

**Report for exercise 2 from group F**

Tasks addressed: 5  
Authors: Yang Cheng (03765398)  
Jianzhe Liu (03751196)  
Hao Chen (03764817)  
Pemba Sherpa (03760783)

Last compiled: 2023-05-15  
Source code: <https://github.com/Chuck00027/MLCMS-Exercise2.git>

The work on tasks was divided in the following way:

Yang Cheng (03765398) <b>Project lead</b>	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Jianzhe Liu (03751196)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Hao Chen (03764817)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Pemba Sherpa (03760783)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%

## Report on task TASK 1, Setting up the Vadere environment

The main goal of task 1 is to get familiar with Vadere and use it to simulate several RiMEA-scenarios in Exercise 1. So before showing the result of simulations, we can first make a general comparison between Vadere and the cellular automaton that we made.

First of all, the user interface of vadere is more perfect and more beautiful. The whole software is divided into several parts to provide different functions. Whether it is the convenience of creating projects or scenes, the visualization of scene files and output files, or the animation presentation of simulated screens, all aiming to enhance the user's experience. On the other hand, the cellular automaton we wrote has only basic functions, so there is still a lot of space for improvement.

By viewing its functions, vadere not only provides the function of loading pre-written files, but also enables users to directly draw in the software, and to customize various required parameters. At the same time, different models and attributes can also be loaded to change the simulation process.

Moreover, when it comes to the results of visualisation of a simulation, it is really convenient to have the results saved as outputs, so that they can be accessed again at any time. This reduces a lot of redundant file operations, whenever there is a change, all we need is one output file, instead of the whole program. What's more, the output files can also be loaded by other software, for example plotting software, to meet the needs of other work types.

### RiMEA scenario 1 (straight line)

After making comparisons, it's time to show the results of scenarios run by vadere. The first scenario is RiMEA scenario 1. it describes a 2 m wide and 40 m long corridor, with a pedestrian in one side and its target in the other one. The purpose of it is to prove that the pedestrian with a defined walking speed can cover the distance in the corresponding time period.



Figure 1: Simulation of RiMEA scenario 1

By looking at the results of the simulation (in Figure 1), the travel time was around 40 sim-time. Here we have tried several times to figure out, where the problem is, but didn't draw a certain conclusion. It's also to mention that by setting the RealtimeSimtime ratio to be 0.1, 50sim-time should be 5 seconds in real time, but the simulation result says it's only 3.93 sec, which means that with speed of 1.34, the pedestrian has taken around 3.1 sec to reach the target. The result seems to be in right numbers, but only ten times smaller.

**RiMEA scenario 6 (corner)**

In RiMEA scenario 6, we want a group of 20 pedestrians to move towards a corner, without passing through walls. The requirements of this scenario is: Both height and width of the scenario have to be 12 m. And the obstacle of the top left part of the scenario has to be 10x10 m. While the corridor should have a width of 2 m and length of 12 m on both horizontal and vertical part. At last, the 20 pedestrians should be uniformly distributed in the first 6 m of the horizontal part of the corridor.

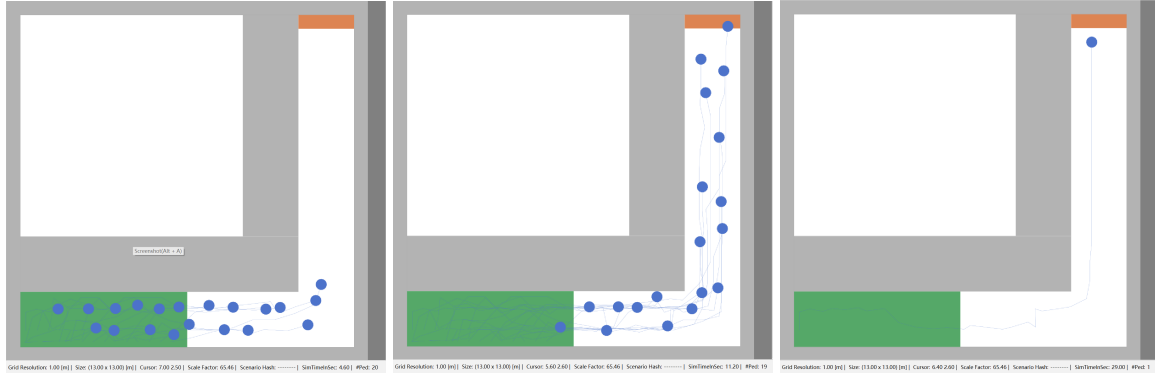


Figure 2: Simulation of RiMEA scenario 6

As shown in Figure 2, in order to show the process more clearly, we turned on the pedestrian trajectory display function during the simulation. It can be clearly seen from the figure that no pedestrian collides with the obstacle, every pedestrian has turned at the corner correctly and reached the target. This result is also same as the one in our cellular automation simulation.

**Chicken Test**

The last scenario we need to simulate is the chicken test scenario. It consists of several groups of pedestrians on one side and one target place on the other side, between them, there is a U-shaped obstacle. In this simulation, we want to test whether the pedestrians can avoid the obstacles and reach the target, instead of getting stuck in the obstacles.

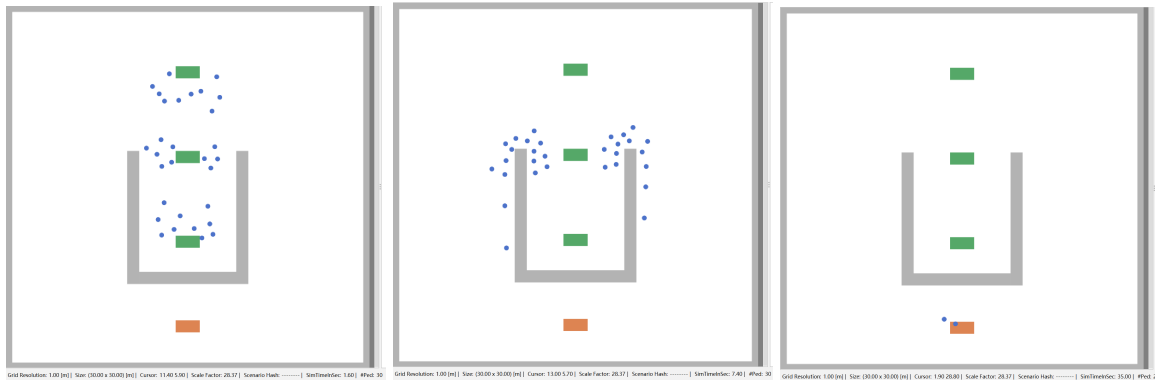


Figure 3: Simulation of Chicken Test

As shown in Figure 3, to better simulate this situation, we have created 3 different sources for pedestrians, which are located respectively inside the obstacle, at the entrance of the obstacle and outside the obstacle. As for the results, just as expected, there are no clear differences between vadere and cellular automation. But it's worth to mention that obstacle avoidance algorithm seems to be already implemented in vadere, but in cellular automation, it's a constriction that one need to set it manually, otherwise they can't avoid obstacles.

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

In last task, the three scenarios are run by a model named **Optimal Steps Model**. But in this task, we need to run them again, with other two different models: **Social Force Model** and **Gradient Navigation Model**. Then we compare their differences by analysing the simulation results.

### For RiMEA Scenario 1

Although it was mentioned before that the travel time of pedestrians in scenario1 is slightly different from the expected time, this does not affect the fact that we can test and compare this scenario under different models with the same parameters. As shown in Figure 4, the left one is run by SFM, and the right one is run by GNM. It's obvious to see that in SFM, the pedestrian is still moving in straight lines, while the one in GNM is heading downwards to the bottom part of the target. As for the time, both new models are about 4 simtime faster than OSM.



Figure 4: Different Models on RiMEA scenario 6

### For RiMEA Scenario 6 and Chicken Test

For the RiMEA Scenario 6 (in Figure 5) and Chicken Test (in Figure 6), You can't actually tell any difference from their final result, since they all reach the target. But one can still tell the changes from the way the pedestrians move, or to say, the path they choose. In the figures on left side, which belongs to SFM, people tend to have a special moving pattern among each other, since this model has more restrictions defining the parameter between people. Meanwhile, the figures on right side, which belongs to GNM, shows that, those pedestrians tend to find the same path to the target, since its algorithm aiming more on calculating the best path through iterations. Generally speaking, by only implementing simple algorithm, the pedestrian can reach the target in many ways, but the more complex the algorithm model is, the fewer exist those available paths.

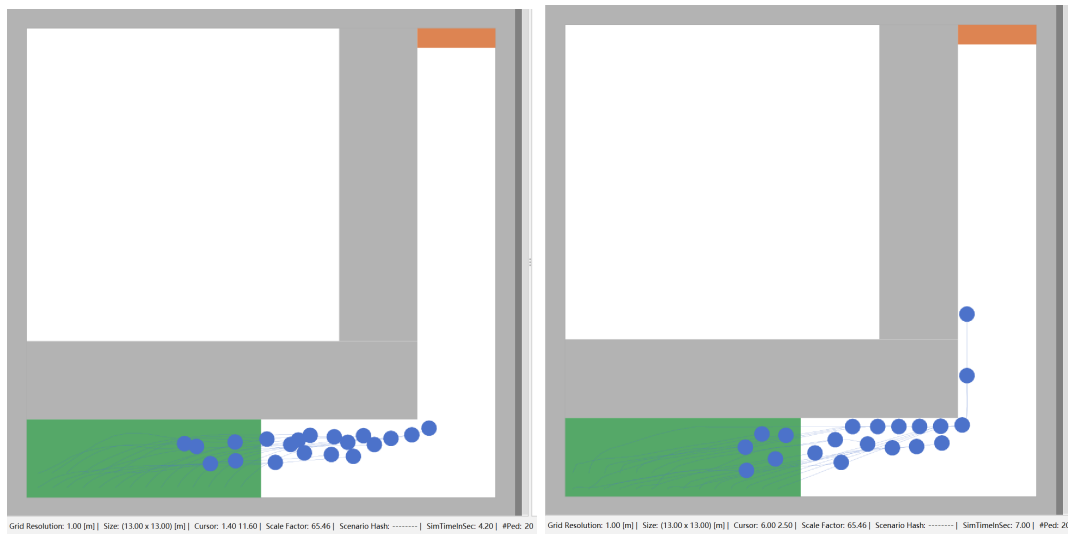


Figure 5: Different Models on RiMEA scenario 6

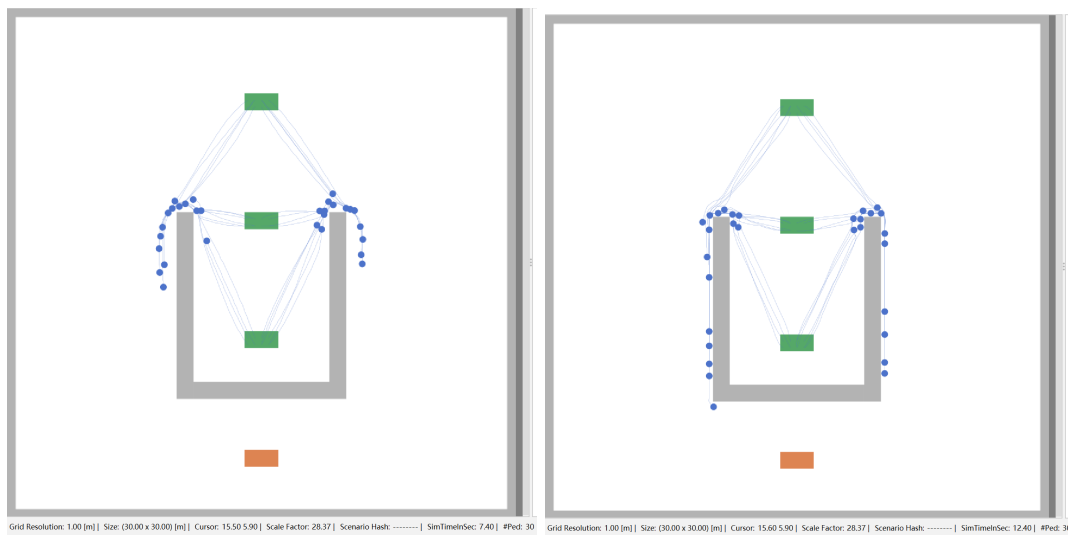


Figure 6: Different Models on Chicken Test

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

In Task 3, we have two sub-tasks to complete. The first is to run the corner scenario file separately in the GUI and the command line, and then compare their output files for differences. The second is to write a program that makes it possible to add pedestrians to the scenario programmatically (as opposed to manually). Then use this program to modify the corner scenario file to get the modified corner scenario file, and then run this file through the command line. We will get its output. Next, I will describe how we completed these two tasks.

For the first sub-task: First, we run the corner scenario file on the GUI. After the run, we automatically get an output folder. Then we enter the following code in the command line to get another output folder:

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

code added in command line

Now we enter the following command in the command line to compare the files in the two folders one by one:

FC /A file1 file2

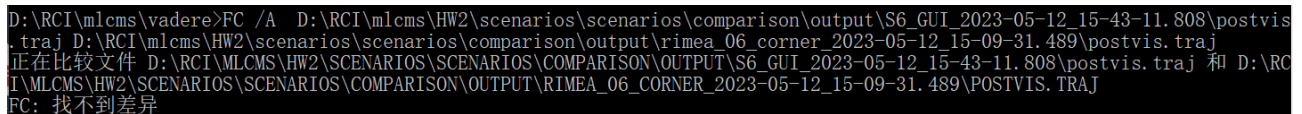


Figure 7: No difference between these two kinds of output folders

As shown in Figure 7, we found that these two output folders are no different, so we can safely use the command line to operate Vadere as a black box.

For the second sub-task: We have written a Python program that allows us to add pedestrians in the corner scenario file. This code defines three methods, `read_scenario`, `add_pedestrian`, and `find_free_id`, which are used to operate the given JSON file.

1. **read\_scenario**: This function reads a JSON file and returns a Python dictionary containing all the information about the scene. The function takes one argument, `path`, which specifies the path to the file to be read.
2. **add\_pedestrian**: This function is used to add a pedestrian in the given scenario. All properties of the pedestrian (such as position, speed, target, etc.) can be provided as arguments to this function. If these arguments are not provided, the default values will be used. After adding the pedestrian, the function writes the modified scenario to a new JSON file.
3. **find\_free\_id**: This function is used to find an id that can be used by a new pedestrian or target. The function first creates a set containing all ids already in use in the scene, then finds an unused id. If the `find_min_free_id` argument is True, the function returns the smallest available id; otherwise, it returns the current maximum id plus one.

In addition, we defined the main entry point for this Python file,

```
1 if __name__ == '__main__':
2     add_pedestrian(
3         scenario_path="D:/RCI/mlcms/vadere/Scenarios/ModelTests/TestOSM/scenarios/
4         rimea_06_corner.scenario",
5         out_scen_name="task3",
6         output_path="D:/RCI/mlcms/vadere/output/task3.scenario",
7         position=(12, 2),
8         targetIds=[1]
9     )
```

code added in command line

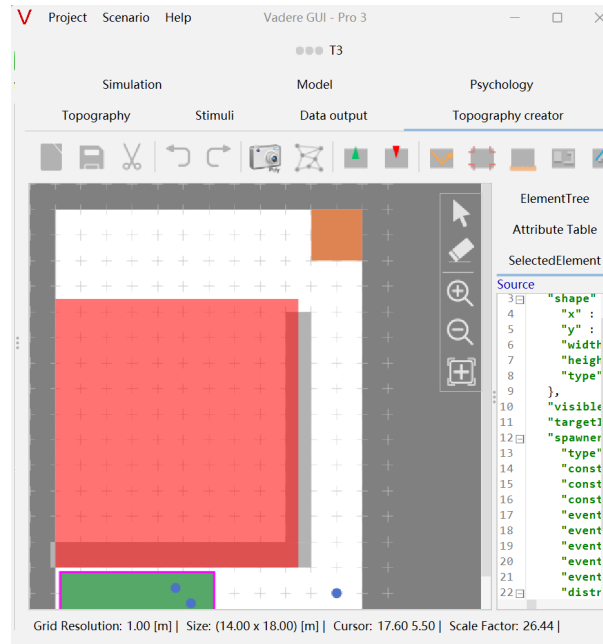


Figure 8: Appearance of the modified corner scenario in the GUI

Figure 8 is the appearance of the modified corner scenario on the GUI. Now when we run it, we find that the added pedestrian arrives at the target the fastest. After checking the postvis.traj file, we find that it only took 0.89 seconds (Runtime).

Finally, as we did in the first task, we run the modified file through the command line, We will get an output folder. This folder is the same as the output folder obtained after manually adding pedestrians.

## Report on task TASK 4, Integrating a new model

### 1 Integrate the SIR model in Vadere

To integrate the SIR model in Vadere, the model classes were added in their respective packages as suggested in the Moodle and the project was rebuilt to verify. The UML diagram of the SIR model is shown in Figure9:

There are three main classes that define the SIR Model: SIRType, SIRGroup, SIRGroupModel. Their important properties and functions are explained below.

**AbstractGroupModel:** This abstract class implements the class GroupModel. SIRGroupModel extends this class and adds proper implementation of the functions. This class has four abstract functions:

- *registerMember(Pedestrian p, T currentGroup)*: This function takes pedestrian and group as input and registers the pedestrian to the given group.
- *getNewGroup(int size)*: This function creates a new group with size given in the input.
- *getNewGroup(int id, int size)*: This function creates a new group with given ID and size.
- *assignToGroup(Pedestrian p)*: This function assigns a pedestrian to a group.

**AttributesSIRG:** This class defines the infection and recovery attributes for the SIR model. This class has the following properties which all have their respective GET methods:

- *infectionAtStart* : the number of pedestrian infected at the beginning.
- *infectionRate*: the probability of a pedestrian to be infected when they come in contact with another infected pedestrian.
- *recoveryRate*: the probability of a pedestrian recovering after infection.

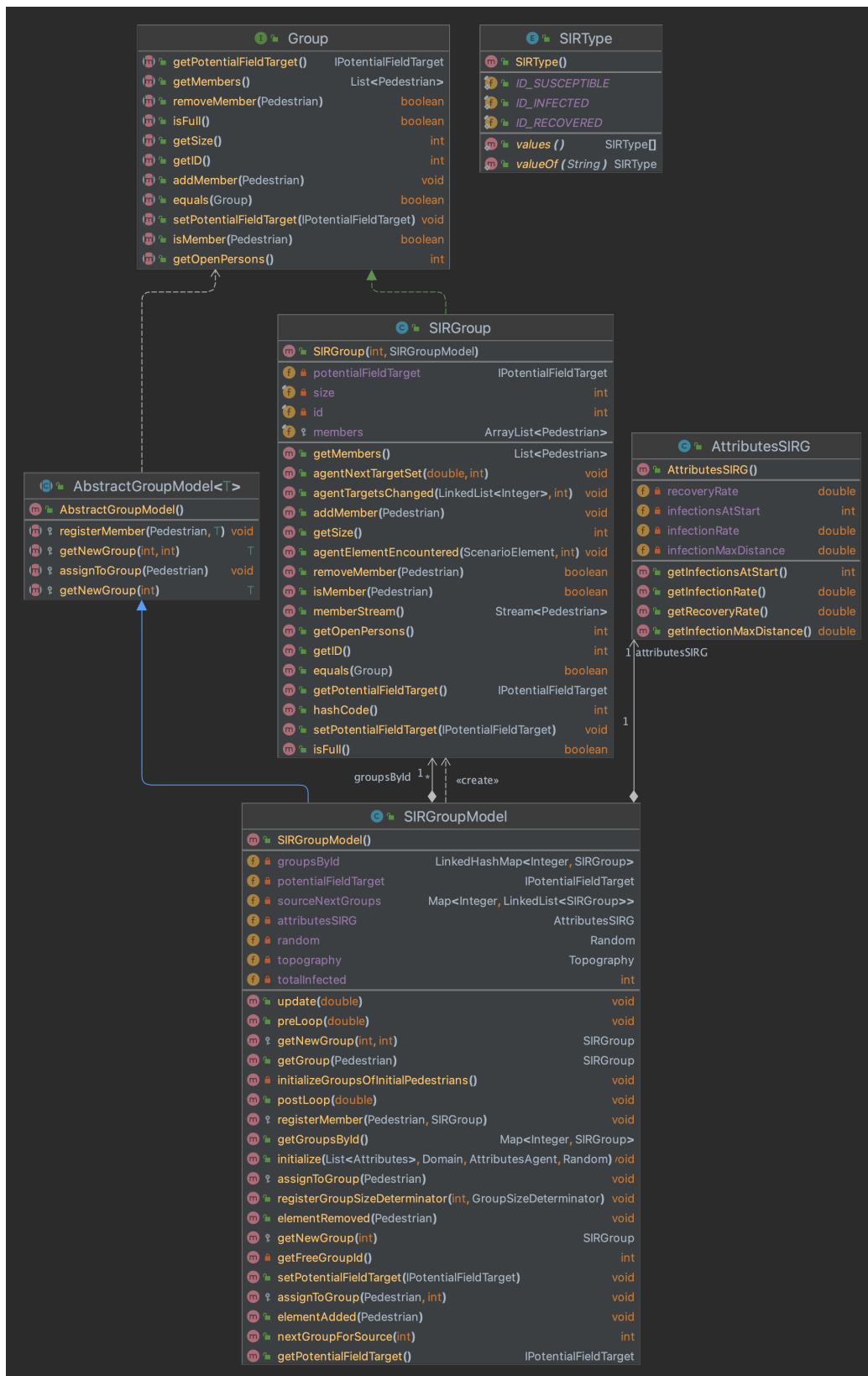


Figure 9: UML Diagram of SIR model



- *infectionMaxDistance*: the range where a infected pedestrain can infect nearby pedestrains.

**SIRGroup**: SIRGroup class implements Group class. It holds important group properties such as group members and group size. It has GET/SET methods for most of the properties.

**SIRType**: SIRType is an enum class which defines three states for the pedestrain: Infected, Susceptible and Removed.

- *ID Infected*: Represents the state of a pedestrian that is infected.
- *ID Susceptible*: Represents the state of a Pedestrian that can be infected
- *ID Removed*: Represents the state of a Pedestrian that has recovered, isolated or has immunity for the infection.

**SIRGroupModel**: This class is the center piece for the integration of SIR model in vadere. This class contains the functions necessary for implementation of susceptible, infected, removed (SIR) groups. It stores values as total infected pedestrians.

- *update(double simTime)*: This function check the nearby pedestrains using LinkedCellsGrid and decides which pedestrains will be infected.

## 2 Output Processor for the simulations

To analyze the simulation, the FootStepGroupIDProcessor provided in the moodle was used. The output was saved in a CSV format format(SIRinformation.csv). This class has a modified **doUpdate** function that takes the current simulation state and if the model is SIR stores the pedestrian id and pedestrain trajectory. Plus the pedestrian are analyzed by groups. In our case, the states will be analyzed as infected, susceptible and recovered.

## 3 S, I, R groups visualization

The `getGroupColor` method in the file **VadereGui/src/org/vadere/gui/components/model/SimulationModel.java** is used to give group color. This is currently done by creating a new color for a group by using a random number generator (RNG). Since there will be three distinct groups (S,I,R), a fixed color map will work just as well. So, the updated SimulationModel has three new addition to the `colorMap(groupId, Color)`: infected(**red**), recovered(**green**), susceptible(**blue**).The corresponding code is shown below.

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

Add new attributes

Then we change the default simulator coloring agent, then the mapping between group ID and group coloring is added. Details are shown below.

```
1  public SimulationModel(T config) {
2      super(config);
3      this.config = config;
4      this.colorMap = new ConcurrentHashMap();
5      this.colorMap.put(-1, config.getPedestrianDefaultColor());
6      this.random = new Random();
7      this.config.setAgentColoring(AgentColoring.GROUP);
8      this.colorMap.put(0, this.infected_color);
9      this.colorMap.put(1, this.susceptible_color);
10     this.colorMap.put(2, this.recovered_color);
11 }
```

Code Extracts of SimulationModel

## 4 Efficient distance computation

Since each pedestrian checks every other pedestrians, the distance computation for neighbors is not efficient. If we establish that the infection only occur in the given infection radius, then all pedestrians who are further away than the infection radius can be ignored. This will greatly decrease the number of near neighbor checks that needs to be computed. Also, once the pedestrians are already infected, there is no need to check its neighbours anymore.

Based on these ideas, we improve the code of *update* in *SIRGroupModel* from two aspects. First, we check whether the pedestrian is already infected. In that way, we could exclude the infected pedestrian from susceptible pedestrian. So only the susceptible pedestrian are looped to be checked. Second, for the potential infected pedestrian, only search for the infected pedestrian inside the max infection radius.

```

1  public void update(final double simTimeInSec) {
2  // check the positions of all pedestrians and switch groups to INFECTED (or REMOVED).
3  DynamicElementContainer<Pedestrian> c = topography.getPedestrianDynamicElements();
4
5  if (c.getElements().size() > 0) {
6      for(Pedestrian p : c.getElements()) {
7          // loop over neighbors and set infected if we are close
8          //The already infected pedestrian could be considered anymore
9          if(getGroup(p).getID() == SIRType.ID_INFECTED.ordinal())
10             continue;
11         // The infection only happens in the range of max infected radios
12         List<Pedestrian> Neighbours = c.getCellsElements().getObjects(p.getPosition(),
13             attributesSIRG.getInfectionMaxDistance());
14
15
16         for(Pedestrian p_neighbor : Neighbours) {
17             if(p == p_neighbor || getGroup(p_neighbor).getID() != SIRType.ID_INFECTED.ordinal())
18                 continue;
19             double dist = p.getPosition().distance(p_neighbor.getPosition());
20             if (dist < attributesSIRG.getInfectionMaxDistance() &&
21                 this.random.nextDouble() < attributesSIRG.getInfectionRate()) {
22                 SIRGroup g = getGroup(p);
23                 if (g.getID() == SIRType.ID_SUSCEPTIBLE.ordinal()) {
24                     elementRemoved(p);
25                     assignToGroup(p, SIRType.ID_INFECTED.ordinal());
26                 }
27             }
28         }
29     }
30 }
31

```

Code Extracts of SimulationModel

## 5 Result analyze

According to the request, we set the first scenario as shown in the Figure10. This is a static scenario with 1000 pedestrians. Some of the pedestrians have already be infected and none of them is recovered.

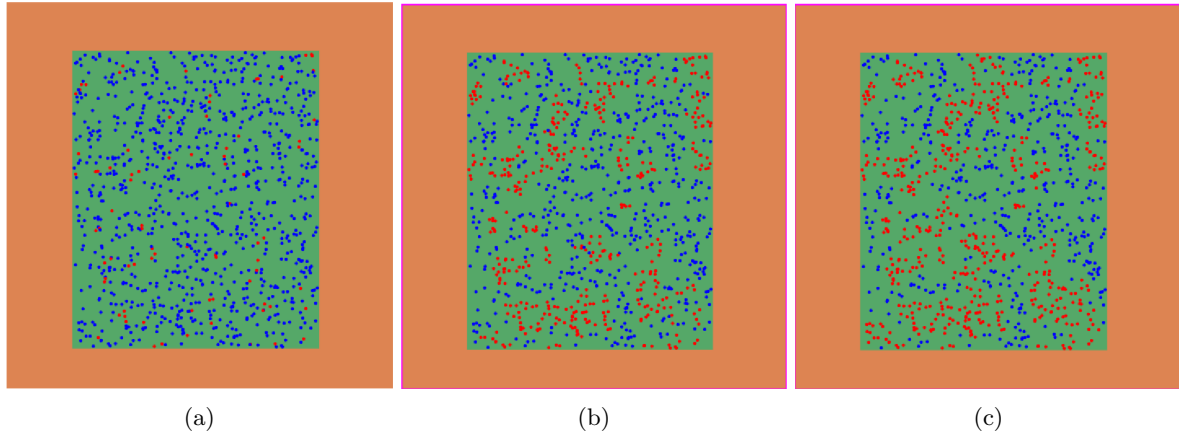


Figure 10: Evolution of simulation for the task 4-1.

At first, we set the infected rate to 0.07 and 10 infected pedestrians are added. As the result in Dash, we could find out that half of the pedestrians would be infected at about 27s.(Figure 11)

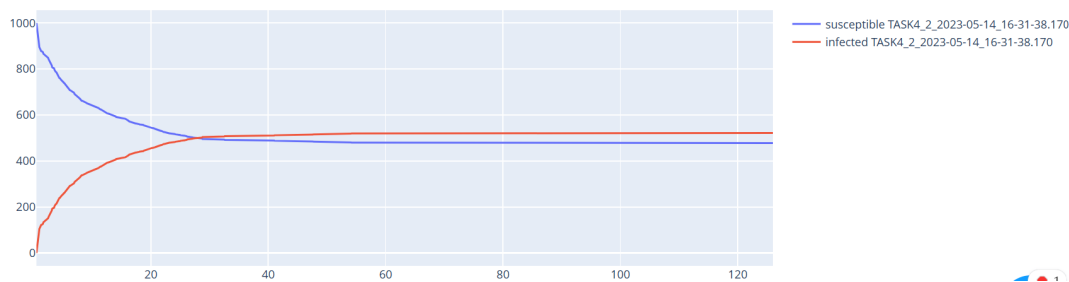


Figure 11: Code Extracts of updateLocomotionLayer()

Then the infection-Rate of the model will increase to 0.12. We simulate the same scenario with the same attributes again. The result as shown in Figure, we can find out that half pedestrians will be infected in about 17s. This half-infected time has been dramatically reduced.(Figure 12)

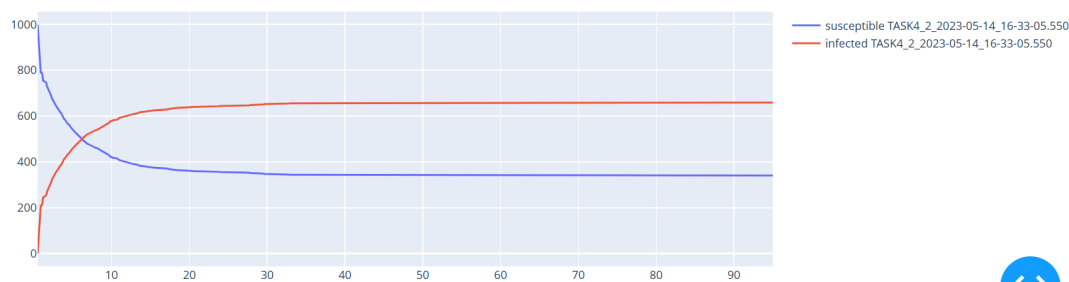


Figure 12: Code Extracts of updateLocomotionLayer()

In next step, we need to set up a new scenario. Actually, 100 pedestrians on the left will go to right. Meanwhile another 100 pedestrians will also move from right to left. So that these two pedestrians will meet others on their way. (Figure 13)

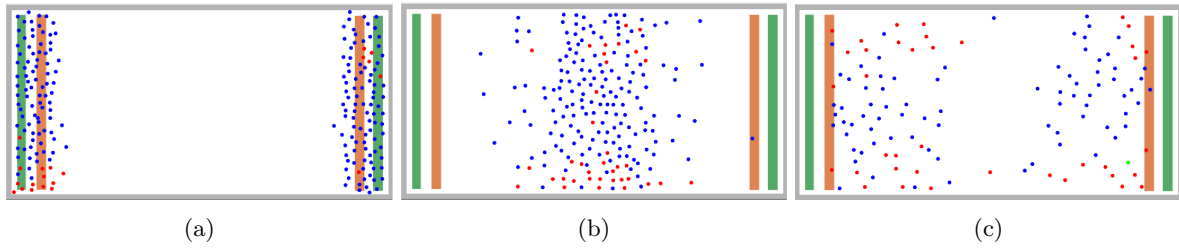


Figure 13: Evolution of simulation for the task4-2

What we found was that at first the infection would happen smoothly. When two groups of people come into contact with each other, infection will occur rapidly. When the two groups of people are separated, the infection situation will level off, and there will be little change. finally 132 Pedestrians will be infected, and in the period 14s-24s, The corresponding development is shown in Figure14)

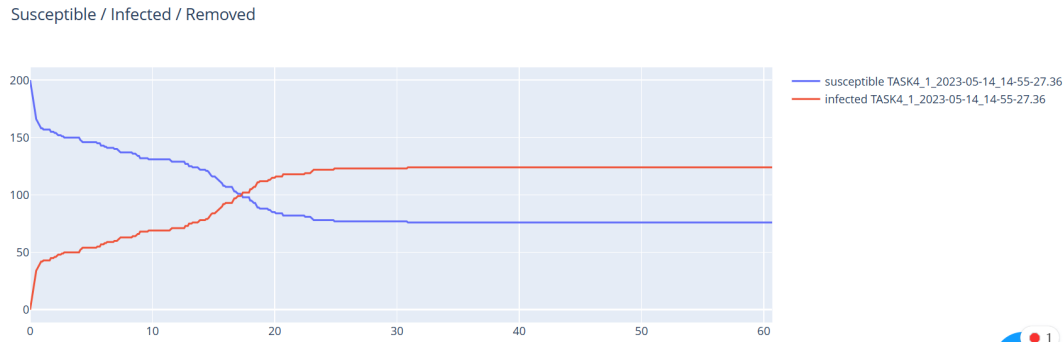


Figure 14: Code Extracts of updateLocomotionLayer()

## 6 Decouple the infection rate and the time step

The infection rate was associated with the step size of the simulation, which is difficult to interpret and also changes the infection behavior if the time discretization changes. So we need to modify the update() method to run at fixed intervals regardless of the time step size.

Instead of calling update() every time updateLocomotionLayer() is invoked, the update() method is called multiple times at once or after a certain number of calls, depending on whether the time step is greater or smaller than the fixed time. This approach allows for flexibility in setting the fixed time. Once the cumulative time step reaches the fixed time, the infection process should occur once after these calls to updateLocomotionLayer(). The cumulativeTime is set to keep track of the accumulated time. The while loop handles both cases. Since this modification only applies to the SIR model, an if statement has been added to verify if the model is an instance of this class.

```

1 private void updateLocomotionLayer(double simTimeInSec) {
2     //double CumulativeTime = 0.0;
3     for (Model m : models) {
4         List<SourceController> stillSpawningSource = this.sourceControllers.stream().filter(s ->
5             !s.isSourceFinished(simTimeInSec)).collect(Collectors.toList());
6         int pedestriansInSimulation = this.simulationState.getTopography().
7             getPedestrianDynamicElements().getElements().size();
8         int aerosolCloudsInSimulation = this.simulationState.getTopography().getAerosolClouds().
9             size();
10
11         // Only update until there are pedestrians in the scenario or pedestrian to spawn or
12         // aerosol clouds persist
13         if (!stillSpawningSource.isEmpty() || pedestriansInSimulation > 0 ||
14             aerosolCloudsInSimulation > 0) {
15             if (m instanceof SIRGroupModel){
16                 CumulativeTime += this.attributesSimulation.getSimTimeStepLength();
17                 double fixedTime = 1.0;
18                 while (CumulativeTime >= fixedTime) {
19                     m.update(simTimeInSec);
20                     CumulativeTime -= fixedTime;
21                 }
22             }
23             else{
24                 m.update(simTimeInSec);
25             }
26             if (topography.isRecomputeCells()) {
27                 // rebuild CellGrid if model does not manage the CellGrid state while updating
28                 topographyController.update(simTimeInSec); //rebuild CellGrid
29             }
30         }
31     }
32 }

```

Code Extracts of updateLocomotionLayer()

## 7 Describe and motivate possible extensions

SIR model is an excellent as base to simulate more complex scenarios. For example:

- Vaccinated Pedestrian: During the past 3 years pandemic, there was a huge push for vaccination development. Those vaccination helped a lot of people recover at higher pace.
- Pedestrian Contact History: A list with recently met pedestrians would help in finding and isolating infected pedestrians. This method was tested and applied by many countries during the covid pandemic.
- Self Isolation: This would define the percentage of pedestrian isolates as soon as they are infected. This would help us model how quickly the infection would disappear if proper actions was taken by the pedestrian. This would reflect closely to the real world as many covid infected people followed self isolation.

**Report on task TASK 5, Integrating a new model****Adding recovered state**

In this Task, we need to add features to the model and show the test result.

First we need to add recovered state. The specify meaning of recover is that those pedestrians could not be infected or infect others anymore. In **AttributesSIRG**, we according to the previous S,I groups, add recoveryRate attribute. We set a random chance that infected pedestrians will recover. By definition of recovery, recovered pedestrians will no longer be involved in infection. Besides we add **ID RECOVERED**

```

1  public void update(final double simTimeInSec) {
2  // check the positions of all pedestrians and switch groups to INFECTED (or REMOVED).
3  DynamicElementContainer<Pedestrian> c = topography.getPedestrianDynamicElements();
4
5  if (c.getElements().size() > 0) {
6  for(Pedestrian p : c.getElements()) {
7  // loop over neighbors and set infected if we are close
8  //The already infected pedestrian could be considered anymore
9  SIRGroup g = getGroup(p);
10 if (getGroup(p).getID() == SIRType.ID_INFECTED.ordinal()) {
11 double recoveryRate = attributesSIRG.getRecoveryRate();
12 if (this.random.nextDouble() < recoveryRate) {
13 elementRemoved(p);
14 assignToGroup(p, SIRType.ID_RECOVERED.ordinal());
15 }
16 }
17
18 else if (g.getID() == SIRType.ID_SUSCEPTIBLE.ordinal()) {
19 // The infection only happens in the range of max infected radios
20 List<Pedestrian> Neighbours = c.getCellsElements().getObjects(p.getPosition(),
21 attributesSIRG.getInfectionMaxDistance());
22 }
23
24 for(Pedestrian p_neighbor : c.getElements()) {
25 if (p == p_neighbor || getGroup(p_neighbor).getID() != SIRType.ID_INFECTED.ordinal())
26 continue;
27 double dist = p.getPosition().distance(p_neighbor.getPosition());
28 if (dist < attributesSIRG.getInfectionMaxDistance() &&
29 this.random.nextDouble() < attributesSIRG.getInfectionRate()) {
30
31 if (g.getID() == SIRType.ID_SUSCEPTIBLE.ordinal()) {
32 elementRemoved(p);
33 assignToGroup(p, SIRType.ID_INFECTED.ordinal());
34 }
35 }
36 }
37 }
38 }
39 }

```

Edited code from SIRGroupModel.py

In SIRVisualization, we also need to add the corresponding code part. In that way the recovery is introduced into SIRVisualization and the green recovered curve could be added.

```

1  for pid in pedestrian_ids:
2  simtime_group = df[df['pedestrianId'] == pid][['simTime', 'groupId-PID5']].values
3  current_state = ID_SUSCEPTIBLE
4  group_counts.loc[group_counts['simTime'] >= 0, 'group-s'] += 1
5  for (st, g) in simtime_group:
6  if g != current_state and g == ID_INFECTED and current_state == ID_SUSCEPTIBLE:
7  current_state = g
8  group_counts.loc[group_counts['simTime'] > st, 'group - s'] -= 1
9  group_counts.loc[group_counts['simTime'] > st, 'group - i'] += 1
10 elif g != current_state and g == ID_RECOVERED and current_state == ID_INFECTED:
11 current_state = g
12 group_counts.loc[group_counts['simTime'] > st, 'group - i'] -= 1
13 group_counts.loc[group_counts['simTime'] > st, 'group - r'] += 1
14 break
15 return group_counts

```

part of file-df-to-count-df

```

1  ID_SUSCEPTIBLE = 1
2  ID_INFECTED = 0
3  ID_RECOVERED = 2
4
5  group_counts = file_df_to_count_df(data, ID_INFECTED=ID_INFECTED, ID_SUSCEPTIBLE=
ID_SUSCEPTIBLE, ID_RECOVERED=ID_RECOVERED)
6  # group_counts.plot()
7  scatter_s = go.Scatter(x=group_counts['simTime'],
8                          y=group_counts['group-s'],
9                          name='susceptible' + os.path.basename(folder),
10                         mode='lines')
11  scatter_i = go.Scatter(x=group_counts['simTime'],
12                          y=group_counts['group-i'],
13                          name='infected' + os.path.basename(folder),
14                         mode='lines')
15  scatter_r = go.Scatter(x=group_counts['simTime'],
16                          y=group_counts['group-r'],
17                          name='recovered' + os.path.basename(folder),
18                         mode='lines')
19  return [scatter_s, scatter_i, scatter_r], group_counts

```

part of create-folder-data-scatter

### Testing the new setup

Before proceeding with the different proposed tests, we set up a scenario that is simpler than the ones that will be tested later, to check that everything works well. There are only 100 pedestrians in this scenario, the infection rate is set to 0.1, and 10 pedestrians will be infected at the beginning. The newly implemented infection/recovery process will be executed in a fixed time of 1 second with a time step of 0.5. First, we set the recovery rate to 0.0. The expected result of the simulation is that the number of infected pedestrians will grow until there are no more susceptible pedestrians left, or the remaining susceptible pedestrians cannot be infected. As shown in the figure 15, at the end of the simulation, there are only 2 susceptible pedestrians left, and they

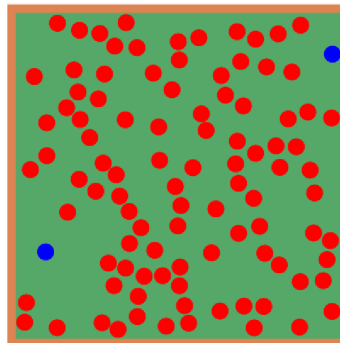


Figure 15: End of the simulation beginning with 10 infected pedestrians

are not infected because they are not within the infectable range of other infected pedestrians. Figure 16 shows the changes in the number of infected and susceptible people. These changes are as we expected.

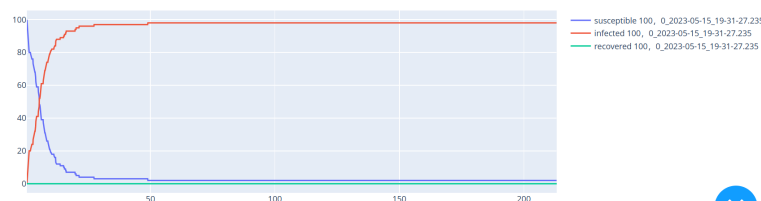


Figure 16: Evolution of the number of pedestrians in different groups when recovery rate=0

Next, the infection rate remains at 0.1 and the recovery rate will be set to 0.05. This means that infected pedestrians have a 5% chance of recovering at each time step after being infected, so it can be expected that all infected pedestrians will recover as long as the simulation is long enough. As shown in figure 17 and figure

18, at the beginning of the simulation, there are many more infected pedestrians than recovered pedestrians. Later, the infected pedestrians gradually recovered until there were no infected pedestrians. The total number of infected pedestrians was reduced compared to previous cases, as some pedestrians had neighbors who would not have been infected if they had recovered from the infection, creating a "safety barrier" for these pedestrians, prevent them from being infected. Everything seems to make sense, so we can proceed to the testing session.

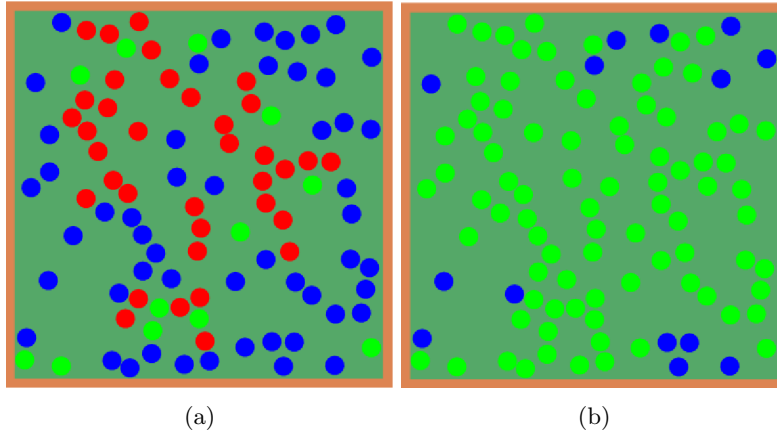


Figure 17: The number of pedestrians in different groups at the beginning and the end of simulation

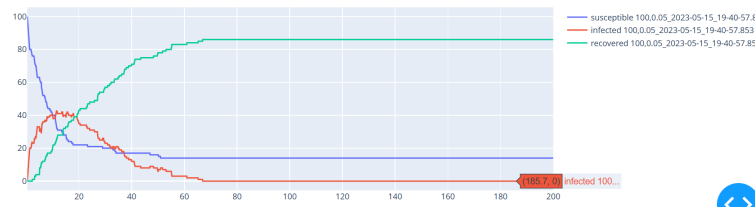


Figure 18: Evolution of the number of pedestrians in different groups when recovery rate=0.05

### Test 1

The first test required building a fairly large scene with a source spawning 1000 pedestrians above a target. The purpose of this test is to simulate how the number of people who are susceptible, infected and recovered changes over time in a real-world scenario. For this purpose, a source of size 30x30 has placed 1000 static pedestrians with 10 infected and 990 susceptible. The Figure 19 shows the results of this simulation. In fact it is similar to the results obtained in the previous test (Figure 18). Now let's take a closer look at the changes in the number of infections. We can see that the number of infected pedestrians does not grow as quickly as before, since pedestrians are now recovering. We can also see that about 300 pedestrians remained uninfected by the end of the simulation, either because their neighborhood had no infected pedestrians at any point, or because infected neighbors have been infected before spreading the virus, has recovered.

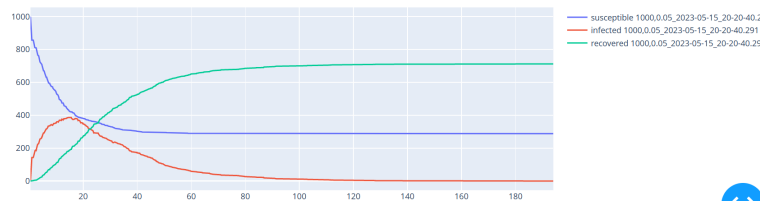


Figure 19: Evolution of the number of pedestrians in different groups with 1000 pedestrians

### Test 2

The second test called for experimenting with infection rates and recovery rates in the same scenarios as before to see how the evolution of the disease changes when these parameters are changed. Since we have studied the behavior when the recovery rate is less than the infection rate in the previous tests, let's study



the behavior when the recovery rate is equal to or higher than the infection rate. Here, we set the infection rate as 0.1. By comparing the simulation results using different recovery rates, as shown in Figure 20, it can be concluded that the higher the recovery rate, the fewer people the disease will infect. Specifically, when the recovery rate is equal to the infection rate, the maximum number of infections is 250. When the recovery rate is 0.15, higher than the infection rate, the maximum number of infections is 200, and both are lower than the maximum number of infections, when the recovery rate is 0.05, which is lower than the infection rate. According to the simulation results, we can speculate that curing patients as soon as possible will be one of the effective ways to curb the spread of the disease.

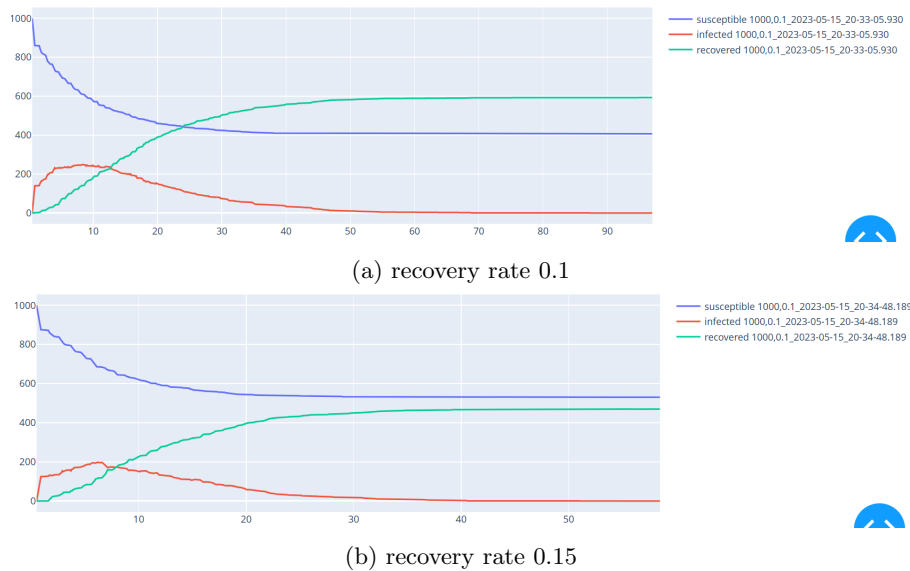


Figure 20: Different behaves under different recover rate

### Test 3

In this scenario, we need to create a 30x30m scene to simulate a supermarket and test the impact of pedestrian behavior on situations like infection, recovery and so on.

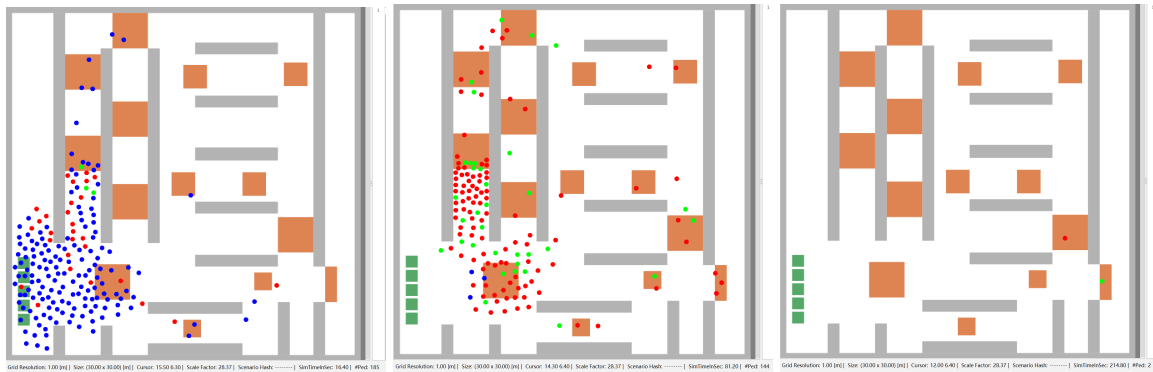


Figure 21: normal Scenario of a Supermarket

As shown in Figure 21, we first define the scenario with following parameters as "Normal": The supermarket has one entrance and one exit, where people from 5 different sources walk in and hang in there for a while, then get out from the exit. each source has 40 people, where the `InfectionatStart` is set to 1, and the `pedPotentialPersonalSpaceWidth` is set to 1.2.

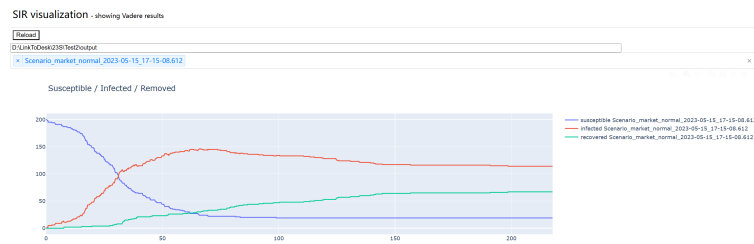
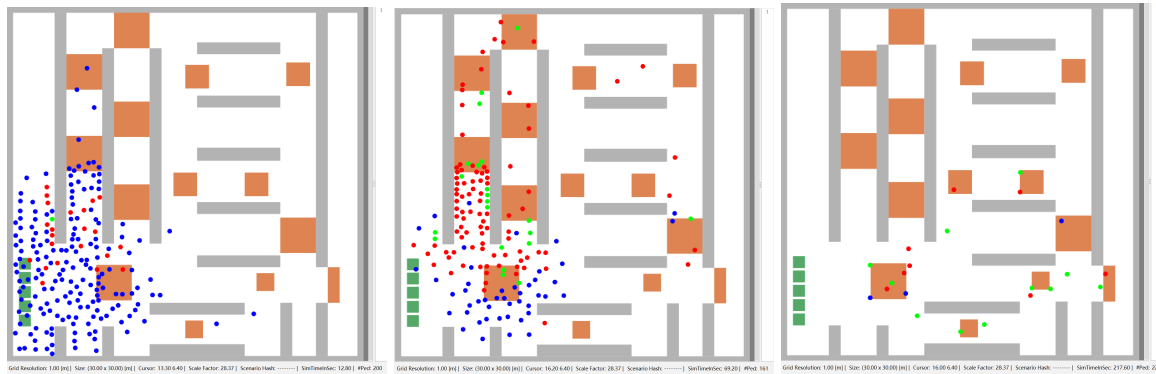


Figure 22: States of people in the supermarket

It can be seen in Figure 22, during the simulation, within the first 50 simtime, the number of uninfected people has fallen sharply, while the numbers of infected and recovered people have respectively risen. Same as before, we define this result as "normal".

Then what will happen if we increase the `pedPotentialPersonalSpaceWidth`, that is, let the distance between people become larger?

Figure 23: Scenario of a Supermarket with high `pedPotentialPersonalSpaceWidth`

In Figure 23 shows the process of a simulation, where `pedPotentialPersonalSpaceWidth` is set to 2.2. We can tell from the Figure 24 that the curve of the uninfected has indeed flattened a little bit, which means the infection is partially slowed down. But it's still not showing too much effect, it can be concluded that if we keep increase the `pedPotentialPersonalSpaceWidth`, the uninfected curve will surely be more flat, but other problems will show up. In our case, we have tried to increase it to 5, and the pedestrian just stuck there, without moving. Because the required `SpaceWidth` is too large that people on the two aisles will restrict each other, so that no one can move.

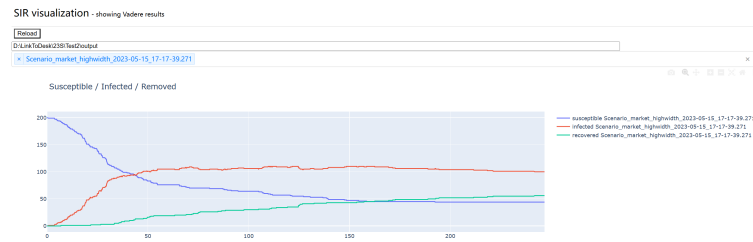


Figure 24: States of people in the supermarket(high SpaceWidth)

So far, we can't help but ask, besides increasing the distance between people, what are other ways to reduce the risk of infection? Here, we tested another possible improvement: reducing the number of people in the market(in Figure 25).

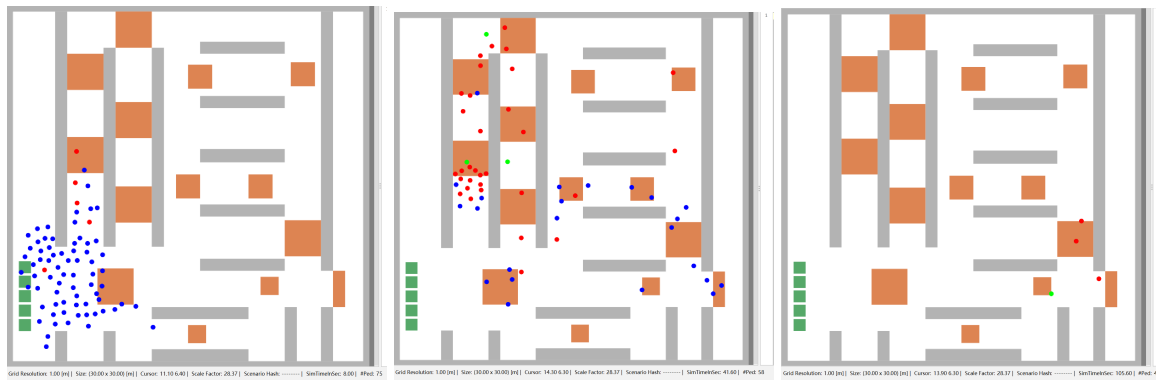


Figure 25: Scenario of a Supermarket with low amount of people

The result can be said to be quite satisfactory, because it can be clearly seen that, with the same SpaceWidth as normal scenario, if we reduce the amount of people per source from 40 to 15, the total amount will be only 75. As shown in Figure 26, If we take simtime = 50 as a mark, the previous two scenarios all have more than 50% infection-rate, but in this low-peopel-scene it is obviously lower than 50%.

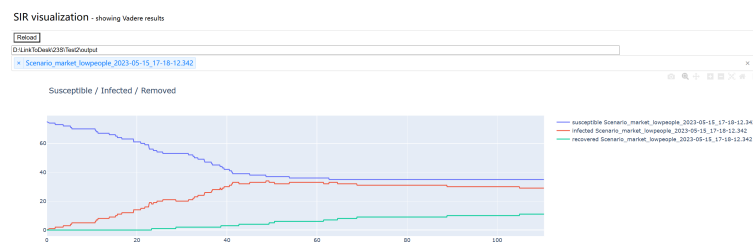


Figure 26: States of people in the supermarket(low amount of people)

From there maybe we can roughly draw one conclusion: Social distancing and purely people reducing can both help lower the risk of getting infected, but the latter one is more effective. But there is also no need to reduce it too hard, after all, if no one comes to supermarket, it's the safest way, but we all know that's impossible.