 scenarios

After carrying out simulations and comparisons of the three models, their characteristics can be revealed. In term of similarity, all of them can make the crowd bypass the obstacles and reach the target effectively. Both OSM and GNM are sensitive to the small mutual distances among pedestrians, which is contrary to SFM. SFM and GNM have smoother trajectories than OSM. OSM prefers to let the crowd seek for new position and change the direction flexibly in order to ensure pedestrian avoidance, which causes the trajectories to be zigzag and the moving distance of each pedestrian is slightly larger. SFM has a higher weight for shorter paths and allows the pedestrians to move parallel regardless of very short distances between pedestrians. GNM makes the crowd act more orderly and they tend to pass the obstacles one by one in order to keep the distances at the cost of spending more time.

3 Task 3

Report on task TASK 3, Using the console interface from Vadere

The objective of this task is to utilize the console of Vadere to execute a specific scenario file. Detailed instructions will be provided to ensure the correct execution process is followed using Vadere's command-line interface. The specific operating instructions are as follows

```
java -jar vadere-console.jar scenario-run
 --scenario-file "/path/to/the/file/scenariofilename.scenario"
 --output-dir "/path/to/output/folders"
```

There is no difference between entering vadere through the console and directly clicking on the vadere.gui file.

3.1 Adding pedestrians programmatically

In order to add new pedestrians to the original scene, the add_pedestrians.py file was created. Its main function is to add pedestrians with determined positions in any scenario. In the add_pedestrians.py file, a method update scenario is defined. To update a scenario file by incorporating additional pedestrian information, including position, speed, target, and other pertinent attributes, a structured approach should be adopted. The following guidelines outline the formal procedure for modifying the scenario file in Vadere:

- Initial Data Collection Open the Vadere simulation tool and navigate to the scene creation page. Configure the parameters for the new pedestrian, setting up attributes such as initial position, speed, and target destination.
- Information Extraction Upon the creation of the desired pedestrian configuration in Vadere, locate the pedestrian's attributes.
- Python Script Update Use a Python script to automate the inclusion of the pedestrian data into the scenario file. Open the Python file responsible for processing scenario data. Paste the copied pedestrian information into the appropriate section of the script. For generating additional pedestrians, modify the position data accordingly within the script to avoid overlapping coordinates or paths.
- File Modification and Saving After updating the pedestrian information, save the modified scenario data back into the scenario file. Ensure that the file is saved to a predefined path that aligns with the project's directory structure and file management protocols.

The updated scene is as shown below

3.2 Discussion

The new pedestrian position is placed in the coordinates written x=17, y=3, and the speed is the default speed. After running the scene, the newly placed pedestrian reaches the target first, taking 13.6 seconds. This makes sense because the new pedestrian is closer to the target than the other pedestrians.

4 Task 4

Report on task TASK 4, Integrating a new model

To utilize the SIR model, one must first clone the Vadere source code from the repository and then use IntelliJ to compile the Java file, thereby updating the pedestrian infection status.

4.1 Description of the SIRGroupModel

The SIR model is a classical epidemiological model that segments a population into three groups or compartments to study the spread of infectious diseases.
The SIRGroupModel extends AbstractGroupModel ¡SIRGroup¿ to manage simulation groups based on the SIR (Susceptible-Infectious-Recovered) infection model within a pedestrian dynamics simulator. Key functionalities include:

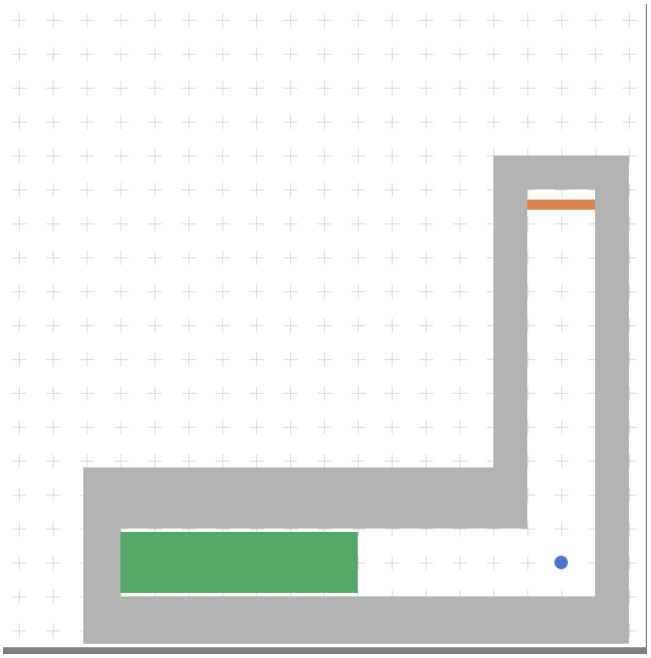


Figure 11: Updated rimea6 scenario

- Initialization: Sets up internal structures and attributes for infection control.
- Group Management: Maintains and dynamically updates groups of pedestrians based on their infection status.
- Event Handling: Responds to additions and removals of pedestrians, reassigning group memberships accordingly.
- Simulation Updates: Implements logic to simulate infection spread among pedestrians based on proximity and infection rates. Engages at key points in the simulation cycle to adjust group statuses and respond to changes in the simulation environment.

The main files called in the SIRGroupModel file are AttributesSIRG, SIRGroup, and SIRType. They respectively define:

- AttributesSIRG: Infections at Start, Infection Rate and Infection Maximum Distance
- SIRType: categorizes individuals into three statuses—infected, susceptible, and removed
- SIRGroup: includes methods addMember and removeMember to manage the membership of pedestrians within a group

4.2 Output Processors

After running the scene, the pedestrian's trajectory will be stored in the output file in the form of a csv file, and other detailed SIR information is also recorded, which can be used for visualization.

4.3 LinkedCellsGrid

The file LinkedCellsGrid manages a 2D spatial grid that organizes objects with specific positions, improving query efficiency for operations like additions, deletions, and spatial searches within a defined radius. Efficiency Improvements:

- Grid Dimensions and Cell Management: The grid is defined by its top-left corner coordinates (left, top), its width, and height. It divides this space into cells, where each cell can hold multiple objects.

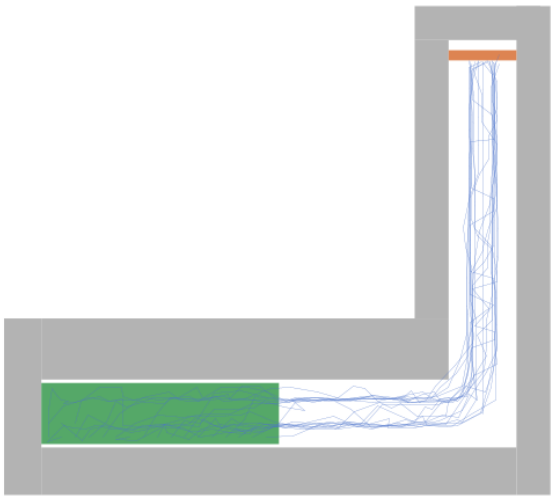


Figure 12: Visualization of chicken test with GNM in Exercise 2

- Spatial Queries: The grid supports methods to add, move, and remove objects. It allows querying for objects within a specified radius (getObjects), which is useful for range queries or proximity searches.
- Iterable Interface: Implements the Iterable interface, allowing iteration over all objects in the grid, facilitating operations like search or modification across all elements.

4.4 Visualization of the Groups

In order to visualize the SIR model in Vadere, the following codes need to be modified

- Coloring visualization: S, I, R groups are not displayed correctly in the current setup. Therefore, it is necessary to set the colors of infected people, susceptible people, and recovered people in the SimulatioModel.java file. The code is as follows In order to facilitate observation, in this experiment, the infected

```
// Colors for disease status in simulations: red for infected, green for
// recovered, blue for susceptible.

protected final Color infected_color = new Color(255, 0, 0);
protected final Color recovered_color = new Color(0, 255, 0);
protected final Color susceptible_color = new Color(0, 0, 255);
```

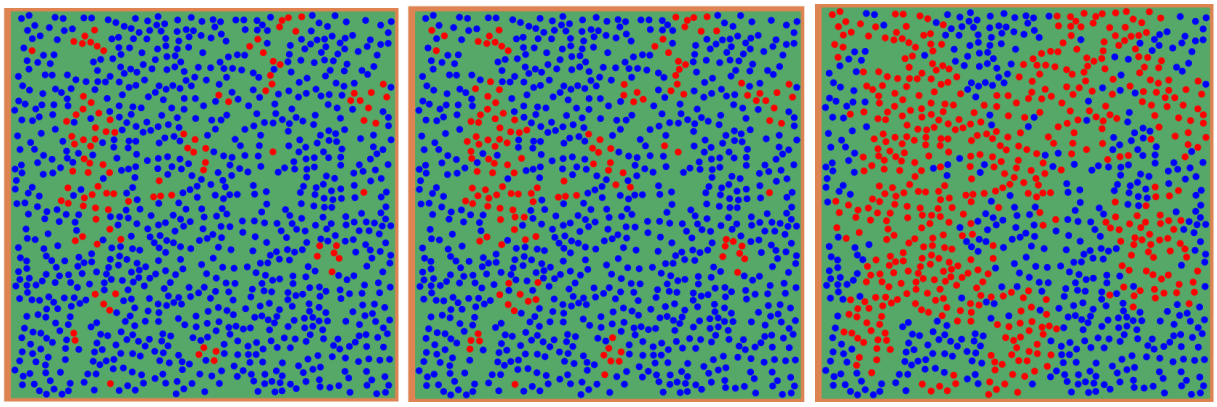
Figure 13: Color modify

  people are colored red, the recovered people are green, and the susceptible people are blue.

- Efficient update: The original idea in the code is to regard all pedestrians as infected objects, but according to daily experience, only people near the infected person have a probability of being infected. According to this idea, the update scope can be narrowed to achieve high efficiency. To achieve efficient updates, you mainly need to use the LinkCellGrid class. The neighbors of a given pedestrian can be directly obtained through the internal getObjects method, and the update of infected people has been implemented. In this way, you do not need to visit all pedestrians, but only need to search for neighbors around the infected person, and determine whether all neighbors will be infected based on the maximum infection radius.

In this task, a rather large scene needs to be constructed, which contains 1000 pedestrians, and the target and source overlap. Since the infection process needs to be observed in this task, the absorbing status is set to false.

The parameters in this experiment are



(a) At the start (b) In the middle (c) Near the end

Figure 14: Visualization of SIR model

- infectionsAtStart : 10
- infectionsRate:0.2
- infectionsMaxDistance: 10

The time it takes for half of the people to be infected is about 120 seconds. See the next section for details.

4.5 Increase the infectionRate

In order to study the infection rate and the time it takes for the total number of people infected to be half, in this experiment the infection rate was simply doubled to 0.4. Repeat the above experiment under the same circumstances to obtain time information on the number of infected people. The comparison shows that when the infection rate doubles, the time required to infect half of the people is about half as long as before. Use the Dash/Plotly visualization utility to visualize the change in the number of infected and susceptible people over time. At the same time, comparison of multiple simulation processes can also be achieved.

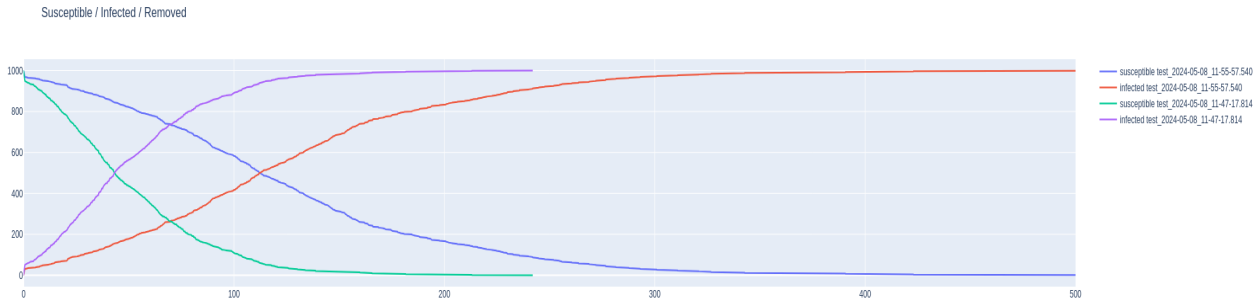
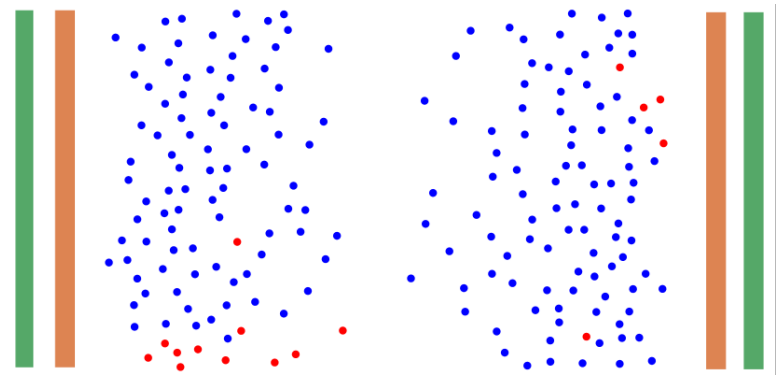


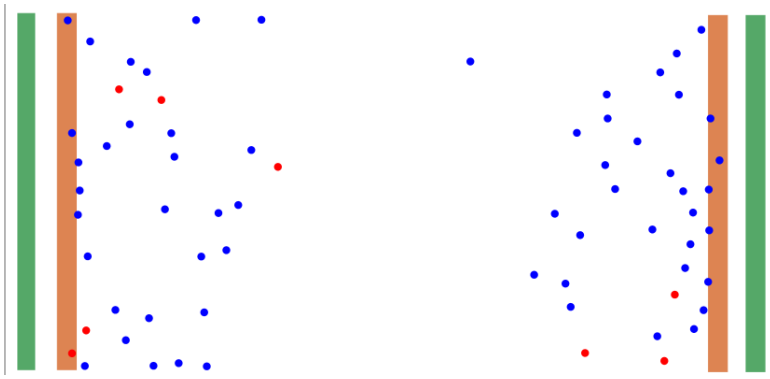
Figure 15: Comparison of changing infection rates

4.6 Corridor scenario

In this task, a 40m×20m corridor scene needs to be constructed. A group of 100 people moves from left to right, and another group of people (also 100) moves from right to left. Absorbable targets need to be set.



(a) At the start



(b) In the middle

Figure 16: Visualization of corridor scenario

In the corridor scene, two groups of people meet and interact in the middle of the scene. Visualizing the infection process through Dash/Plotly was able to identify approximately 20 healthy pedestrians who were infected during the interaction.

Figure 17: Corridor scene infection changes over time

4.7 Extensions

Three possible extensions are expected to be applied in models.

Age. People of various ages have different probabilities of getting the disease, for example the elderly and children are more likely to be infected. Therefore, a variable "age" can be integrated into the attributes of the pedestrians, which affects the infection rate of every individual.

Vaccination. Vaccination is of great significance in preventing diseases, and booster shots will further enhance immunity. Therefore, the number of vaccinations is also a variable that can be added to the attributes

of the pedestrians potentially.

Hygiene measures. The government has taken extensive security measures to block the spread of the virus. For example, targets may deny access to infected persons, which may cause virus carriers to stay away from crowded targets. Embodied in the programs, a binary variable in the attributes of the target determines whether infected people can approach it.

5 Task 5

Report on task TASK 5, Analysis and visualization of results

5.1 Implementation of recovery state and recovery rate

Listing 1 demonstrates how the recovery state and recovery rate are implemented. Each infected person is likely to get infected. The probabilistic event is simulated through random numbers. $this.random.nextDouble()$ is a generated random number between 0 and 1, while $attributesSIRG.getRecoveryRate()$ is the preset recovery rate. Since the generated random number is uniformly distributed in the interval $[0, 1)$, the probability that it is less than the recovery rate is exactly equal to the recovery rate. Therefore, by setting an if condition: $this.random.nextDouble() < attributesSIRG.getRecoveryRate()$, the simulation of pedestrian getting recovered at the recovery rate can be achieved. If the pedestrian is recovered, he/she will be removed from the group of INFECTED and be added into the group of RECOVERED.

```
if (currentGroup.getID() == SIRType.ID_INFECTED.ordinal()) {
    if (this.random.nextDouble() < attributesSIRG.getRecoveryRate()) {
        elementRemoved(currentPedestrian);
        assignToGroup(currentPedestrian, SIRType.ID_REMOVED.ordinal());
        continue;
    }
}
```

Listing 1: Java code of implementation of recovery state and recovery rate

The SIR visualization is modified from the original demo version offered in Moodle, where only the state S and I can be visualized. The adaption should enable the implementation to visualize all of the three states. Listing 2 demonstrates the adapted codes. Depending on the initial state of each pedestrian (current_state), the code categorizes them into one of three groups (group-s for susceptible, group-i for infected, group-r for recovered) and increments the count for that group starting from the beginning of the simulation time. As the code iterates through each time and state record for a pedestrian, it updates the number of susceptible, infected and recovered groups accordingly for all simulation times after the transition. Finally, the function returns the group_counts, a DataFrame which includes updated counts of pedestrians in each group for each simulation time step.

```
for pid in pedestrian_ids:
    simtime_group = df[df['pedestrianId'] == pid][['simTime', 'groupId-PID5']].values
    current_state = simtime_group[0][1]
    if current_state == ID_SUSCEPTIBLE:
        group_name = 'group-s'
    elif current_state == ID_RECOVERED:
        group_name = 'group-r'
    elif current_state == ID_INFECTED:
        group_name = 'group-i'

    group_counts.loc[group_counts['simTime'] >= 0, group_name] += 1
    for (st, g) in simtime_group:
        if g == ID_RECOVERED and current_state == ID_INFECTED:
            current_state = ID_RECOVERED
            # for each time step later, increase the number of recovered by 1 and
                decrease that of the infected by 1
            group_counts.loc[group_counts['simTime'] >= st, 'group-r'] += 1
```

```
            group_counts.loc[group_counts['simTime'] >= st, 'group-i'] -= 1
            break
        # if the susceptible pedestrian becomes infected
        elif g == ID_INFECTED and current_state == ID_SUSCEPTIBLE:
            current_state = ID_INFECTED
            # for each time step later, increase the number of infected by 1 and
                decrease that of the susceptible by 1
            group_counts.loc[group_counts['simTime'] >= st, 'group-i'] += 1
            group_counts.loc[group_counts['simTime'] >= st, 'group-s'] -= 1
```

Listing 2: Python code of visualization of SIR

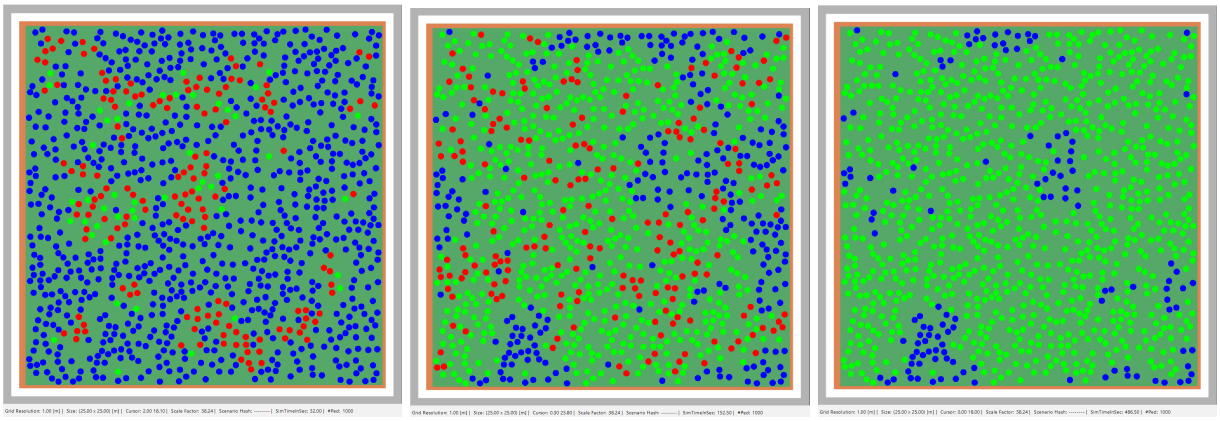
In addition, the original code of the function create_folder_data_scatter(folder) does not contain the information of the group of the recovered, so it is supposed to add scatter_r as Listing 3 shows.

```
def create_folder_data_scatter(folder):
    ...
    scatter_r = go.Scatter(x=group_counts['simTime'],
                           y=group_counts['group-r'],
                           name='recovered ' + os.path.basename(folder),
                           mode='lines')
    return [scatter_s, scatter_i, scatter_r], group_counts
```

Listing 3: Python code of the function create_folder_data_scatter(folder)

5.2 Test 1

In test 1, 1000 pedestrians are randomly distributed in a large rectangle, with 10 infective and 990 susceptible at the start. Both of the infection rate and recovery rate are 0.02. Figure 18 depicts the visualization of Test 1 in Vadere and Figure 19 shows the curve of SIR. The infection is slow and the number of infected people reaches its maximum of 253 at $t = 102.2s$. As time goes on, most of the people are recovered. In the end, 877 of them are recovered, while the rest are still susceptible surrounded by the recovered.



(a) At the start ($t = 32s$) (b) In the middle ($t = 152.5s$) (c) In the end ($t = 486.5s$)

Figure 18: Visualization of Test 1 in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.02$

5.3 Test 2

In Test 2, it will be tested how the infection and recovery rate can influence the results. The scenario of Test 2 is the same as that of Test 1, with 10 infective and 990 susceptible in the rectangle at the start. By using control variates, Test 2 are split into 2 subtests: Test 2a and Test 2b.

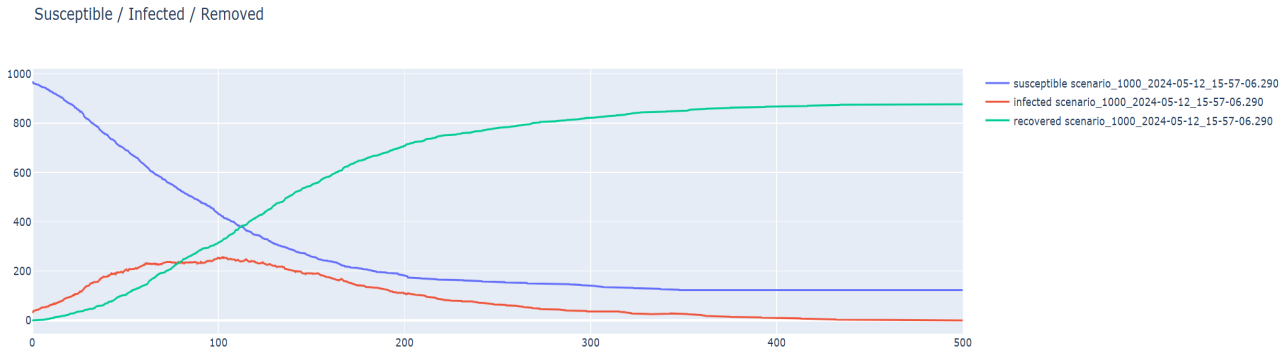
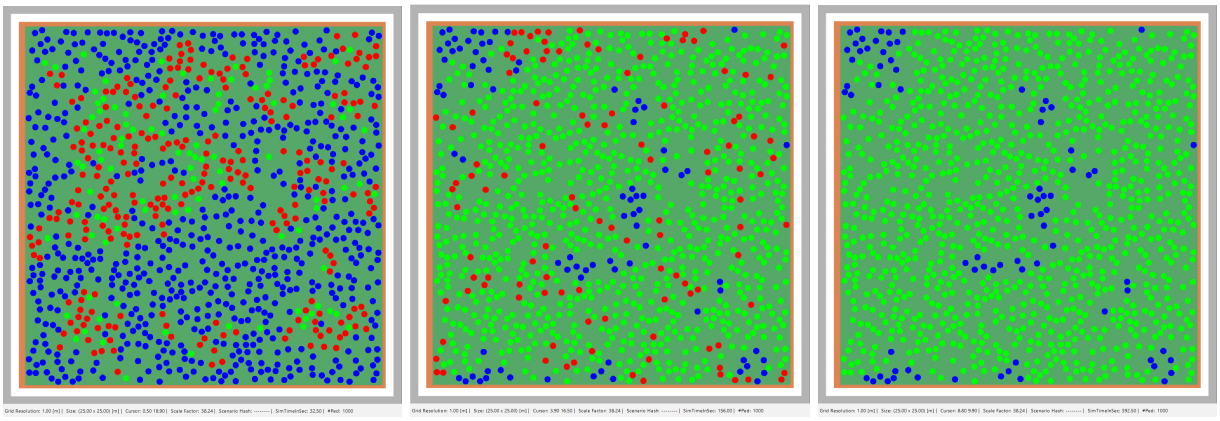


Figure 19: SIR curve of Test 1 in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.02$

5.3.1 Test 2a

In Test 2a, the infection rate is raised from 0.02 to 0.03 and 0.04, while the recovery rate remains at 0.2, so that we can determine the influence of the increase of the infection rate.

As we see in Figure 20 and Figure 21, when the infection rate is 0.03, the number of infected people reaches its maximum of 382 at $t = 68s$. In the end, 924 of them are recovered, while 76 people are still susceptible.



(a) At the start ($t = 32.5s$) (b) In the middle ($t = 156s$) (c) In the end ($t = 392.5s$)

Figure 20: Visualization of Test 2a in Task 5, with $infection\,rate = 0.03, recovery\,rate = 0.02$

Figure 22 and Figure 23 display that when the infection rate is 0.04, the number of infected people reaches its maximum of 473 at $t = 48.4s$. In the end, 976 of them are recovered, while 24 people are still susceptible.

5.3.2 Test 2b

In Test 2b, the recovery rate increases from 0.02 to 0.03 and 0.04, while the infection rate remains at 0.2, so that we can determine the influence of the increase of the recovery rate.

As we see in Figure 24 and Figure 25, when the recovery rate is 0.03, the number of infected people reaches its maximum of 158 at $t = 46.3s$. In the end, 698 of them are recovered, while 302 people are still susceptible.

Figure 26 and Figure 27 reveal that when the recovery rate is 0.04, the number of infected people reaches its maximum of 78 at $t = 40s$. In the end, 328 of them are recovered, while 672 people are still susceptible.

5.3.3 Summary of Test 2

According to the above-mentioned results, infection rate and recovery rate have different effects on the model. As the infection rate increases (from 0.02 to 0.04), the curve of the infected reaches its peak earlier with larger maximum (from 253 to 473), and the total number of the infected increases from 877 to 976, which means that the infection spreads faster and more widely. In contrast, higher recovery rate (from 0.02 to 0.04) makes the

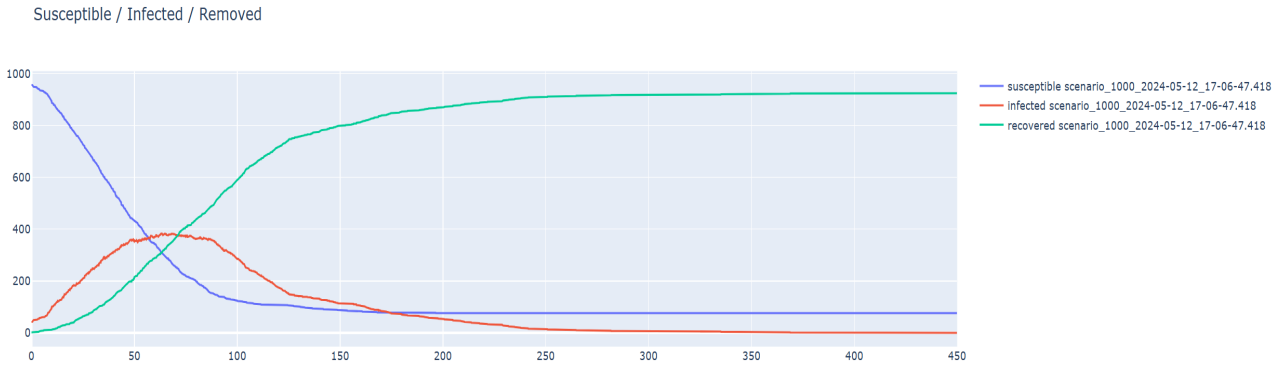
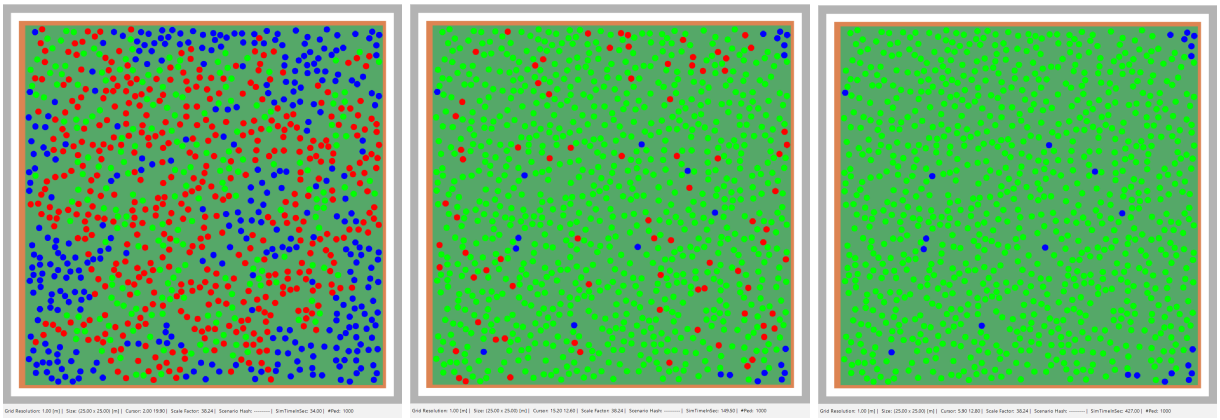


Figure 21: SIR curve of Test 2a in Task 5, with $infection\,rate = 0.03, recovery\,rate = 0.02$



(a) At the start ($t = 34s$) (b) In the middle ($t = 149.5s$) (c) In the end ($t = 427s$)

Figure 22: Visualization of Test 2a in Task 5, with $infection\,rate = 0.04, recovery\,rate = 0.02$

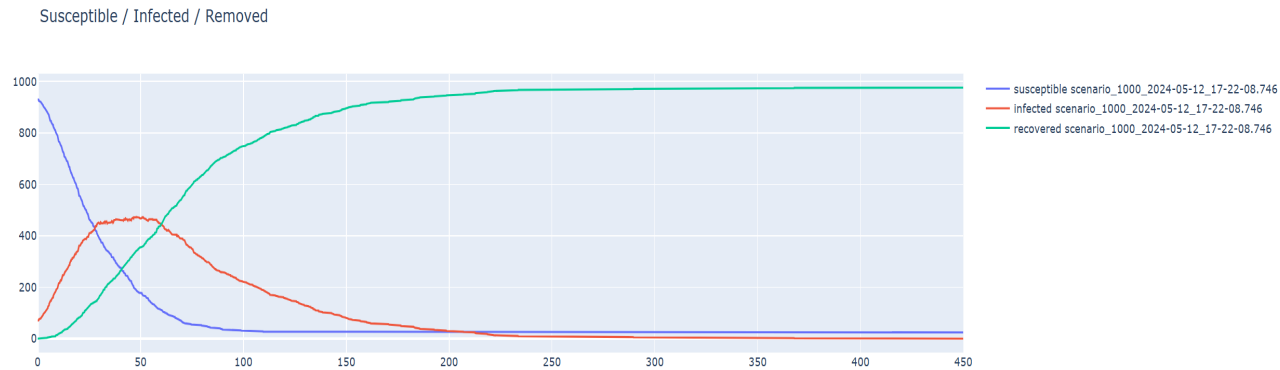
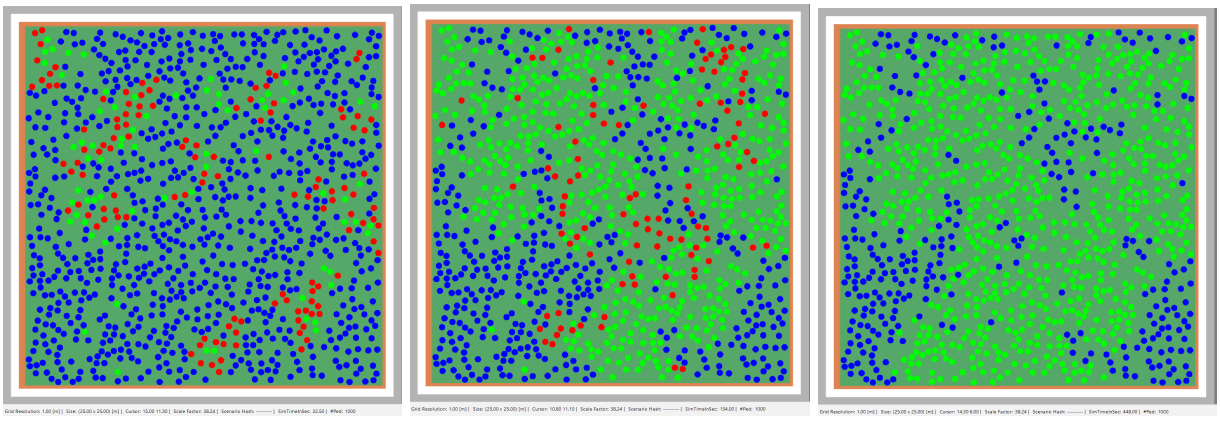


Figure 23: SIR curve of Test 2a in Task 5, with $infection\,rate = 0.04, recovery\,rate = 0.02$



(a) At the start ($t = 32.5s$) (b) In the middle ($t = 154s$) (c) In the end ($t = 448s$)

Figure 24: Visualization of Test 2b in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.03$

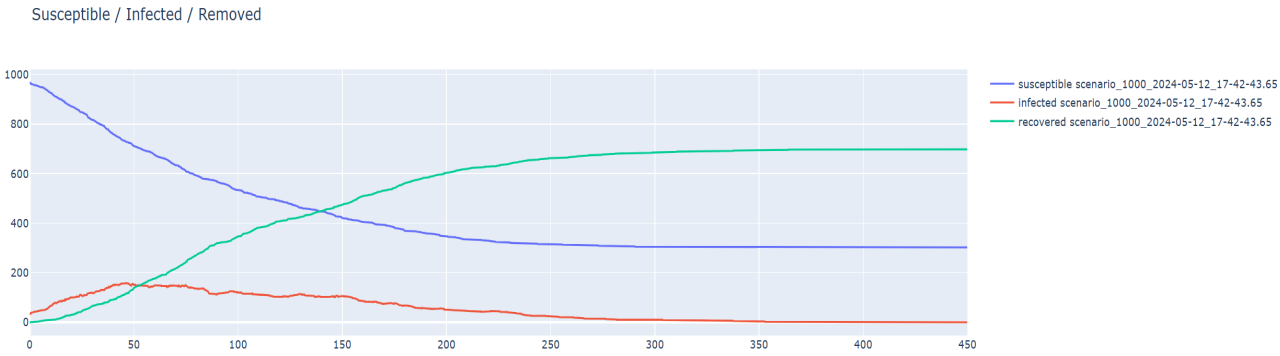
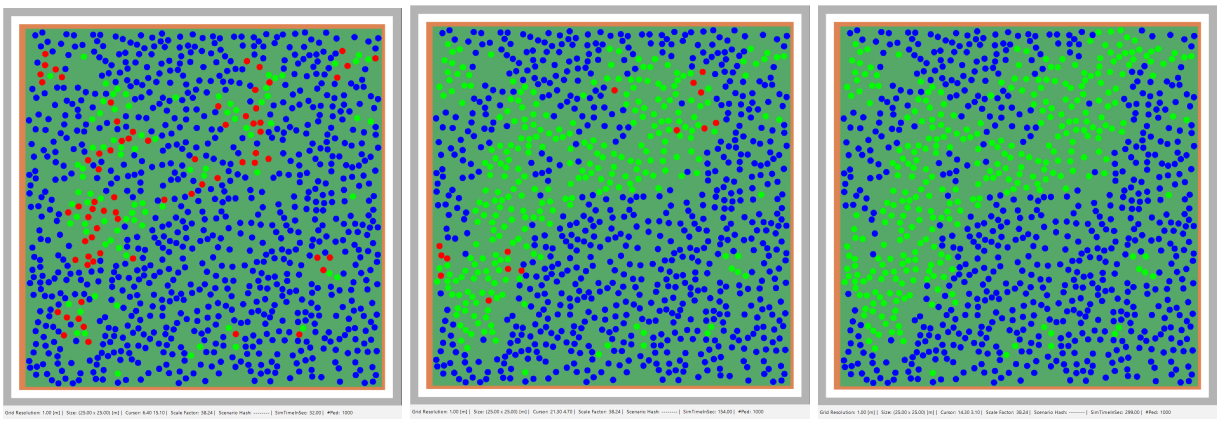


Figure 25: SIR curve of Test 2b in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.03$



(a) At the start ($t = 32s$) (b) In the middle ($t = 154s$) (c) In the end ($t = 299s$)

Figure 26: Visualization of Test 2b in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.04$

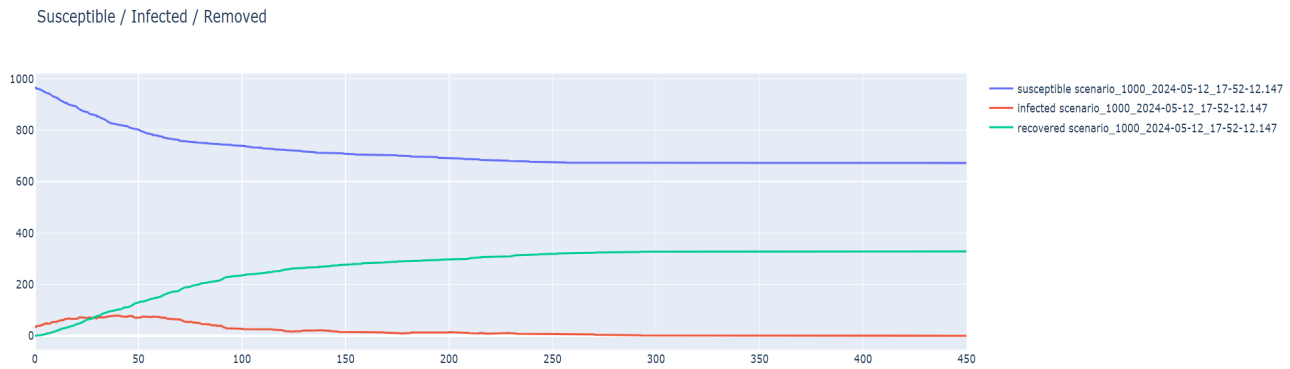


Figure 27: SIR curve of Test 2b in Task 5, with $infection\,rate = 0.02, recovery\,rate = 0.04$

maximum of the infected much lower (from 253 to 78), and the total number of the infected decreases from 877 to 328, which indicates the spread of the virus becomes worse. In a word, a high infection rate causes the virus to spread rapidly and widely, while a high recovery rate leads to slower, milder, and more contained decrease transmission.

5.4 Test 3

In test 3, a scenario of a simplified supermarket will be created, in which a series of test about will be executed. The tests are about keeping social distance and reducing infections in public areas and will be divided into 4 sub tests.

5.4.1 Test 3.1

In test 3.1, a scenario of $30 \times 30m$ will be created, as shown in Figure 28 , this scenario have a infection rate of 0.015 and a recovery rate of 0, since it's nearly impossible to recover in a short period. The pedestrians will be spawned from the sources, wandering in the scenario and absorbed at the target if the top right corner.

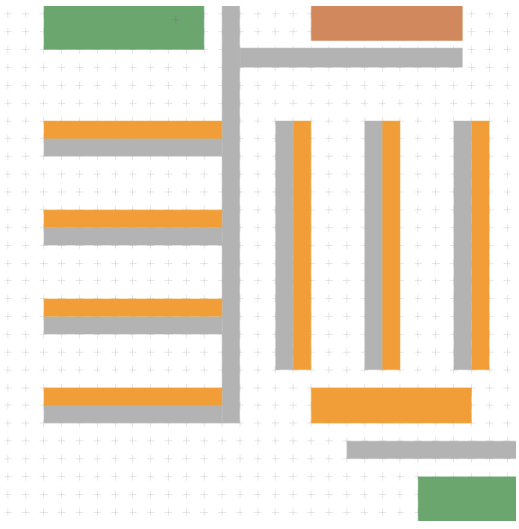
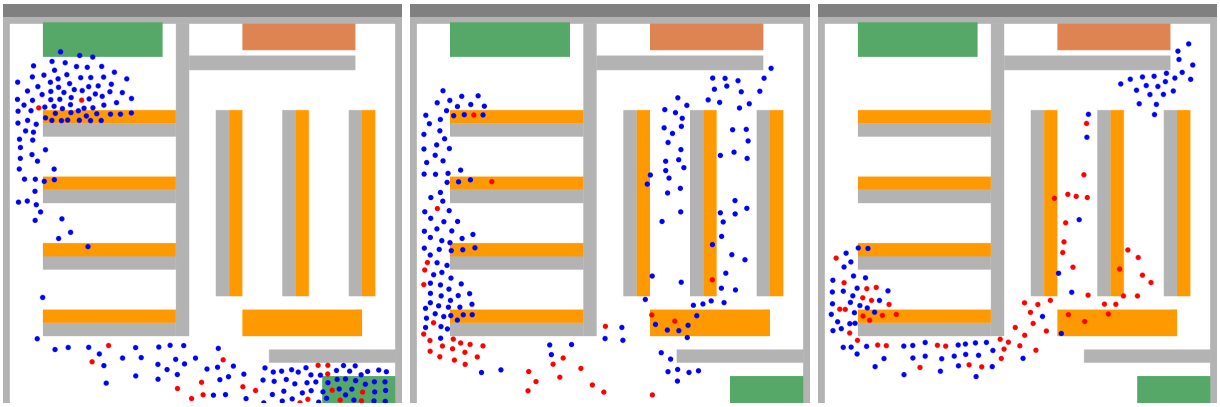


Figure 28: Scenario of a simplified supermarket

5.4.2 Test 3.2

In this sub test, the effect of increasing `pedPotentialPersonalSpaceWidth` will be demonstrated. The value of `pedPotentialPersonalSpaceWidth` is initially set to 1.0, presenting a relatively shorter social distance. Each

source generates 100 pedestrians, and there are 10 infected pedestrians in total. The simulation process is shown in Figure 29, And the result is shown in Figure 30.



(a) At the start ($t = 12.9s$) (b) In the middle ($t = 34.5s$) (c) In the end ($t = 65.7s$)

Figure 29: Visualization of Test 3.2 in Task 5, with `pedPotentialPersonalSpaceWidth` = 1.0

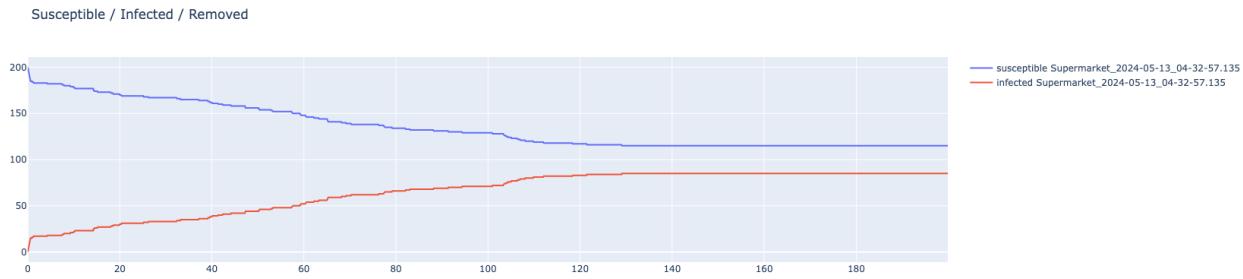


Figure 30: Result of Test 3.2 in Task 5, with `pedPotentialPersonalSpaceWidth` = 1.0

From the first simulation with `pedPotentialPersonalSpaceWidth` = 1.0, 75 pedestrians were infected in the process. With an increased `pedPotentialPersonalSpaceWidth`, the result would be different. Now the value of `pedPotentialPersonalSpaceWidth` is set to 3.0, presenting a relatively larger social distance. The simulation process is shown in Figure 31, And the result is shown in Figure 32.

Now only 35 pedestrians were infected in the simulation process.

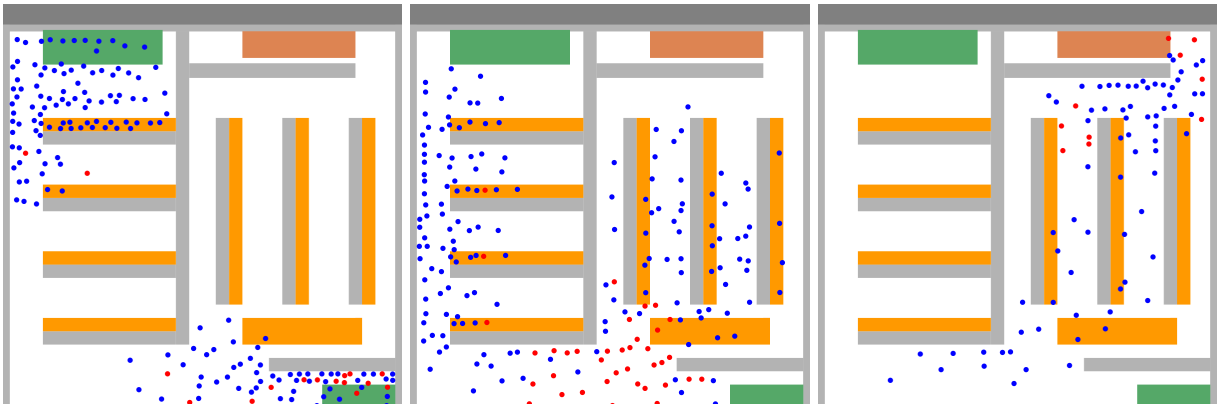
5.4.3 Test 3.3

From the simulations in test 3.2, it can be concluded that a increased `pedPotentialPersonalSpaceWidth`, or a larger "Social Distance", will lead to less infection in the supermarket. With the `pedPotentialPersonalSpaceWidth` increase from 1.0 to 3.0, the infections happened during the simulation is reduced by 46.7%.

Furthermore, by the Figure 32, it could be found that most of the infections happened in the early stage of the simulation. The reason behind it was, it was too crowded around the sources at the start. So by reducing the amount of pedestrians in the scenario, the result would be better.

5.4.4 Test 3.4

To reduce the amount of pedestrians in the supermarket, only 30 pedestrians would be generated from a source in this sub test. Among all the pedestrians, only 3 would be infected at the beginning. The value of `pedPotentialPersonalSpaceWidth` would still be set to 3.0. The simulation process is shown in Figure 33, And the result is shown in Figure 34.



(a) At the start ($t = 8.1s$) (b) In the middle ($t = 26.1s$) (c) In the end ($t = 72.9s$)

Figure 31: Visualization of Test 3.2 in Task 5, with `pedPotentialPersonalSpaceWidth` = 3.0

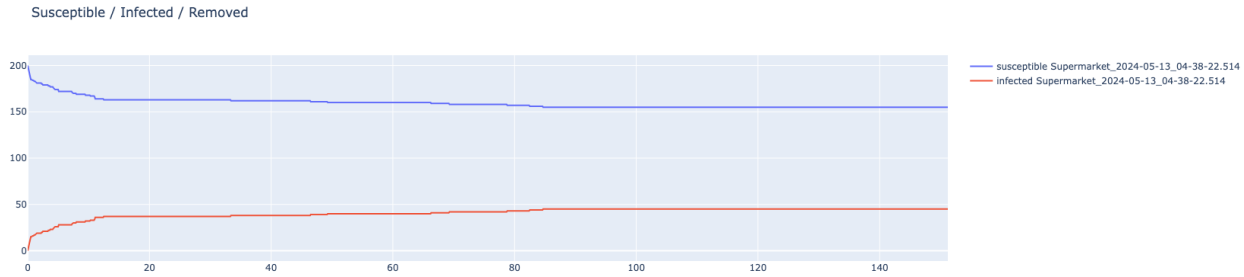


Figure 32: Result of Test 3.2 in Task 5, with `pedPotentialPersonalSpaceWidth` = 3.0

In this test, only 5 pedestrians were infected during the process. Compared to 35 new infection in 200 pedestrians (17.5%), 5 new infection in 60 pedestrians (8.3%) is significantly lower. So the limitation of the amount of the pedestrians will also help reducing the infections.

5.4.5 Summary of Test 3

According to the results above, both a larger social distance and a limitation of amount of pedestrians in the supermarket will lead to a less infection rate.

References

[1] RIMEA. *Guideline for Microscopic Evacuation Analysis.*

(a) At the start ($t = 9.9s$) (b) In the middle ($t = 24.9s$) (c) In the end ($t = 55.5s$)

Figure 33: Visualization of Test 3.4 in Task 5, with 60 pedestrians

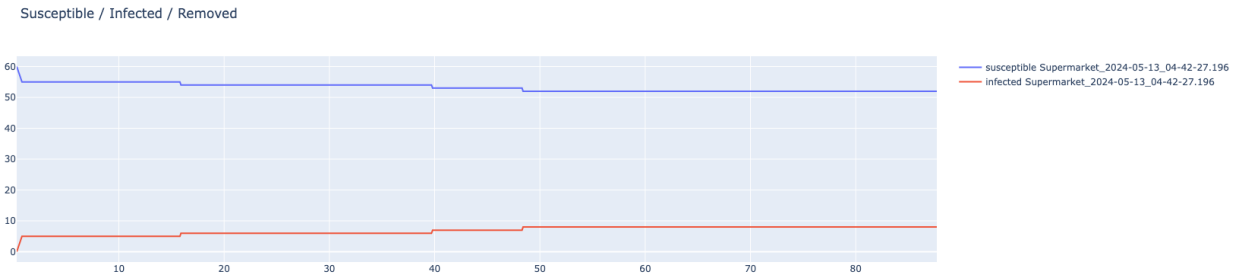


Figure 34: Result of Test 3.4 in Task 5, with 60 pedestrians