Report for exercise 2 from group f

Tasks addressed:　5
Authors:　Shihong Zhang (03764740)
　　　　　Mengshuo Li (03792428)
　　　　　Yuxuan Wang (03767260)
　　　　　Augustin Gaspard Camille Curinier (03784531)
　　　　　Qingyu Wang (03792094)
Last compiled:　2024–05–13

The work on tasks was divided in the following way:

Shihong Zhang (03764740)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Mengshuo Li (03792428)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Yuxuan Wang (03767260)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Augustin Gaspard Camille Curinier (03784531)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Qingyu Wang (03792094)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

Report on task 1, Setting up the Vadere environment

Introduction to Vadere

Vadere is a free, open-source simulation framework that specializes in modeling crowd dynamics. It features versatile model classes, as well as tools for visualization and data analysis. Designed to model pedestrian movements, Vadere is apt for various environments including public areas, buildings, or during evacuations.

Compared to earlier tools, Vadere offers a visually intuitive interface that simplifies the setup of diverse scenarios. Moreover, it automatically saves simulation outputs, enhancing usability and efficiency.

There are two primary methods to access Vadere. The first method involves downloading the compressed package from the official website and launching the "vadere-gui.jar" to open the graphical user interface (GUI). Alternatively, users can load Vadere through an Integrated Development Environment (IDE). This requires downloading the "master branch" version of Vadere, installing Java 11 or later, and executing the main function within the IDE to start the program.

For our initial task, we are to employ the Optimal Steps Model within Vadere to simulate three distinct scenarios: RiMEA scenarios 1 and 6, and the "chicken test."

1.1   RiMEA scenarios 1

Figure 1: RiMEA scenarios 1 in Exercise 1

RiMEA scenario 1 defines a pathway that is 40 meters long and 2 meters wide. The starting point for pedestrians is on one end, with the target endpoint directly opposite. With a set walking speed of 1.34 meters per second, pedestrians should complete this distance in a timeframe ranging from 26 to 34 seconds.

To initiate this scenario, select the Optimal Steps Model (OSM) by navigating to the "Model" menu and choosing "Load template" in the Vadere GUI.

Upon reviewing the simulation outcomes, as depicted in Figure 2, it is observed that Vadere's results closely align with those from Exercise 1, with transit times consistently falling within the expected 26 to 34-second range. A notable distinction, however, lies in the path trajectories between the two simulations. Vadere's visualizations reveal a nearly straight pedestrian path, whereas in Exercise 1, the pedestrian trajectory veers closer to the wall as they commence walking, as shown in Figure 1. This deviation is attributed to not accounting for the proximity to obstacles, resulting in a curved path.

(a) In the beginning

(b) Near the end

Figure 2: RiMEA scenarios 1 in Optimal Steps Model
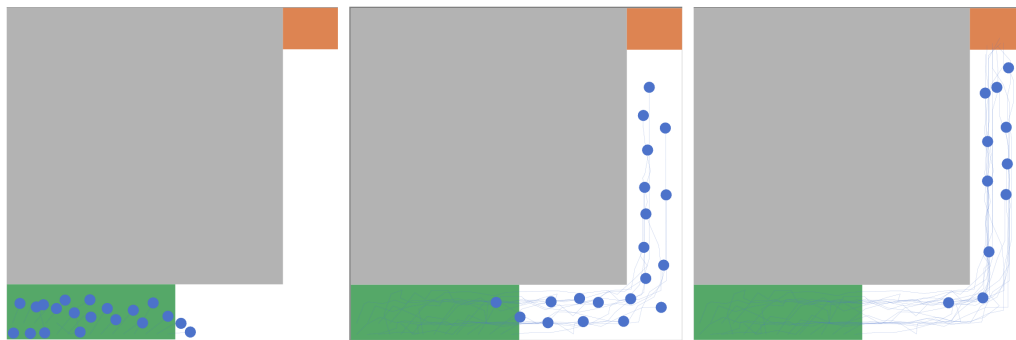
1.2   RiMEA scenarios 6

Figure 3: RiMEA scenarios 6 Optimal Steps Model

RiMEA Scenario 6 tasks 20 pedestrians with reaching a target located in the upper right corner of the space without crossing through a wall or turning left into the corner.

In this setup, the right and lower edges of the area, along with a square obstacle positioned in the upper left corner, are utilized to structure the scene as illustrated in Figure 3. We arranged 20 pedestrians in the green starting area of the corridor, spacing them equally. This arrangement results in varying distances for each pedestrian to the target. According to the figure, all pedestrians successfully completed the test without breaching the wall boundary.

The results from Vadere closely mirror those from Exercise 1, though there are slight differences in pedestrian movement. As depicted in Figure 4, pedestrians appear to walk closely against the wall and cluster towards the

target, seemingly disregarding the space between one another. Conversely, in Vadere's Optimal Steps Model (OSM), pedestrians navigate in a more orderly manner, often walking in pairs, which prevents crowding at the corners. This orderly movement mechanism in Vadere leads to longer simulation times compared to Exercise 1, as pedestrians take a lengthier route.
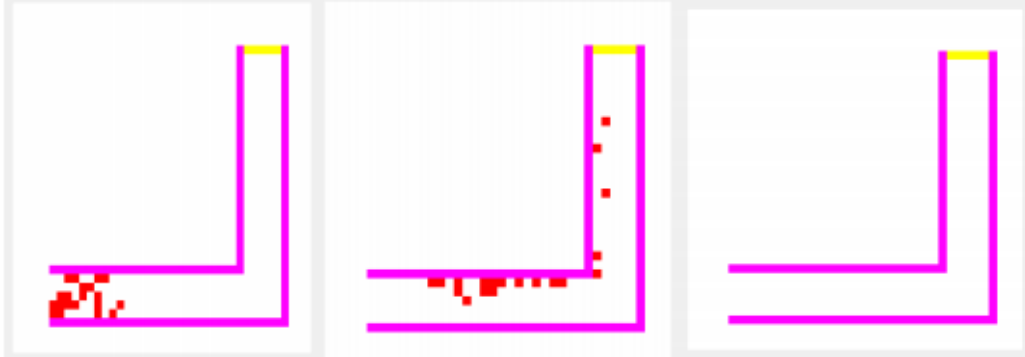

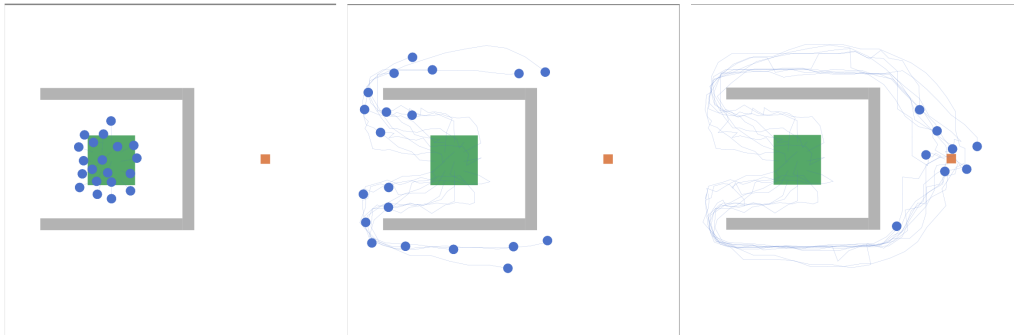Figure 4: RiMEA scenarios 6 in Exercise 1

1.3 Chicken Test

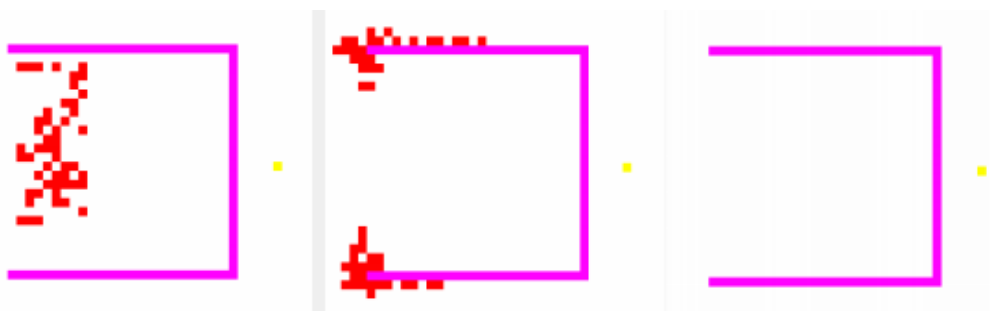Figure 5: Chicken Test in Optimal Steps Model

Figure 6: Chicken Test in Exercise 1

The "chicken test" scenario involves a U-shaped obstacle that disrupts the direct path between a group of pedestrians and their target, effectively trapping them and preventing straightforward access to the target.

In Vadere, we constructed a U-shaped obstacle and positioned the target above the center of the canvas. We placed three groups of pedestrians: inside, in the middle of, and outside the U-shaped obstacle. This setup allows for an insightful observation of the pedestrians' movement trajectories.

As depicted in Figure 5, all pedestrians successfully reached their target, aligning with the results from Exercise 1. However, a noticeable difference from Exercise 1 is how pedestrians in the simulation consciously maintain a distance from the U-shaped obstacle. Those within the obstacle circumvent the wall and move towards the target in an orderly fashion. Similarly, pedestrians in the middle and outer parts of the obstacle navigate around it, rather than taking a direct path that could lead to them getting stuck.

By ensuring a buffer between pedestrians and obstacles, Vadere's simulation presents a more realistic depiction of crowd movement, though it results in longer simulation times. In contrast, when the Dijkstra or FMM algorithm is applied in Exercise 1, as shown in Figure 6, pedestrians tend to crowd against the wall, particularly at corners, resembling the movement of a swarm of bees or a highly congested crowd.

1.4 comparisons between cellular automaton implementation and optimal step models in Vadere

In our user interface, we incorporated only basic buttons with simple functionalities. In contrast, Vadere offers a broader range of features. Particularly in user-defined scenarios, Vadere's approach is more optimized and user-friendly.

Regarding the modeling of trajectories, Vadere consistently accounts for the distance between pedestrians as well as between pedestrians and obstacles, resulting in trajectories that differ significantly from ours.

A key distinction between the two simulations is the level of visualization provided. Vadere offers extensive visualization options, including tracking trajectories, movement directions, various groupings, and step lengths. Our implementation, however, is limited to the basic movement of pedestrians and offers considerably less visual detail compared to Vadere.

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

In Task 1, we utilized Optimal Steps Models to simulate three distinct scenarios. For Task 2, we are required to employ the Social Force Model (SFM) and the Gradient Navigation Model (GNM) to simulate these same scenarios and then perform a comparative analysis of the results.

To access the SFM and GNM, navigate to the "Model" menu and select the "Load template" tab within the GUI.

Optimal Steps Model (OSM):

This model typically calculates the most efficient path to the target, minimizing walking time and avoiding obstacles straightforwardly.It usually shows clear, direct routes toward the target with minimal deviation unless obstructed by barriers.

Social Force Model (SFM):

This model simulates pedestrian dynamics by incorporating social and physical forces, like personal space and group dynamics, influencing movement.Trajectories might appear more organic and less direct compared to OSM, reflecting real-life social behaviors such as avoiding close proximity to others.

Gradient Navigation Model (GNM):

This model uses potential fields where each point in space has a potential value that guides the pedestrian toward the target while avoiding high potential (obstacle) areas. Paths can be curvilinear, adapting dynamically to the changing "potential landscape" created by obstacles and other pedestrians.

2.1 RiMEA scenarios 1

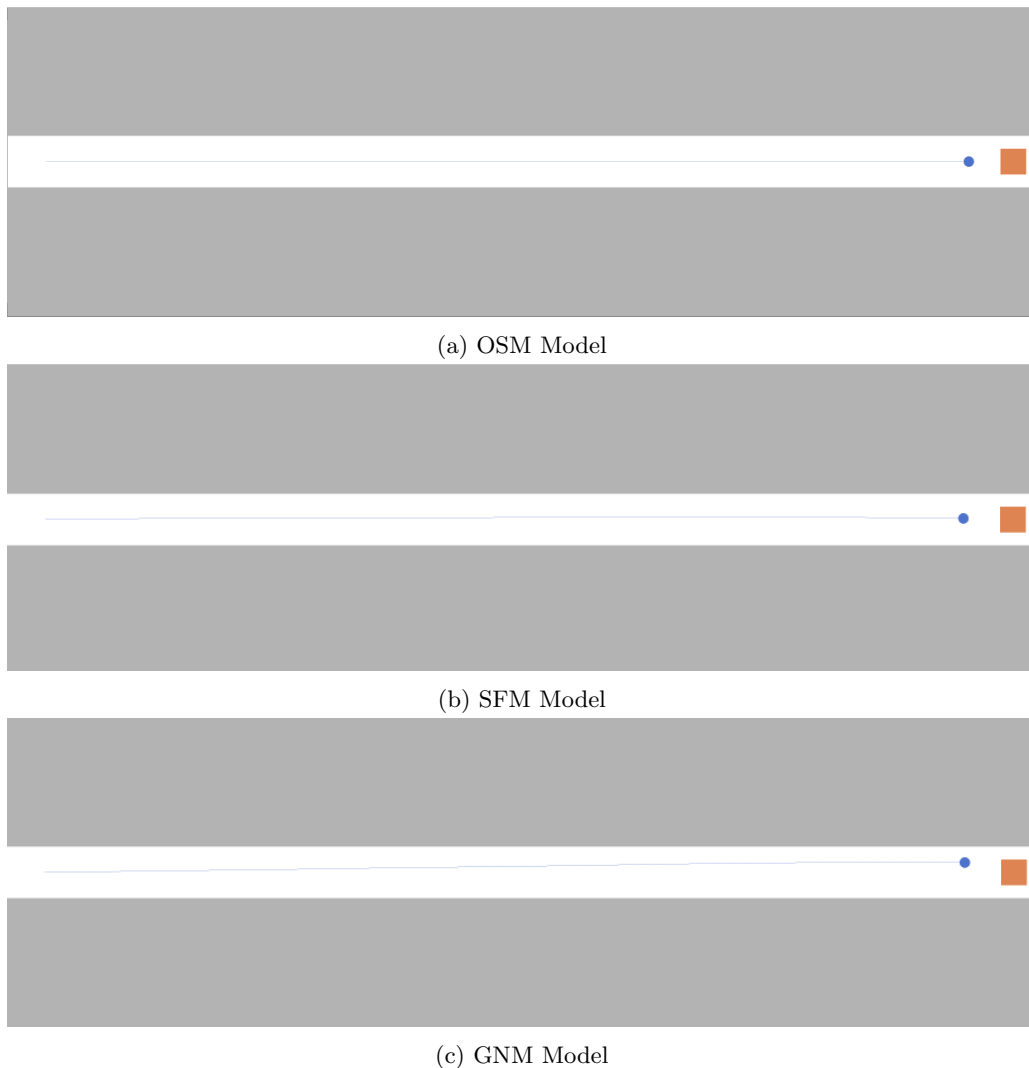(a) OSM Model

(b) SFM Model

(c) GNM Model

Figure 7: RiMEA scenarios 1 in different model

In the straightforward scenario of a pedestrian crossing a straight corridor, the outcomes produced by the three models are quite similar (Figure 7), with distinctions primarily in the trajectories. The Optimal Steps Model (OSM) maintains a straight line. The Social Force Model (SFM) exhibits a trajectory that is slightly inclined. In contrast, the Gradient Navigation Model (GNM) distinctly displays a trajectory where the pedestrian veers towards the wall, creating a clear diagonal path.

2.2 RiMEA scenarios 6

In Figure 8, for the Social Force Model, it's evident that pedestrians display less concern for maintaining distance from each other and the wall, with two or three individuals able to walk abreast. This model achieves the quickest transit time among the three.

In the case of the Gradient Navigation Model, pedestrians tend to form a single file, navigating turns and moving orderly close to the wall. If the leading pedestrian blocks the path around a curve, those following adjust by reducing speed and waiting, adhering to the leader's trajectory. The resulting paths are fewer and more defined, resulting in the longest transit time of 35.2 seconds.

Compared to the Optimal Steps Model, the trajectory in the Social Force Model (SFM) is smoother with fewer deviations, indicating a shorter travel distance for pedestrians. Thus, OSM takes slightly longer than SFM. OSM is faster than GNM because, in GNM, pedestrians who are blocked slow down and wait, whereas in OSM, pedestrians opt for alternative routes.

(a) OSM Model

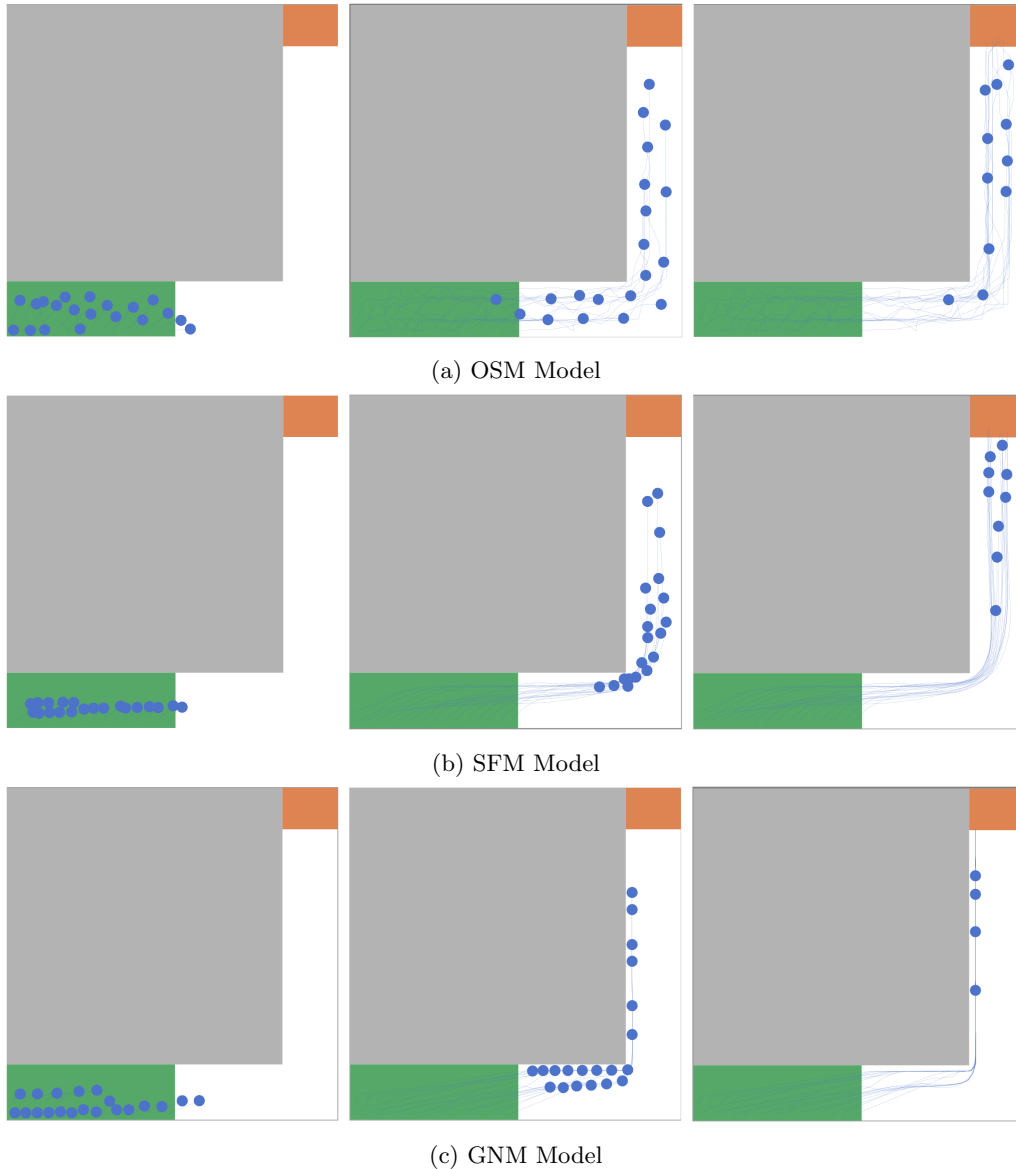(b) SFM Model

(c) GNM Model

Figure 8: RiMEA scenarios 6 in different model

2.3   Chicken Test

As shown in Figure 9, in the Social Force Model, which results in the shortest travel time, the trajectories are smooth with minimal corners, allowing pedestrians to take the most direct path. This model ensures a consistent distance is maintained both among pedestrians and between pedestrians and walls.

For the Gradient Navigation Mode, the trajectories are less frequent and primarily consist of straight lines. Pedestrians are observed lining up sequentially to navigate around the U-shaped obstacle and head towards the target. Notably, the paths between the obstacle and the target form almost two diagonal straight lines, distinctly different from other models.

In comparison, both the Social Force Model (SFM) and the Optimal Steps Model (OSM) offer a variety of pedestrian trajectories, indicating more dynamic route adaptations within these models.

During the "chicken test", the completion times recorded for the Social Force Model (SFM), Gradient Navigation Model (GNM), and Optimal Steps Model (OSM) were 22, 24.8, and 26.5 seconds, respectively. A detailed examination of figure 5 reveals that pedestrians in the OSM, maintaining their required distances, paused at the exit of the U-shaped obstacle to allow the person ahead to clear the area, resulting in congestion. This delay is likely the reason for OSM's comparatively slower performance. Despite OSM being the slowest, all three models successfully completed the "chicken test" with relatively similar times, showing only minor variations in duration compared to scenario 6.

(a) OSM Model

(b) SFM Model

(c) GNM Model

Figure 9: Chicken Test in different model

2.4 Summary

The analysis and images suggest that the choice of model has a significant impact on the distribution of pedestrian trajectories. Although the directions in which pedestrians move appear similar across the three

models, their actual trajectory paths and the time they take to complete them vary notably.

Both the Optimal Steps Model (OSM) and the Social Force Model (SFM) exhibit comparable times and movement patterns. However, OSM places a greater emphasis on maintaining spacing between pedestrians. It opts to increase travel time and take longer routes to ensure more personal space, whereas SFM prioritizes reaching the target more directly.

The Gradient Navigation Model (GNM) enforces more rigid path delineation and restricts pedestrians from deviating to alternative routes for avoidance, which tends to result in more congestion and generally longer completion times compared to the other two models.

Report on task 3, Using the console interface from Vadere

This task focuses on using the Vadere simulation software's console interface to modify scenario files and programmatically add pedestrians. The console version facilitates the integration of Vadere as a "black box" within external software environments, such as Python scripts.

The main goal of this project is to add pedestrians to scenario files using programming techniques and to analyze the outcomes by comparing the results from the console version with those from the graphical user interface. We will use the "corner scenario" as the baseline for our experiments. The development work will involve writing code to place pedestrians at designated locations within the scenario.

By comparing the execution times, we aim to evaluate the differences and similarities in how quickly the programmatically added pedestrians reach their target compared to those originally present in the scenario. This comparison will help us understand the effectiveness of using the console interface for scenario manipulation and pedestrian dynamics analysis.

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{\_}
```

We input the result in the folder:

```
mlicms24ex2-groupf\task3\console
```

By running the GUI again, we find that the results have not changed.

3.2   Adding pedestrians

To modify the Vadere simulation scenario file programmatically in order to add pedestrian, we created a Python script:

```
mlicms24ex2-groupf\task3\add_pedestrian\add_pedestrian.py
```

Here's a concise overview of the implementation details:

- **Predefined pedestrian template**

  - We formulated a JSON template, representing the attributes of the pedestrian intended for addition to the scenario.

- **Adding Pedestrian function**

  - The heart of the implementation lies in a function:
    * It reads the existing scenario file.
    * Picked a unique ID for the newly introduced pedestrian.
    * Updates the predefined pedestrian template with the ID and user-provided inputs.
    * Appends the modified pedestrian template to the 'dynamicElements' list within the scenario file.

- **Setting Pedestrian Parameters**

  - During script execution through the Command Line, users specify parameters such as the scenario path to be modified, the initial position of the pedestrian to be added, their speed, and the target ID.

The following is the workflow for the task. For the sake of convenience, these commands has been recorded in:

```
mlicms24ex2-groupf\task3\add_pedestrian\task3.ipynb
```

- **add pedestrian to RiMEA scenario 6**

```
!python add_pedestrian.py -s
"..mlicms24ex2-groupf\task3\add_pedestrian\scenario_6.scenario"
--x 11.5 --y 1.5 --targetID 3 --speed 0.5
```

- **call vadere-console.jar with the newly modified scenario file.**

```
!java -jar ..\\vadere-console.jar scenario-run
  --scenario-file "..mlicms24ex2-groupf\task3\add_pedestrian\scenario_6.scenario"
--output-dir "..mlicms24ex2-groupf\task3\add_pedestrian"
```
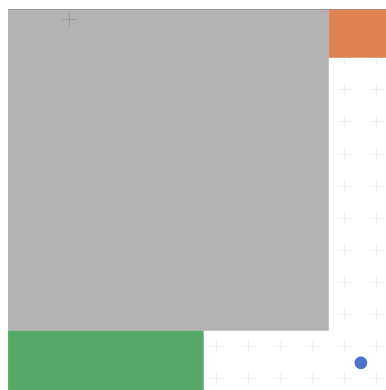
3.3 Results and Discussion

Figure 10: Added Pedestrian Scenario

By recoding, we successfully added pedestrians at the position of X 11.5 Y1.5 in scenario 6, as shown in Figure 10.
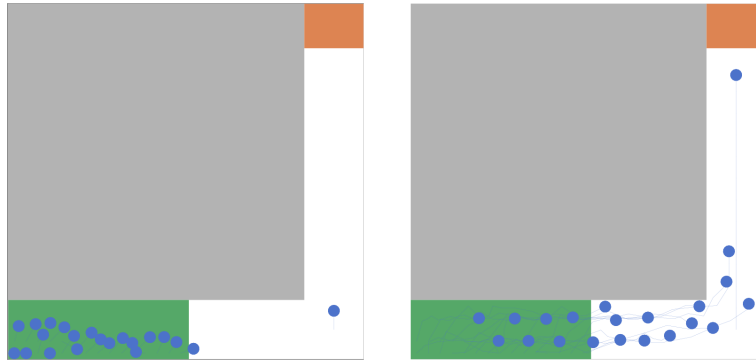

Figure 11: Added Pedestrian Scenario begin and near end

As shown in Figure 11, after executing the modified scenario file through the program, the newly added pedestrian reached the target in 6.8 seconds, while the average travel time for the original source pedestrians was around 20 seconds. This outcome was anticipated because the added pedestrian was positioned closer to the target, bypassing the crowd effects that typically slow down movement.

No discrepancies were observed in the output files between the graphical user interface (GUI) and the simulations conducted with the programmatically modified scenario file. This consistency confirms that the final inputs to the simulation software were identical, ensuring that the comparison of results is based on equivalent scenarios.

Report on task 4, Integrating a new model

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

First, we cloned the Vadere repository and installed IntelliJ as our development environment. After transferring the necessary files to their appropriate directories, we successfully integrated the Susceptible-Infected-Recovered (SIR) model into Vadere. We were then able to execute Vadere from IntelliJ using the newly integrated SIR model. This setup allows for simulation and analysis of pedestrian dynamics within the context of infectious disease spread, utilizing the capabilities of the SIR model directly from the integrated development environment (IDE).

Description of the SIR Model:

- In every Java project, besides the primary operations, there are essential initializer, getter, and setter functions that facilitate the running of the project.

- In the model, the `preLoop` function is initially called, which then invokes the `initializeGroupsOfInitial Pedestrians` function. This function returns a list of Dynamic Element-Containers of type Pedestrian, with 'n' pedestrians being randomly initialized.

- During each time step, the `update` function is executed to refresh the current status of each pedestrian. This function begins by checking the positions of all pedestrians, categorizing them into groups such as INFECTED or REMOVED. It then iterates through neighboring pedestrians, setting their status to infected if they are in close proximity.

- The `getFreeGroupId` function assigns a groupId to an entry in the dictionary named `getGroupsById`. At the start of the simulation, this function is used to increase the count of infected and recovered individuals.

- The `assignToGroup` function takes a pedestrian's own instance as a parameter. If the pedestrian does not already belong to a group, a new groupId is randomly assigned based on the specified initial conditions. If the pedestrian is already in a group, it is reassigned to the new groupId provided by the function and removed from its current group.
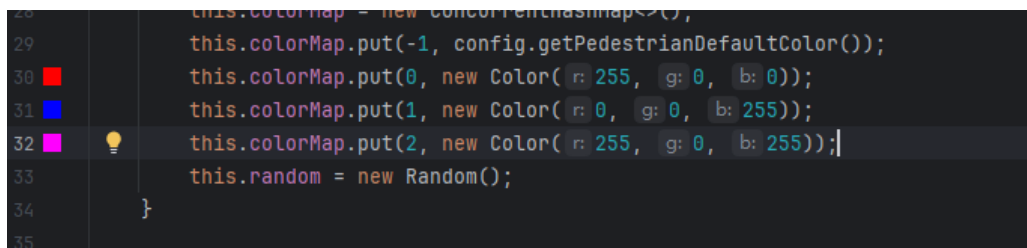
4.2   Description of the output processor

In this section, we have opted to use our custom output processor, the `SecondGroupIDProcessor`, instead of the standard `FootStepGroupIDProcessor`, which is located in the same directory as the other output processors. This processor captures the pedestrian ID and group ID at each time-step, ensuring that even non-moving pedestrians are accounted for, unlike the footstep processor which provides a less comprehensive output. Initially, we could have designed a program to interpret outputs from the footstep processor and retain the group ID from one time-step to the next, but this approach was not considered at the time.

For visualization, we employ a Python notebook that reads the output file, extracts the group IDs at each time-step, and then displays the percentages of these group IDs. This process is managed through the `visualizeSIR.ipynb` notebook, which produces charts that monitor the shifts within the SIR groups over time.

4.3   Description of the output processor

To assign colors to the pedestrians based on their group, we need to update the `colorMap` in the `SimulationModel`.java class. Here, we map each group ID to a specific color. We have designated the following colors to represent each group ID:

- 0= RED is for infective

- 1= BLUE is for Susceptible

- 2= PINK is for Recovered


```
this.colorMap = new ConcurrentHashMap<>();
this.colorMap.put(-1, config.getPedestrianDefaultColor());
this.colorMap.put(0, new Color( r: 255,  g: 0,  b: 0));
this.colorMap.put(1, new Color( r: 0,  g: 0,  b: 255));
this.colorMap.put(2, new Color( r: 255,  g: 0,  b: 255));
this.random = new Random();
}
```

Figure 12: Source Code for changing colors

We opted not to use green or white to represent susceptible individuals, as the pedestrian spawning area is already visualized in green, and the background color is white. To ensure that pedestrians are visualized according to their group color during the simulation, we need to configure the `DefaultSimulationConfig.agentColoring` setting to `AgentColoring`.GROUP. It's important to note that our customization through the `SecondGroupIDProcessor` currently supports only the visualization of groups during the simulation. Post-simulation visualization of groups has not been implemented in this version of our code.

4.4   Pseudocode for improving the efficiency of `LinkedGridCells`

The update method is responsible for checking the positions of all pedestrians and updating their health status based on the SIR (Susceptible-Infected-Removed) model. Here's a detailed explanation of its implementation:

Retrieve Pedestrians: The method starts by retrieving all the pedestrians from the environment. Check if there are any Pedestrians: It checks if there are any pedestrians to update. If there are no pedestrians, the method exits early.

Initialize `LinkedCellsGrid`: It initializes a `LinkedCellsGrid` to manage spatial queries more efficiently. The grid divides the environment into cells, with the cell size based on the maximum infection distance. Populate the Grid: All pedestrians are added to the `LinkedCellsGrid`, so their positions are now efficiently managed within the grid.

Update Health Status of Each Pedestrian: The method iterates over each pedestrian to update their health status. Retrieve Neighbors: For each pedestrian, the method retrieves a list of neighbors within the infection maximum distance using the `LinkedCellsGrid`.

Check and Update Infection Status: The method iterates over each neighbor to check if they are within the infection distance and if a random check based on the infection rate succeeds. If both conditions are met, the pedestrian's status is updated from susceptible to infected.

Check and Update Recovery Status: The method checks if the pedestrian is infected and if a random check based on the recovery rate succeeds. If both conditions are met, the pedestrian's status is updated from infected to removed. Summary

The update method efficiently updates the health status of pedestrians by: Using `LinkedCellsGrid` to optimize neighbor lookup.

Iterating through each pedestrian to check and update their infection and recovery status based on proximity to infected individuals and predefined infection and recovery rates.This approach ensures that the simulation runs efficiently, even with a large number of pedestrians.

```java
public void update(final double simTimeInSec) {
// check the positions of all pedestrians and switch groups to INFECTED (or REMOVED).
DynamicElementContainer<Pedestrian> c = topography.getPedestrianDynamicElements();

if (c.getElements().size() > 0) {
    // Initialize LinkedCellsGrid to increase neighbor lookup efficiency
    LinkedCellsGrid<Pedestrian> linkedCellsGrid = new LinkedCellsGrid<Pedestrian>(
            topography.getBounds().x,
            topography.getBounds().y,
            topography.getBounds().width,
            topography.getBounds().height,
            attributesSIRG.getInfectionMaxDistance()); // using infection max distance as the grid size if relevant

    // Populate the grid with pedestrians
    for (Pedestrian p : c.getElements()) {
        linkedCellsGrid.addObject(p);
    }

    for (Pedestrian p : c.getElements()) {
        // Retrieve neighbors from the grid within the infection maximum distance
        List<Pedestrian> neighbors = linkedCellsGrid.getObjects(p.getPosition(), attributesSIRG.getInfectionMaxDistance());

        for (Pedestrian p_neighbor : neighbors) {
            if (p == p_neighbor || getGroup(p_neighbor).getID() != SIRType.ID_INFECTED.ordinal()) {
                continue;
            }
            double dist = p.getPosition().distance(p_neighbor.getPosition());
            if (dist < attributesSIRG.getInfectionMaxDistance() &&
                    this.random.nextDouble() < attributesSIRG.getInfectionRate()) {
                SIRGroup g = getGroup(p);
                if (g.getID() == SIRType.ID_SUSCEPTIBLE.ordinal()) {
                    elementRemoved(p);
                    assignToGroup(p, SIRType.ID_INFECTED.ordinal());
                }
            }
        }
    }
```

Figure 13: update coad

4.5   Test Cases

4.5.1   Static Scenario with Default Settings

Following the task requirements and aligning with the illustration provided in the exercise sheet, we created a static scenario featuring 1000 pedestrians. We set the Boolean variable absorbing to "false" to denote that the target area does not absorb pedestrians. The target is positioned at coordinates (1, 1) and measures 28 meters in both height and width.

For the source area, we configured its parameters with `eventPositionRandom` set to "true" and `eventPositionFree Space` set to "false". It is located at (2, 2) and spans 24 meters both in height and width, which facilitates clear observation of the spatial relationship between the source and the target. We specified the `spawnNumber` parameter to 1000 to ensure the generation of a total of 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 Figure 14:

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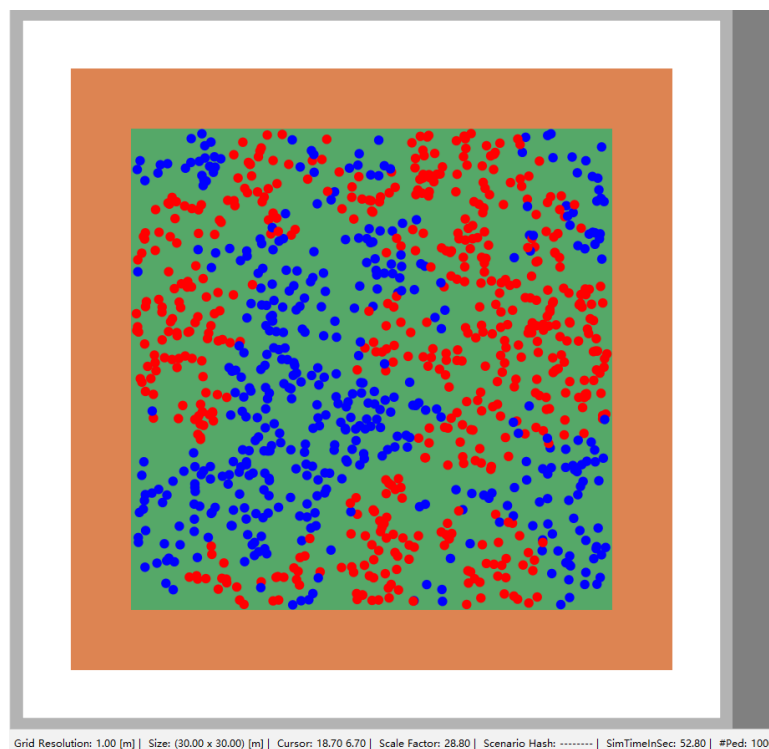

Grid Resolution: 1.00 [m] | Size: (30.00 x 30.00) [m] | Cursor: 18.70 6.70 | Scale Factor: 28.80 | Scenario Hash: -------- | SimTimeInSec: 52.80 | #Ped: 1000

Figure 14: infection rate: 0.01 at time 52.8s

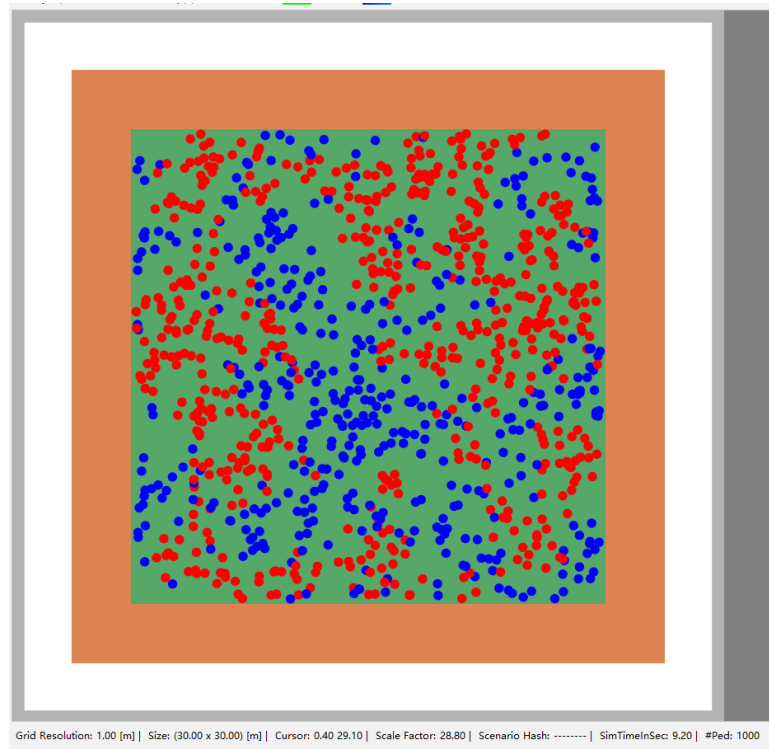### 4.5.2   Static Scenario with Increased Infection Rate

Figure 15: infection rate: 0.03 at time 9.2s

The only change here we did is we increased the infection rate from 0.01 to 0.03. However, significant changes happened in the time half of the pedestrians were infected. There is Figure 15 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.
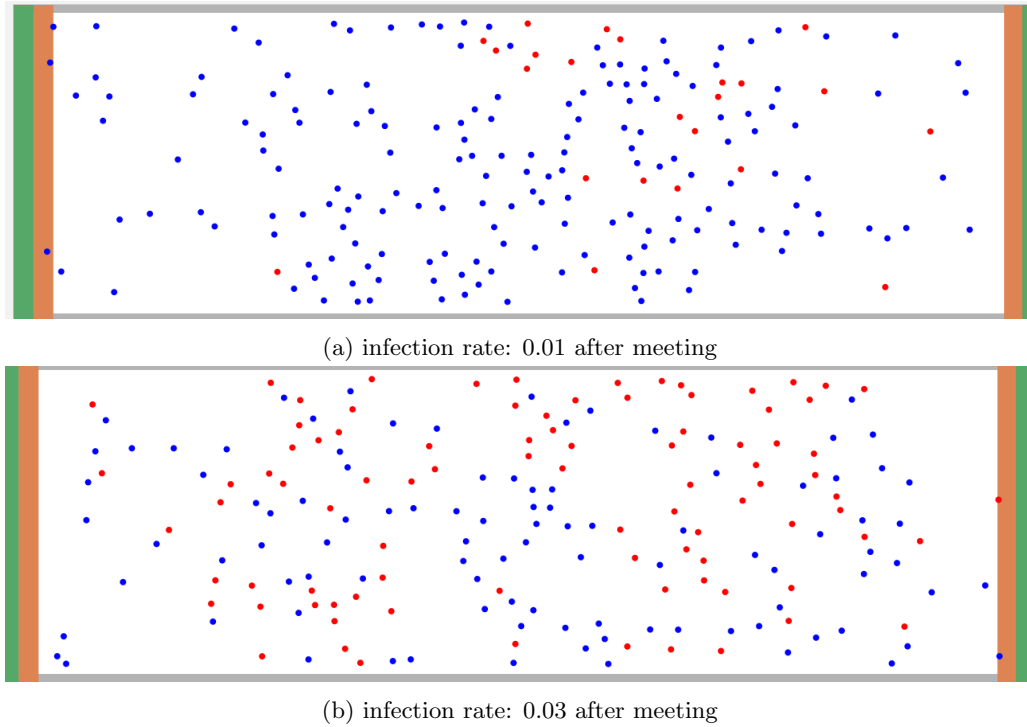
4.5.3 Corridor Test

(a) infection rate: 0.01 after meeting

(b) infection rate: 0.03 after meeting

Figure 16: In-simulation Look

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 15 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

In this application, adjusting the step size causes pedestrians within the simulation to potentially switch groups, effectively altering the simulation's visual representation. Our proposal involves calibrating the relationship between the infection rate and the time step based on real-world data. Unfortunately, since we do not have access to actual data from real simulations, we will use default settings as a benchmark.

We recommend setting the infection rate at 0.01 and the time step length at 0.4 seconds. Under this proposal, the product of the infection rate and the time step length should consistently equal 0.04 across all simulations to maintain a standard measure of infection likelihood. This multiplication approach is used because a shorter time step length increases the frequency of updates during the simulation, potentially raising the likelihood of infection per unit time. Conversely, lowering the infection rate decreases the probability that a pedestrian will become infected, balancing the dynamics between simulation granularity and infection risk.

4.7 Possible Extensions

To enhance the realism of our simulation, we can implement several features:

- Pedestrians can be categorized into different groups based on factors such as pre-existing health conditions

or age. This grouping allows for varying probabilities of being removed from the simulation upon infection, reflecting real-life scenarios where certain individuals might be at higher risk of severe outcomes.

- Adding the dimension of vaccination would provide a more comprehensive model of recovery. Unlike the current model where recovery only follows an infection, real-life scenarios include preventative measures like vaccines. The effectiveness of the vaccine could be modeled as a function that increases over time, representing improvements in vaccine efficacy or coverage.

- To simulate responsible behavior adjustments over time, interactive controls such as buttons could be introduced. These would modify the infection rate dynamically within the simulation. For instance, pressing a button could simulate an increase in mask usage or adherence to social distancing guidelines, thereby reducing the infection rate. This change would be visualized by increasing the distance between pedestrians in the simulation.

Each of these enhancements aims to mirror real-world complexities and human behaviors, making the simulation a more valuable tool for predicting and understanding the dynamics of pedestrian movement and infection transmission.

Report on task 5, Analysis and visualization of results

5.1 Adding recovered state

In the enhancement of our simulation, we've introduced a "recovered" state to allow for the progression from "infected" to "recovered." This addition means that only pedestrians who have been "infected" can transition to "recovered"; directly going from "susceptible" to "recovered" is not possible.

Each time step offers a chance for "infected" pedestrians to recover, based on a defined probability termed the `RecoveryRate`. This rate of recovery is independent of the pedestrian's location or the proximity of other pedestrians. Once a pedestrian transitions to the "recovered" state, they are no longer able to transmit the infection to others, nor can they be re-infected themselves. As shown in Figure

```java
// Let infected pedestrians recover with a given probability
SIRGroup g = getGroup(p);
if (g.getID() == SIRType.ID_INFECTED.ordinal() &&
        this.random.nextDouble() < attributesSIRG.getRecoveryRate()) {
    elementRemoved(p);
    assignToGroup(p, SIRType.ID_REMOVED.ordinal()); // Assuming REMOVED is the correct state
}
```

Figure 17: added code

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 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 Test 1: Static scenario of 1000 pedestrians

These two sets of Figures 18 and 19 each display the results of an SIR model simulation with 1000 pedestrians in a static setting, specifically under different infection and same recovery rate 0.01.
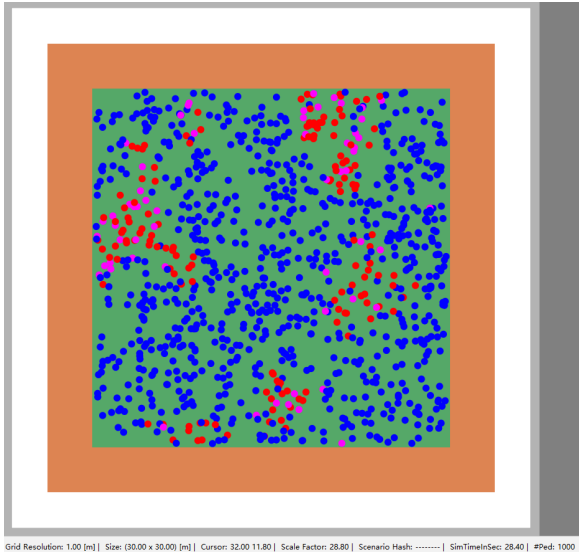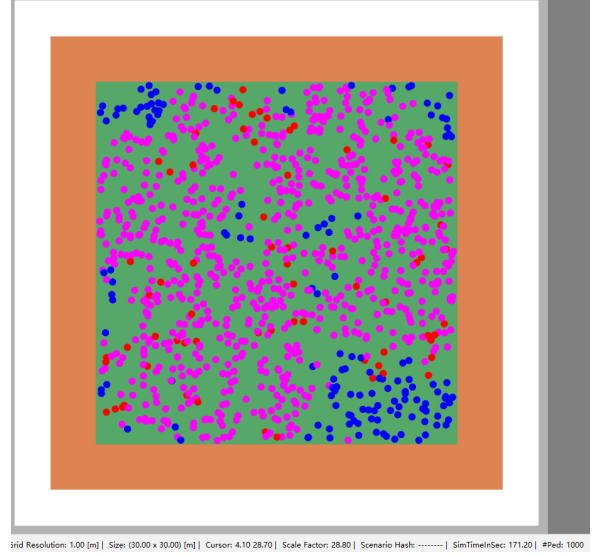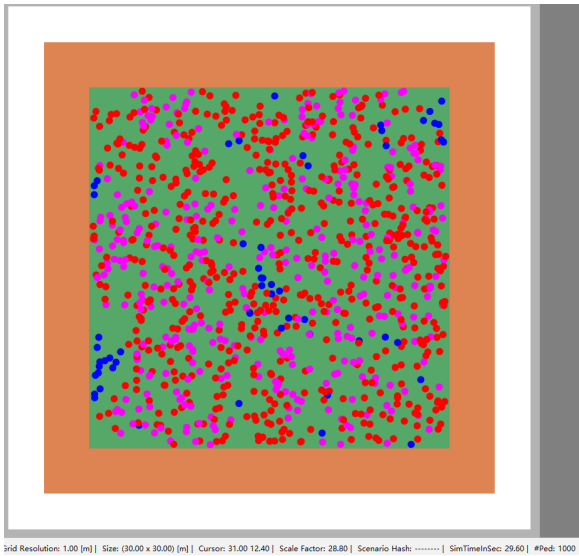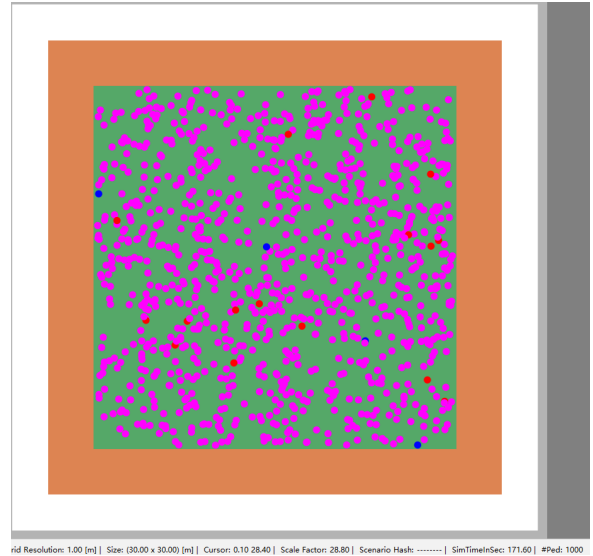
(a) at start

(b) in the end

Figure 18: infection rate: 0.01 recovery rate: 0.01

(a) at start

(b) in the end

Figure 19: infection rate: 0.03 recovery rate: 0.01

These two sets of Figures 18 and 19 each display the results of an SIR model simulation with 1000 pedestrians in a static setting, specifically under different infection and same recovery rate 0.01.

As shown in Figure 18, the infection rate is 0.01. At the start state, shows a majority of blue dots (susceptibles) and a few red dots (infected), indicating a limited number of initial infections. And in the end, a large number of pink dots (recovered) dominate the image, indicating that the vast majority of pedestrians have recovered from the infection by the end of the simulation. A significant reduction in red dots (infected), suggesting that the infection has been controlled, with most infected transitioning to recovered. The relative scarcity of blue dots (susceptibles) implies that most uninfected individuals eventually went through the infection and recovery process.

As shown in Figure 19, the infection rate increase to 0.03. At the start state, more red dots (infected) compared to the initial state in Figure 18, indicating a more widespread infection. The number of blue dots (susceptibles) remains high, but the proportion of infected has increased, consistent with the higher infection rate. And in the end, Nearly the entire image is covered with pink dots (recovered), with almost no red dots (infected), showing that almost all infected have recovered. This indicates that the infection spread very quickly, but given the relatively low recovery rate, the overall recovery process is still somewhat slow.

After comparison of Figures 18 and 19, the spread of infection in Figure 19 is faster due to the higher infection rate (0.03 compared to 0.01), which caused more susceptibles to become infected in a shorter period. Although the recovery rate is the same in both figures (0.01), Figure 19 shows virtually no red dots at the end, indicating that the infected individuals quickly moved into the recovery phase even under a higher infection rate. In Figure 19, virtually no blue dots are visible at the end, reflecting a more intense virus transmission process where almost all susceptibles went through the infection and recovery process.

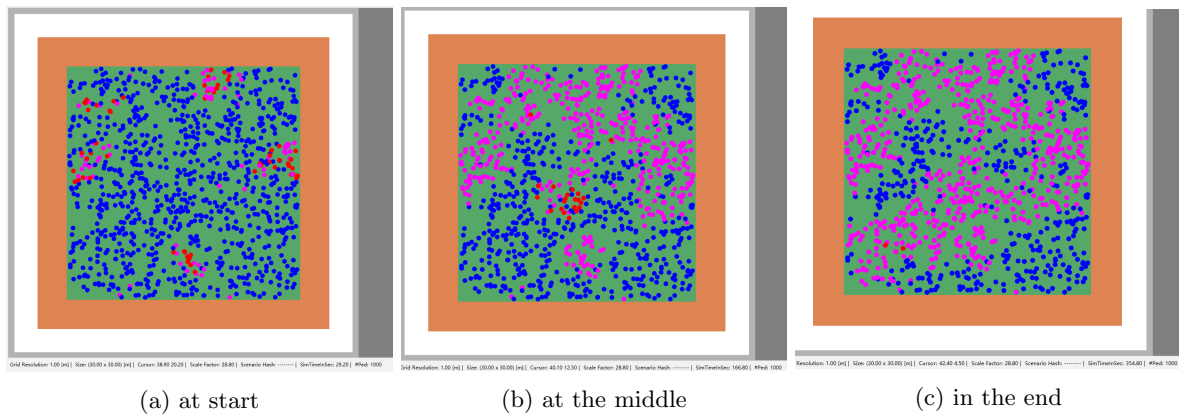## 5.3　Test 2: Experiment with the infection and recovery rate

(a) at start　　　　　　　(b) at the middle　　　　　　　(c) in the end

Figure 20: infection rate: 0.01 recovery rate: 0.02
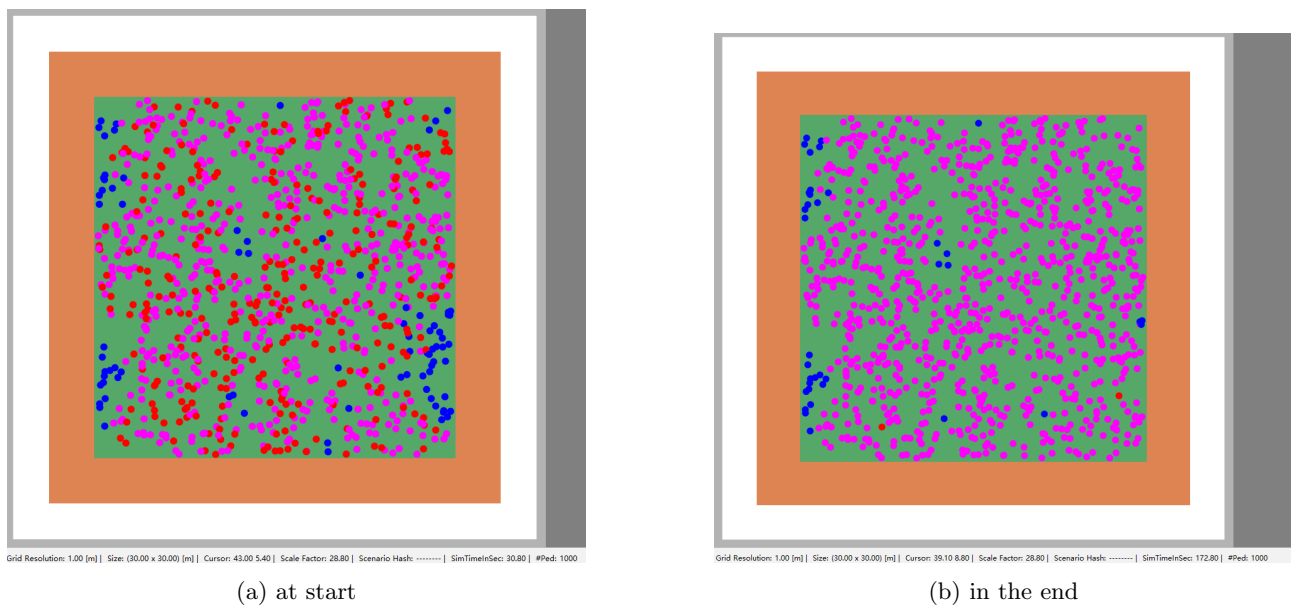

(a) at start　　　　　　　　　　　　　　　　(b) in the end

Figure 21: infection rate: 0.03 recovery rate: 0.02

In Test 2, we increase the recovery rate to 0.02 and keep the same infection rate as before. After simulate we can get Figure 20 and 21. we analyse below.

Figure 20 (Infection rate: 0.01, Recovery rate: 0.02)

- **(a) At start:** Majority of the individuals are blue (susceptible), with a few red (infected) dots.

- **(b) At the middle:** Increased number of pink (recovered) individuals indicates a faster recovery process, with slight increase in red dots suggesting ongoing transmission.

- **(c) In the end:** Predominantly pink (recovered) individuals, with very few red and blue dots, indicating effective control of the infection.

Figure 21 (Infection rate: 0.03, Recovery rate: 0.02)

- **(a) At start:** Similar to Figure 20(a), predominantly blue dots with scattered red dots.

- **(b) In the end:** Almost no blue dots, substantial increase in pink dots, and almost no red dots, showing a quick spread of the infection but also a quick recovery phase.

Comparison with Previous Figures 18 and 19

The speed of infection and recovery, the end state population distribution, and the impact of balanced vs. unbalanced rates are compared.

- Figures 18 and 19 demonstrated a slower progression of infection and recovery due to lower infection rates (0.01 and 0.03) and a constant recovery rate (0.01).

- Figures 20 and 21, particularly Figure 21, show a higher recovery rate (0.02) facilitating a quicker transition from susceptible to recovered, indicating effective epidemic management.

- The end state in Figures 20 and 21 shows almost entirely recovered individuals with virtually no remaining infection, unlike Figures 18 and 19 where infection persisted throughout the simulation.

These simulations illustrate how crucial the balance between infection and recovery rates is in managing epidemics. Effective epidemic management not only depends on slowing down the infection rate but also significantly on speeding up the recovery process.

5.4   Test 3: The Supermarket Scenario

In this part, 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 in the figure below.
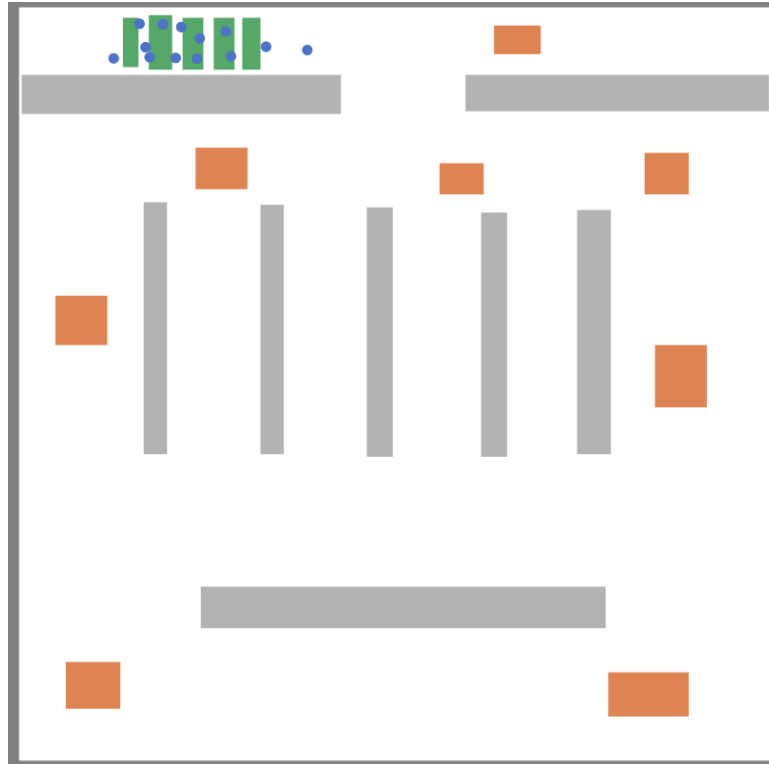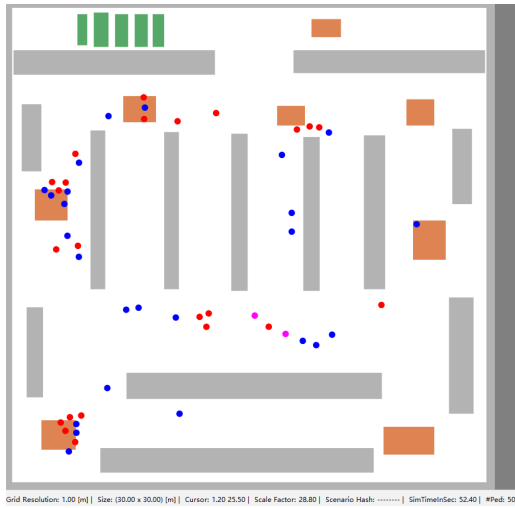
Figure 22: Supermarket Scenario

Details of Scenario:

- Dimensions- 30 x 30

- Number of customers to be spawned- 50 (10 from each source)

- Number of infected people at the start- 5

- Infection Rate- 0.01

- Recovery Rate- 0.001

- infection Max Distance- 1.0

- Number of sources- 5
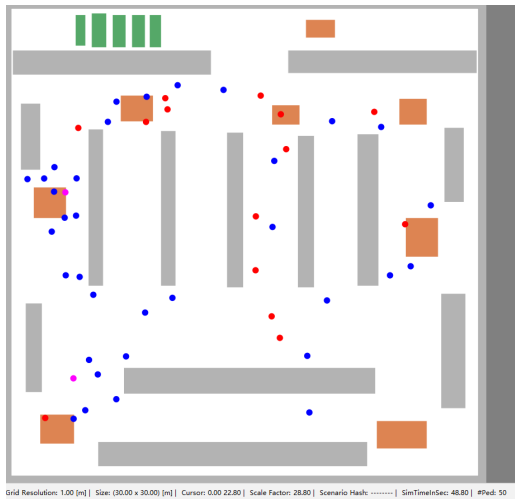
- Number of targets- 8

- Number of obstacles- 8

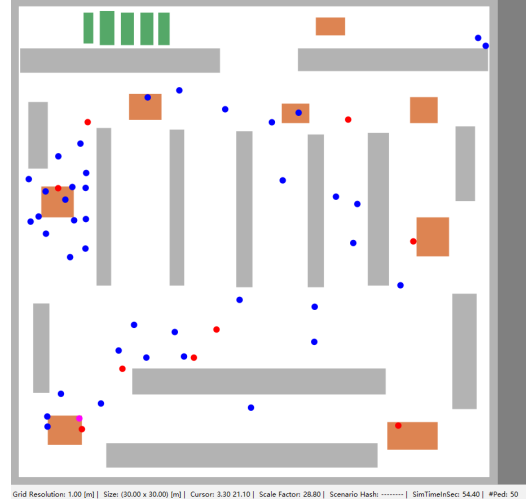**Explanation of Simulation in Figure 23:**

(a) pedPotentialPersonalSpaceWidth: 0.5
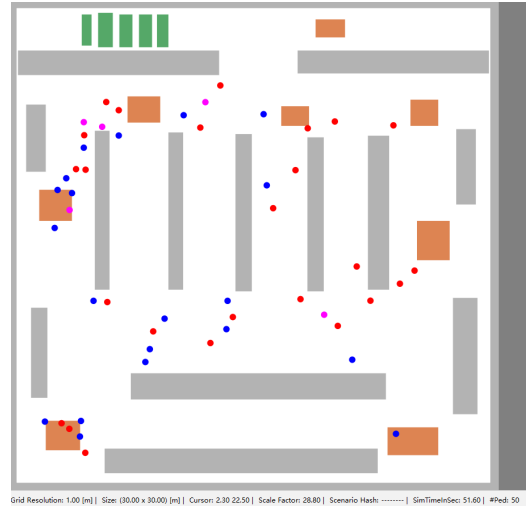
(b) pedPotentialPersonalSpaceWidth: 1.0

(c) pedPotentialPersonalSpaceWidth: 2.0

(d) pedPotentialPersonalSpaceWidth: 3.0

Figure 23: Different pedPotentialPersonalSpaceWidth in the same infection rate and recovery rate

Variation in Personal Space Width:

- In figure (a) where the personal space width is set to 0.5, there is a sparse distribution of red dots (infected individuals), suggesting that a smaller personal space setting might limit the spread of infection.

- In figure (b) with a personal space width of 1.0, the number of infections slightly increases, possibly indicating that increasing personal space allows more susceptibles to come into contact with infected individuals.

- Figures (c) and (d), with personal space widths of 2.0 and 3.0 respectively, show that as personal space increases, the spread of infections becomes more widespread, especially evident in figure (d) with significant spread.

Simulation of Infection Dynamics:

- Wider personal space settings, although possibly increasing physical distance between individuals, may also result in longer dwell times in specific areas, thereby increasing the chances of infection.

- The pattern of infection spread shows that larger settings might lead to more congregations of people, which in turn could increase the probability of transmitting the infection.

Impact of Supermarket Layout:

- The layout of the supermarket (e.g., the arrangement of shelves and aisles) also influences the movement of crowds and the spread of infection. The orange and gray blocks in the images show the placement of shelves, which may cause congestion in certain areas, thus increasing contact and chances of infection.

This type of simulation is very useful for understanding how different social distancing measures and environmental layouts affect the spread of disease. By adjusting the width of personal space, the effectiveness of disease control strategies under various settings can be studied, providing theoretical support for public health interventions.