Report for exercise 2 from group B

Tasks addressed:	5
Authors:	HORIA TURCUMAN (03752553) GEORGI HRUSANOV (03714895) UPPILI SRINIVASAN (03734253)
Last compiled:	2022–11–17
Source code:	https://gitlab.com/turcumanhoria/crowd-modeling

The work on tasks was divided in the following way:

HORIA TURCUMAN (03752553)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
GEORGI HRUSANOV (03714895)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
UPPILI SRINIVASAN (03734253)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%

Report on task 1, Setting up the VADERE environment

In terms of configuration, in order to use VADERE [2] you needed to either download Java 11 or a newer version of Java or have it already installed on your machine. This could be achieved by visiting the Oracle[1] website and downloading the Java Development Kit (JDK) installation package that is tailored precisely for the computer that will be used to complete the task at hand. After this step has been finished, the package will be installed. Another option you could choose is to download the OpenJDK, which is the free and open source implementation of the Java SE Platform Edition. To use VADERE, you must now download the program from the VADERE releases website [2]. More specifically, the "master branch" release must be downloaded rather than the stable one. For the first exercise however we do not need the source code version, but rather only the compiled *JAR* file for the Graphical User Interface. After unzipping the downloaded directory there are two ways to launch the *gui*, either by double-click the file vadere-gui.jar to open the VADERE GUI, or by using the terminal with the command

```
$ java -jar vadere-gui.jar
```

in the directory where you unzipped the downloaded file . First several test from the RiMEA Guidelines, more precisely the first and the sixth scenario had to be fondly described, visualized[6]. Apart from the that the already known from the last exercise, so-called "chicken test," had also been put to the test. All of this should happen using the Optimal Steps Model (OSM)[7, 9, 8] first within the scope of this task.

RiMEA scenario 1: This is the first and very basic test that was mentioned in the RiMEA Guidelines for Microscopic Escape Analysis. As this is the most fundamental test, it consists of a single person walking in a straight line towards a single target positioned in the center of the grid [6]. The first step is to load it using the GUI that is provided with Vadere itself. One should open the project, navigate through the scenarios, find the OSM[7, 9, 8] ones and simply open them, in order for VADERE to carry out this test. Users can select from a range of models before initiating simulations by clicking on the "Model" tab located within the graphical user interface (GUI). The "Load template" option gives you access to a library of pre-made models that you can use in VADERE. These models can be chosen to serve as the foundation for new developments, as we will see later on. Another way to load the tests is to goal to the "Project" section on the top left corner of the gui, go to *Open* and navigate to the concrete scenario folder one would like to open and load. The situation in object is quite simple because all the pedestrian has to do is walk in a straight line toward the goal, with no obstacles or other pedestrians to contend with along the way. As a result, the end result of this simulation is analogous to the one developed from the ground up in the previous exercise the cellular automaton; nevertheless, there are several major differences:

- The grid of the image that we had on the cellular automaton and every object that was assigned in it, occupies one complete cell, and the general shape of all of the objects was a square.
- In exercise on the previous sheet, the time step is fixed from the begging and was a constant value, and the simulated time is kept in sync with the real time based on concrete. On the other hand, utilizing VADERE, it is possible to simulate more complex scenarios in considerable less time. This is made feasible by the ability to choose a ratio that correlates between the time that passes in the simulation and the actual amount of time that has passed in the real world. This could be altered and changed when one opens the simulation tab of the concrete scenario and adjust the *"realTimeSimTimeRatio"*. When this value is set to be 1.0, this means that the simulation would be as fast as real time. Reducing this value would speed up the simulation, whereas if this value is to be increased this will slow up the simulation.
- In exercise 1, the walking pace was predefined and set to be 1.0 meters per second. However, in VADERE with OSM [7, 9, 8], is a dynamic and not a predefined constant value and is unique.
- Another thing, we can observe is that the trajectory remains basically unchanged in both cases (the cellular automaton and VADERE with OSM). This is most likely due to the case's simplicity, as the whole idea behind that is simply to go in a straight line and also important to mention is that there are no specific interactions and avoidances that need to be described and handled by both the cellular automaton and Vadere. When a pedestrian approaches the target in the VADERE implementation, there is a slight deviations from the perfect straight line, to the right and to the left, as the individual gets closer to the goal as seen in Figure 1. This does not happen in the performance of exercise 1, however, because the pedestrian and the goal are the same size and perfectly aligned and they are also of the same shape.

- We could conclude that the implementation of the cellular automaton was much more "closed" and also more discretized

Figure 1: Sligth deviations from the perfect line in the OSM of first RiMEA Scenario

RiMEA scenario 6: In this scenario, twenty persons circle to the left of a corner to get to their objective. In this case, the following differences may be observed between the implementation of the cellular automaton from the first execercise sheet and the one in VADERE using OSM [7, 9, 8]:

- The total length of time required for all of the pedestrians to arrive at their destination is approximately 30 seconds.
- In the first portion of the experiment, the 20 pedestrians will be randomly distributed at the beginning of the hallway. This makes it automatically clear that the pedestrians in the scenario each of them will begin at varied distances from the goal, because they cannot be stacked upon each other.
- The trajectories taken by pedestrians using OSM (Figure 3) are significantly more realistic than the pathways used by pedestrians in exercise 1. Furthermore, they can be seen using the "Post-Visualization" area of the GUI. On the other we did not had to implement such functionality used for visualization in the cellular automaton, which makes it automatically much harder to examine the exact path, direction and trajectory of the pedestrians in the scenario. In the cellular automaton, the only thing you could do was try to track people moving throughout simulation time, which made examination of trajectory impossible.
- In any event, unexpected behaviors in the last segment of the trajectories presented in Figure 3 are possible.
- Some of the pedestrians on the outside of the hallway enter, while others seek to make their way to the centre of the path for no apparent reason.
- There is also something quite interesting in the behaviour of the pedestrians, because some of the people in the scenario are sometimes leaving the perfect straight line of their movement. This is to be seen when another pedestrian is approaching them and a collision could happen. However the pedestrian make this deviation and right after that they try to return to the center as soon as possible This behaviour that has less chance to be observed in a real simulation with real people. This can be seen by watching the simulation with the added option from Vadere to display the direction in which every pedestrians go, which could be enabled from the GUI.

Chicken test: During this phase of the test, one or many pedestrians standin in front of a target, and between them is a U-shaped barrier meant to catch the pedestrian if it approaches the target head-on. In the cellular automaton exercise we had to tackle the chicken test scenario in two different approaches, once using the Euclidean distance and in the other approach with Dijkstra. This delivered once successful run of the test and one unsuccessful. So we could observe that the test is passed in both the implementations of exercise 1 and VADERE with OSM[7, 9, 8]; this means that the pedestrian can move around the impediment rather than being impeded by it and thus they will successfully arrive at his or her destination. The trajectories of the two implementations, on the other hand, are extremely different:

- Cellular Automaton: The movement we had to develop on the last exercise sheet was very discretized and in a very closed scope of action. The directions where either to move forward, to the left, to the right or diagonally at a 45° angle. That is why when we simulate this scenario with the automaton the pedestrians start and make a diagonal move right away, thus being able to coast the barrier's boundary until it surpasses it. After the obstacle is avoided, again a move diagonally is made in order to go back to its main path for reaching the goals. This limited amount of actions for the pedestrians made the whole movement within the specified grid much more robotic and distant from the behaviour of a real person. Figure 2 depicts a reconstruction of the path taken by the walker in the first exercise using Dijkstra's

method (a). This diagram depicts how the pedestrian navigated the path and then reach his goal in successful manner.

- VADERE with OSM: the trajectory is more "circular" and also adaptive, closer to the real world behaviour, as seen in Figure 6. However there is also something unreal in the Vadere simulation environment. First when the pedestrians start their journey they tend to go directly inside the impediment and then after a very short amount of time they just go back and avoid the obstacle. This is also quite unnatural, because when a real person sees an obstacle, he will directly just try and avoid it and go to the outer boundary of the barrier. Even if this slight last part could be a little more direct and straightforward, it has a more natural feel overall when we compare it to the cellular automaton.
- The time needed in order for all pedestrians in the Vadere simulation to achieve their goal is something around 8 seconds when run with a *"realTimeSimTimeRatio"* value of 0.

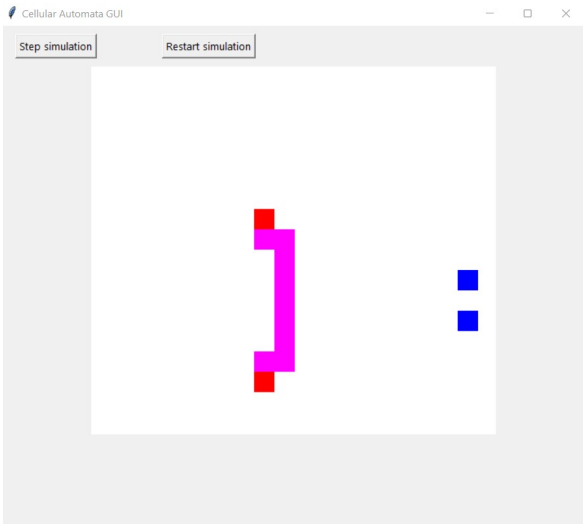

Figure 2: Chicken test scenario with the dijkstra from the Cellular Automation

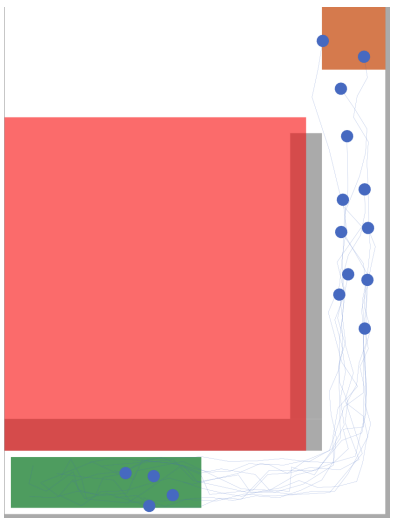

Figure 3: Sixth RiMEA Scenario composed with Vadere with OSM[7, 9, 8]

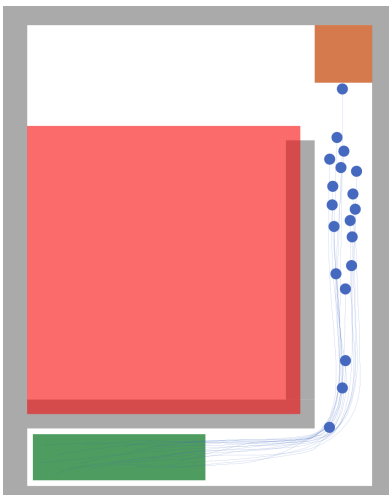

Figure 4: Sixth RiMEA Scenario composed with Vadere with SFM [5, 4]

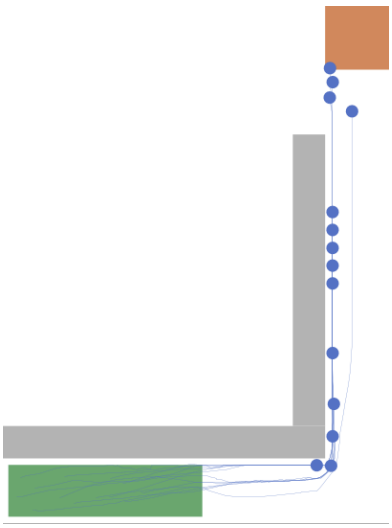

Figure 5: Sixth RiMEA Scenario composed with Vadere with GNM [3]

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

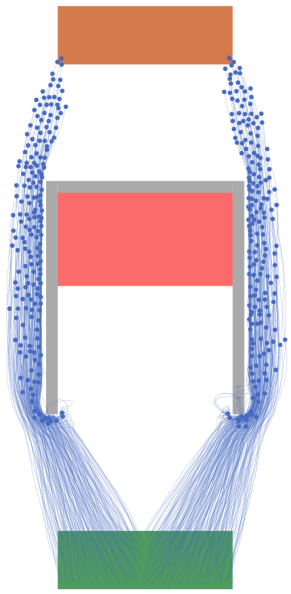

Figure 6: Chicken test scenario with the use of Vadere with OSM [7, 9, 8]

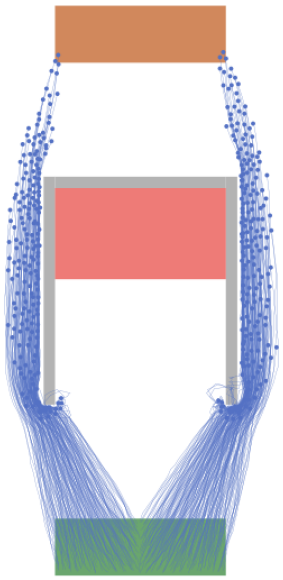

Figure 7: Chicken test scenario with the use of Vadere with SFM [5, 4]

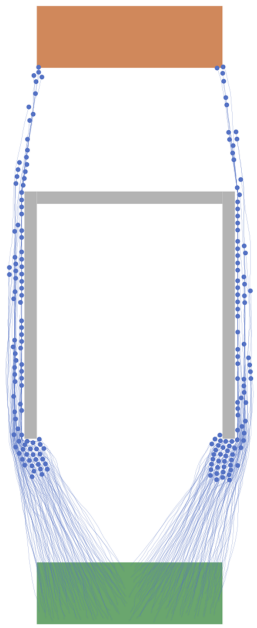

Figure 8: Chicken test scenario with the use of Vadere with GNM [3]

This task entails running the Social Force Model (SFM) [5, 4] and the Gradient Navigation Model (GNM)[3] through their paces to mimic the three scenarios from the prior assignment (RiMEA scenario 1, RiMEA scenario 6, and the chicken test). Again a found analysis of the observations that were made had to be done and some comparisons to the cellular automaton had to be drawn.

Social Force Model (SFM)

- **RiMEA scenario 1:** In this straightforward demonstration, the path follows a direct line in the direction of the destination. We can see similarities between the Social Force Model, the OSM[7, 9, 8] and the cellular automaton.
- **RiMEA scenario 6:** As we run Vader with in this situation while using SFM [5, 4] the very first thing we can observe in this behaviour is that it is much natural than the behavior displayed when using OSM [7, 9, 8]. Now unlike in the OSM all pedestrians try to take the optimal path and line for reaching their goal and make their walk much more efficient and quick. A similarity could be drawn from the optimal racing line for example, know from motorsports. Now the pedestrians do not care that much for the others surrounding them and just stick to their route. Furthermore, while using SFM [5, 4], the paths are more rounded, however when using OSM [7, 9, 8], there are some abrupt angles and stretches that appear square, the deviations we mentioned and described in the last task.
  - Comparison of the different trajectories of both OSM and SFM could be made when comparing Figures 3 and 4 for examination. The ones that come with SFM [5, 4] appear more realistic overall for various cases.
  - The pedestrians in OSM seems like they constatly strive to give way to others, going constantly in different directions or halting. This specific behaviour motion is much more "solid like". There is no such thing in the movement of the crowd in the Vadere with SFM, where the whole manoeuvre seem more fluid.
  - This is more obvious than the trajectories, which are also visible. Maybe that could be the reason why the total time required to convey all of the pedestrians to the location is 26 seconds, whereas OSM took 30 seconds.

- **Chicken test:** There is now surprise that also this model could pass successfully the chicken test with the basic obstacle avoidance. When measured against OSM [7, 9, 8], the trajectory is once again, just like in the left turn scenario, a little more straightforward and natural, but other than that we could conclude that is more similar than different. Visual comparison of the two trajectories is provided for your perusal in Figure 6 and 7. The similarities that could be seen in the trajectories could explain that both approaches had a total amount of 8 needed to complete the task.

Gradient Navigation Model (GNM):

- **RiMEA scenario 1:** Using this method, the trajectory once again follows a course that is unwaveringly oriented towards the target. Again this is the anticipated behavour, given the basic nature of the situation and the scenario definition.
- **RiMEA scenario 6:** All of the pedestrians truly endeavor to go the shortest distance to their destination (in Figure 5). It is possible to notice this line that runs adjacent to the wall highlighted more prominently; this line can be observed. Interestingly this run and the examinations we did showed that this time pedestrians were slightly slower and more closer to the inside. That is why some of the pedestrians avoided this behaviour and just decided to go round the outside of the big queue of people.
  - In general, we could not say if this movements appears more natural or not when compared to FSM [5, 4], but we could definetly can say that for the trajectory of the pedestrians when using OSM [7, 9, 8];
  - in fact, some slowdowns are to be expected. The motion of going outside the curve and getting through slightly reduces the slowing of this scenario. This way of avoiding other pedestrians however is much more practical than the one in OSM.
  - with this remarks that we made, we could explain why the total time required to transport all of the pedestrians to their destination is something around the 28 seconds mark. In RiMEA scenario 6
  - this makes this approach slightly slower than SFM, but faster than OSM [7, 9, 8].
- **Chicken test:** Just like all the other cases that we observed within the scope of this exercise, the GNM has also no problem at all when it comes to fortunately pass the the chicken test. Now here in this case the path to the target that is taken is very direct and also it could be observed that the pedestrians again do their best when it comes to sticking closely to the obstacle barrier that has to be avoided [3]. This can be seen by examining Figure 8 in conjunction with Figure 6.
  - In this running we could interestingly see that now the pedestrians behave much more naturally in the sense of approaching the impediment, the realising that they have to come back and procede to their goal. Now this is done much more smoothly than both OSM and FSM
  - The total time required to complete the task is eight seconds, just like in the previous two approaches that were used.

Report on task 3, Using the console interface from Vadere

The very first thing to be done in this exercise is to run and test Vadere on the console. There is no visible difference in the output files when Vadere is run from the terminal rather than the GUI, as expected. The results of two different runs of the identical scenario, one with a graphical user interface (GUI) and the other with a command line interface (console), were compared file by file using the diff tool available in the Linux bash shell. This command is used to show the differences between the files by doing a line-by-line comparison of the two sets of files. It differs from its sibling members, cmp and comm, in that it provides information on which lines in one file need to be modified so that the two files may be similar.

The next thing we had to do within the scope of this task is to create a way to programmatically add pedestrians to some of the scenarios, more concretely the **RiMEA scenario 6**. The script `add_pedestrian.py` was written in order to be able to insert pedestrians programmatically. It is a Python script with four functions, the most important of which is `addPedestrian`. The following stages are carried out by this function: 4:

- Load the scenario that you want to modify: This functionality is provided by the function `read_json_file` that accepts the scenario itself as a dictionary as a parameter

- The next function of this script is `run_scenario`. This function takes as an argument the concrete scenario we would like to add pedestrians to. The main logic behind this function is to run the console line command for running Vadere on the command line. This is realized with the help of the os import from python. The Python OS module contains simple methods that enable us to interact with the Operating System, get information about it, and even exert some limited control over its operations.
- Make the pedestrian: it's just a dictionary with the same structure (nested lists and dictionaries) as the one in the scenario's JSON file under the key dynamic. As a result, its structure and keys are fixed, and they must mimic what is contained in the JSON file in order to be compatible, while the values connected with each key are supplied as parameters to add pedestrian. Add the new pedestrian to the list of pedestrians: - `scenario['scenario']` returns the list: `['topography']['dynamicElements']`. The inclusion of the new pedestrian itself (more specifically ped) is done with:

  `scenario['scenario']['topography']['dynamicElements'].append(ped)`.

  After the pedestrian is added to the scenario we would like to save the new JSON that represents the modified version. This is done with the help of the `json.dump` method from the json module. The new pedestrian is now included in this modified scenario file, which can be loaded into Vadere.

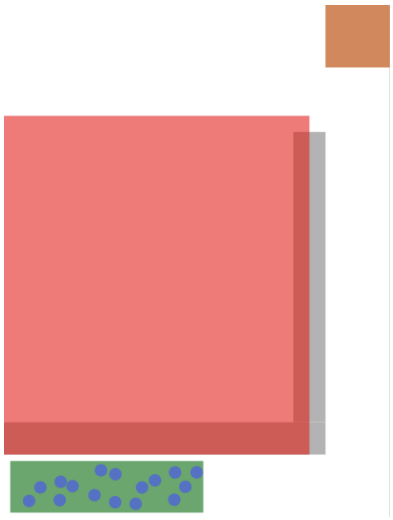

Figure 9: The original RiMEA Scenario

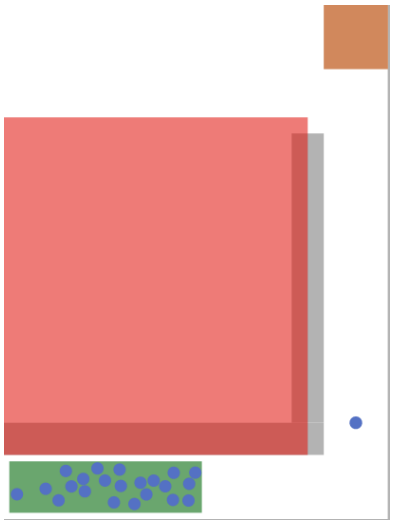

Figure 10: The RiMEA Scenario with one added pedestrian

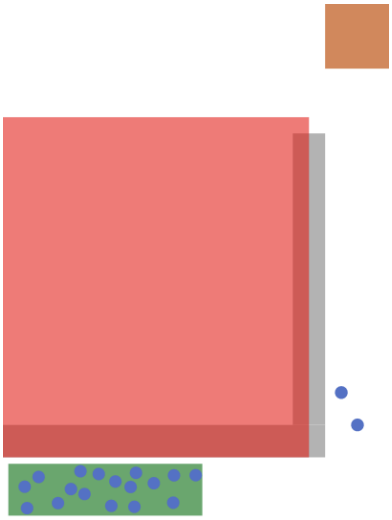

Figure 11: The RiMEA Scenario with two added pedestrians

In the end there is also a main method, that is used solely for testing purposes. It takes the sixth RiMEA Scenario and add a pedestrian to it, using the already implemented functionality described above. When calling the `addPedestrian` method, in the first case, there is a specific id given for the pedestrian, this being 7. The newly created scenario with the added pedestrian to it is saved in the folder where the other scenarios are with the name *rimea_06_corner_new.scenario*. After that another test is being conducted. This time we call the `addPedestrian` purposefully without explicitly initializing the id of the pedestrian. The functionality we implemented should create a new pedestrian with the next inline possible id, that being 8. As it can be seen on Figure 10, this is exactly what happens. We take *rimea_06_corner_new.scenario*, add onto it the recently created pedestrian and save the new scenario with the name *rimea_06_corner_new2.scenario*. Now there are three different versions of the sixth RiMEA scenario: the initial one, the one with a single added pedestrian and the one with two added pedestrians. Within the main method also the functionality of the run scenario could be tested, as an console is opened and waits for a scenario as an input, in order to launch Vadere from the terminal. The final result and the scenario after the addition of two pedestrians could be examined in Figures 9,10 and 11

Report on task 4, Integrating a new model

Integration of SIR model in Vadere:

We downloaded vadere-master, also known as the source code, from the website located at `https://gitlab.lrz.de/vadere/vadere` in order to incorporate the SIR model into vadere. Following that, we transferred the files of the SIR Group model to the vadere-master in the areas that we requested. The package name included inside each Java file may be used to determine the places that are wanted. The following is a another explanation for the same thing:

```
SIRGroup.java:  package org.vadere.simulator.models.groups.sir
SIRGroupModel.java:  package org.vadere.simulator.models.groups.sir;
SIRType.java:  package org.vadere.simulator.models.groups.sir;
```

In light of this, the aforementioned files (or packages) ought to be present in the path if we are to consider the packages. Consequently, it is simpler to copy the files to the desired folder by following the UML diagram that can be found below in Figure 12

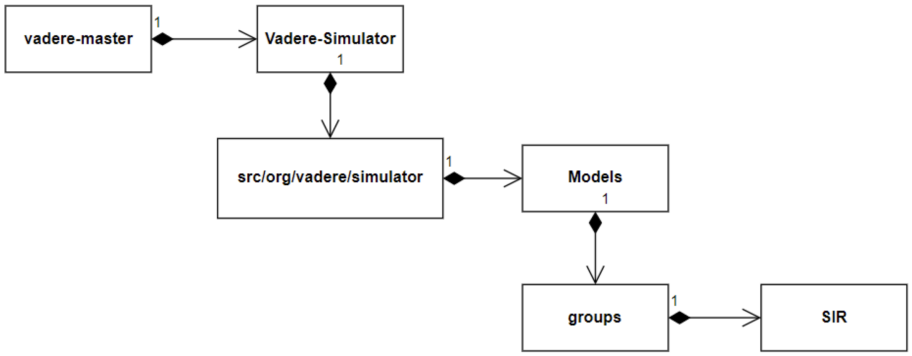

Figure 12: UML diagram for files SIRGroup, SIRGroupModel and SIRType.java

Similarly, the following files are added as per the package mentioned.

```
AbstractGroupModel.java :  package org.vadere.simulator.models.groups;
AttributesSIRG.java :  package org.vadere.state.attributes.models;
FootStepGroupIDProceesor.java :  package org.vadere.simulator.projects.dataprocessing.processor;
```

Simulation analysis:

To run simulation, we use the 2 scenarios that we had to create for task 4_5.

1. Construct a scenario as shown in Figure 17.
2. Construct a corridor scenario of 40m × 20m, where one group of 100 people moves from left to right and another one (also 100) moves right to left 19.

To use the SIRG model for situations, we add the AttributesSIRG to each one. Simply enter AttributesSIRG on the Model tab of the GUI, as shown in Figure 13, or do the same with a scenario file, as shown in Figure 14.

We need pedestrian groupIDs in each time step so that we can analyze/monitor the number of susceptible and infected people over time. We utilize the Processor - FootStepGroupIDProcessor from the Data Output tab in gui to get this (as seen in Figure 15). This produces the desired results in the SIRinformation.csv file. (Figure 16 shows a snapshot of the file.)

S, I, R groups Visualizations

To view S, I and R groups in different colours, we use the method getGroupColor in VadereGui/src/org/-vadere/gui/components/model/SimulationModel.java. Using different values set for ped.GroupIds().getFirst(), we define different colours for SIR groups.. Following are the values:

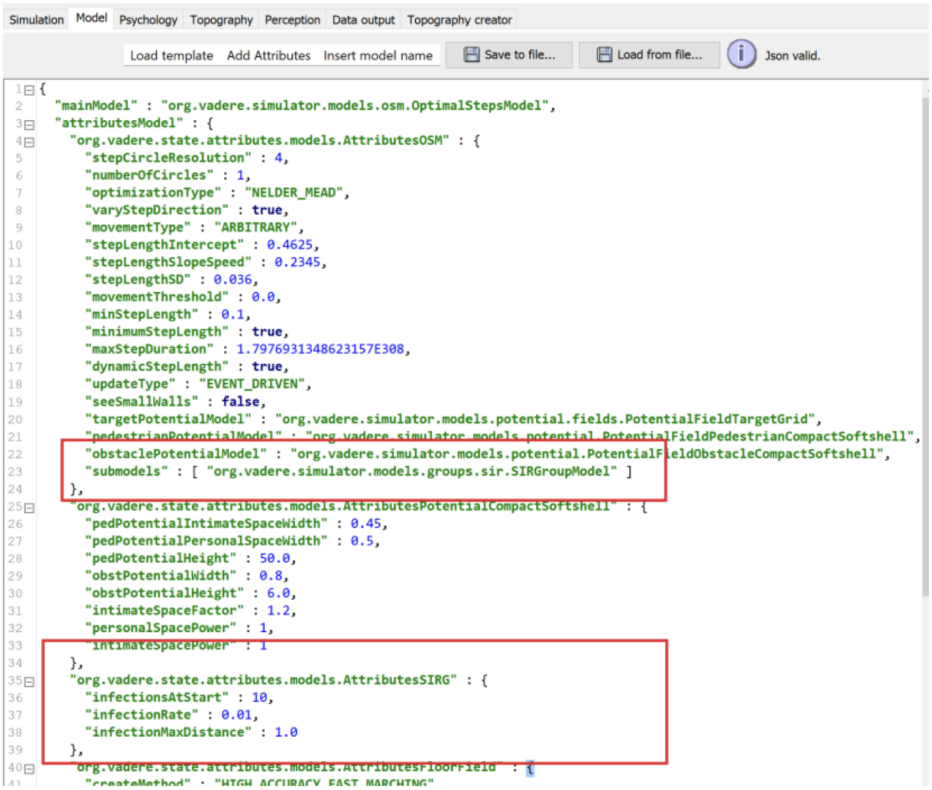

Figure 13: Adding AttributesSIRG to the model in GUI directly.

Infected Pedestrians
Susceptible Pedestrians
Recovered Pedestrians

LinkedCellsGrid Implementation:

We use `LinkedCellsGrid` in org.vadere.util.geometry to speed up the distance computation. Without `LinkedCellsGrid`, all pedestrians check all other pedestrians, and there is no efficient data structure for only checking immediate neighbors. Instead of storing the actual list, we use references with `LinkedCellsGrid`. As a result, the step is reduced to simply finding the references, giving us additional performance gains.

Task 4.5 tests:

- Please see figure 2 for the topography of Scenario1. Following are the simulation images : Figure 17 for the starting infection state and also 18 for the infection states in the end. Following is the S, I visualization plots for different infection rates (0.01, 0.1) plotted together.
- For above scenario, the time take to infect half the population is as mentioned in Table 1.

Infection Rate	Time to infect half Population
0.01	125
0.1	<= 10

- Please see figure 3 for the topography of Scenario2. Following are the simulation images 19 in the begging and then 20 after they cross with each other. Following is the S, I visualization plots for different infection rates (0.05,0.15) plotted together.

  For scenario2, the number of pedestrians infected in counter flow are as follows:

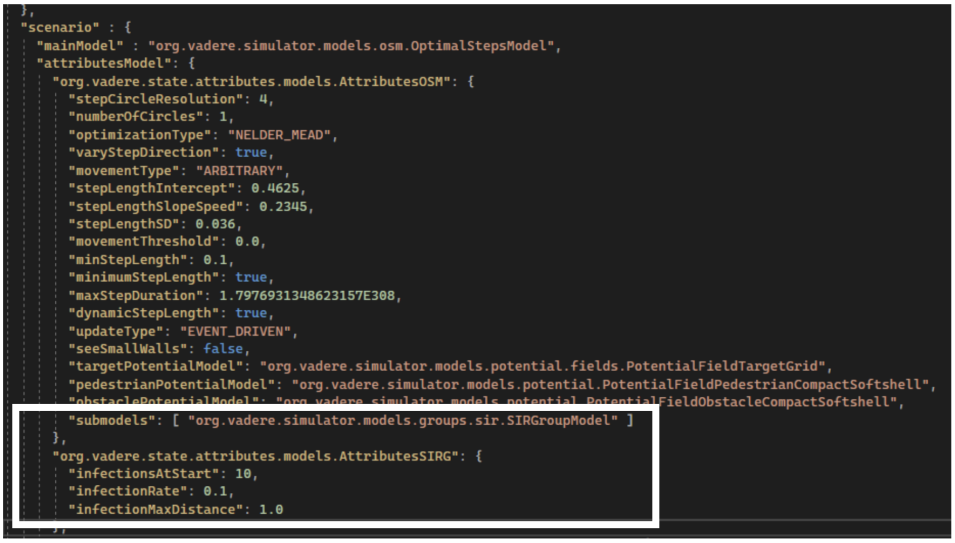

Figure 14: Adding AttributesSIRG in the scenario file

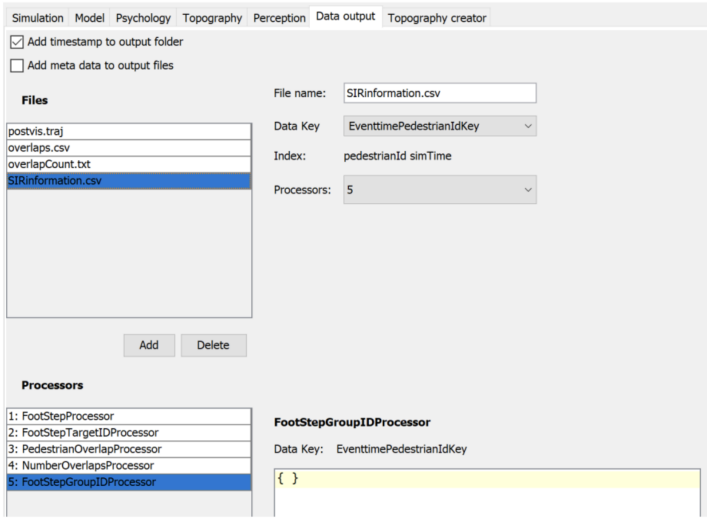

Figure 15: Using FootStepGroupIDProcessor from Data Output tab

Infection Rate	Time to infect half Population
0.05	140
0.15	200

Decoupling Infection rate and time step:

Using stochastic modelling, we obtain infection rate as a function of In the number of new infections since previous sampling, I is number of infected individuals, S is number of susceptible individuals and T is the Sampling interval.

$$\beta = \frac{-\log\left(1 - I_N/I\right)}{T\,S/N} \tag{1}$$

$$\beta = -\frac{1}{T}\log\left(1 - I_N(\frac{1}{I} + \frac{1}{S})\right) \tag{2}$$

This allows us to have similar infection rates for different values of time steps taken (small or large) for simulation.

pedestrianId	simTime	groupId-PID5	
1	0.4	0	
2	0.4	0	
3	0.4	0	
4	0.4	0	
5	0.4	0	
6	0.4	0	
7	0.4	0	
8	0.4	0	
9	0.4	0	
10	0.4	0	
11	0.4	1	
12	0.4	1	
13	0.4	1	
14	0.4	1	
15	0.4	1	
16	0.4	1	
17	0.4	1	
18	0.4	1	
19	0.4	1	

Figure 16: Details of Group ID obtained from SIRinformation.csv

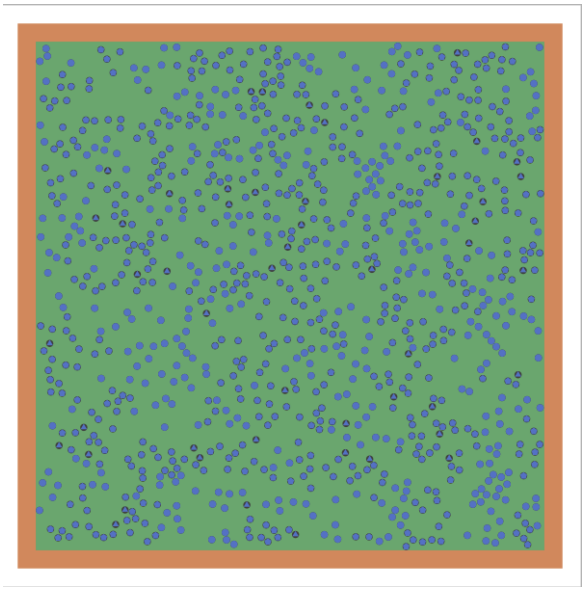

Figure 17: Snapshot for the first scenario with the initial infections

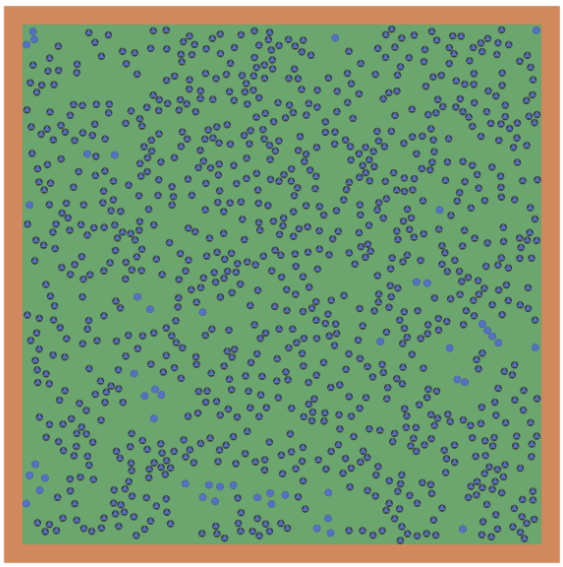

Figure 18: Snapshot for the first scenario with the infections in the end

Possible Extensions:

- The Susceptible, Infected, and Recovered (SIR) models make the fundamental assumption that environmental factors remain constant, specifically that infected and susceptible populations are distributed evenly. However, as demonstrated by the COVID-19 pandemic, these assumptions are no longer valid due to the various preventative measures implemented by governments. Travel bans, partial and total lockdowns, quarantines, social distancing, and other non-pharmaceutical interventions are examples of preventative measures. As a result, the use of stochastic modeling is advised.
- Incorporating the fatality rate, D, into the SIR model. The sickness that killed the people you removed

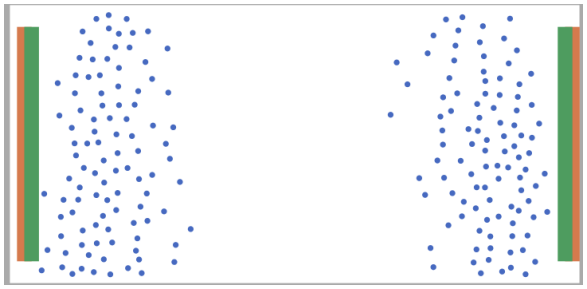

Figure 19: Snapshot for the second scenario of task 4 in the begging

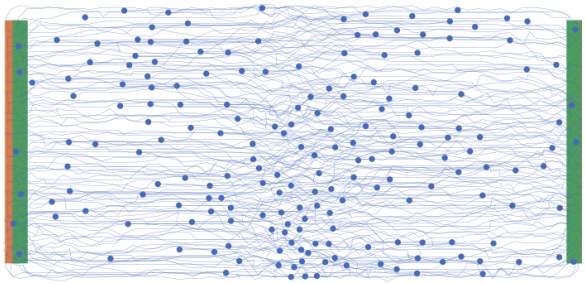

Figure 20: Snapshot for the second scenario of task 4 in the end

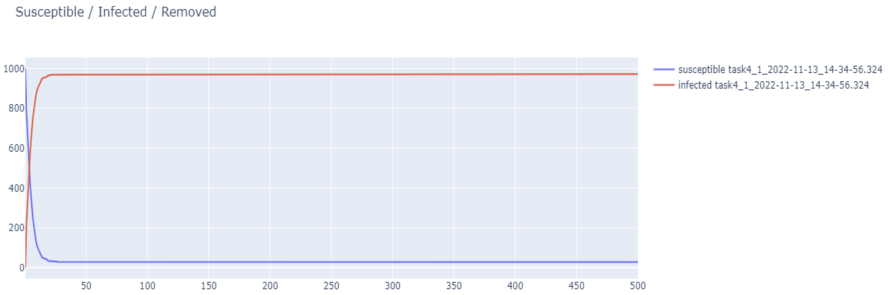

Figure 21: S,I visualization for scenario1 (as mentioned in section .1 of the report)

was spread by the removed pedestrians. As a result, we may use the model to perform predictive analysis on the number of deaths caused by the virus.

- Defining a fourth state as Vaccinated for pedestrians who have been vaccinated. This changes the probability of being infected. Also, recovery rate can be different for vaccinated pedestrians. This allows us to create a model as close to real natural cases as possible.

Report on task 5, Tests

This task requires implementation of code for the recovered state in both Vadere SIR model and the visualization software. The main contribution to the group model was the infected to recovered coin flip shown bellow:

```
if(groupId == SIRType.ID_INFECTED.ordinal())
    if(this.random.nextDouble() < attributesSIRG.getRecoverRate()) {
      elementRemoved(ped);
      assignToGroup(ped, SIRType.ID_RECOVERED.ordinal());
}
```

In the visualization software one needed to add a check for conversion from infected to recovered and properly return and store the data for the recovered group as well:

```
if g != current_state and g == ID_RECOVERED and current_state == ID_INFECTED:
    current_state = g
    group_counts.loc[group_counts['simTime'] > st, 'group-i'] -= 1
    group_counts.loc[group_counts['simTime'] > st, 'group-r'] += 1
    break
```

The simulation results are shown in the following plots:

As one would expect, a non-zero recovering rate would cause all infected people to become recovered at some point. A very high recovery rate prevents the infection from spreading. High infection rate merely causes the population to get infected fast but can not cause the population to be infected at the end of the simulation.

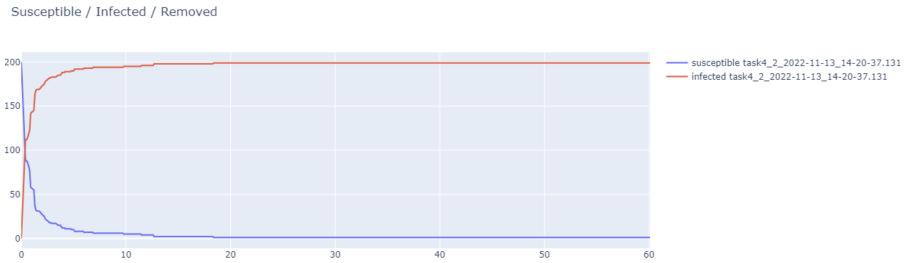

Figure 22: S,I visualization for scenario2 (as mentioned in section .2 of the report)

```
public void update(final double simTimeInSec) {
    // check the positions of all pedestrians and switch groups to INFECTED (or REMOVED).

    DynamicElementContainer<Pedestrian> pedContainer = topography.getPedestrianDynamicElements();
    LinkedCellsGrid<Pedestrian> cellsGrid = new LinkedCellsGrid(topography.getAttributes().getBounds().x,
            topography.getAttributes().getBounds().y,
            topography.getAttributes().getBounds().getWidth(),
            topography.getAttributes().getBounds().getHeight(),
            1000);
    if (pedContainer.getElements().size() > 0) {
        for (Pedestrian ped : pedContainer.getElements()) {
            cellsGrid.addObject(ped);
        }
        //Task4.4 Implementation of LinkedCellsGrid
        for (Pedestrian ped : pedContainer.getElements()) {
            int groupId = getGroup(ped).getID();

            for (Pedestrian ped_neighbour : cellsGrid.getObjects(ped.getPosition(), attributesSIRG.getInfectionMaxDistance())) {
                if (ped == ped_neighbour || getGroup(ped_neighbour).getID() != SIRType.ID_INFECTED.ordinal())
                    continue;
                if (this.random.nextDouble() < attributesSIRG.getInfectionRate()) {
                    if (groupId == SIRType.ID_SUSCEPTIBLE.ordinal()) {
                        elementRemoved(ped);
                        assignToGroup(ped, SIRType.ID_INFECTED.ordinal());
                    }
                }
            }
```

Figure 23: Adding LinkedCellsGrid to SIRGroupModel

The supermarket scenario is shown in the figure 6. There are 100 people generated over time of which the first 25 are ill from the beginning.

Doubling the space between people significantly reduces the infections even when the supermarket is quite crowded.

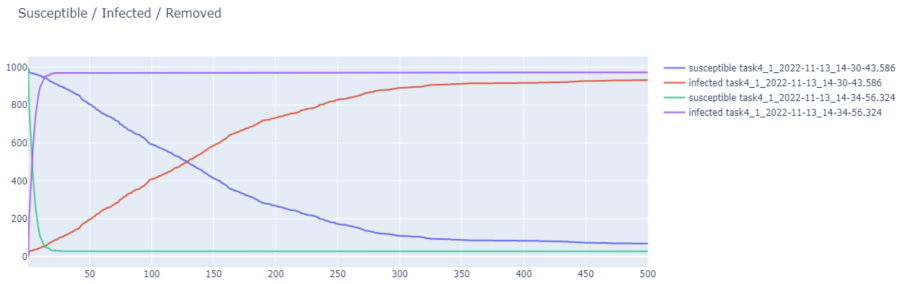

Figure 24: S,I visualization plot for scenario1 at infection rates (0.01,0.1)

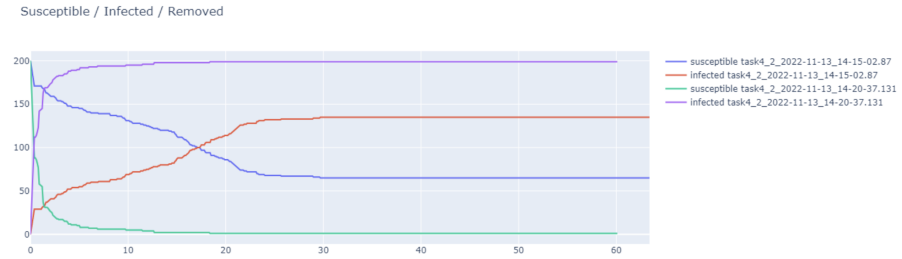

Figure 25: S,I visualization plot for scenario1 at infection rates (0.05,0.15)

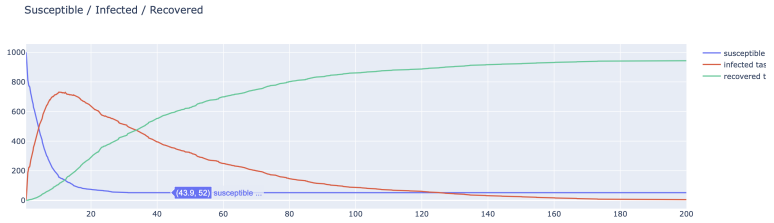

Figure 26: infectionRate=0.1, recoverRate=0.01

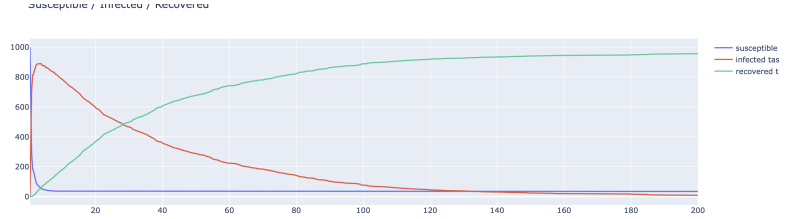

Figure 27: infectionRate=0.3, recoverRate=0.01

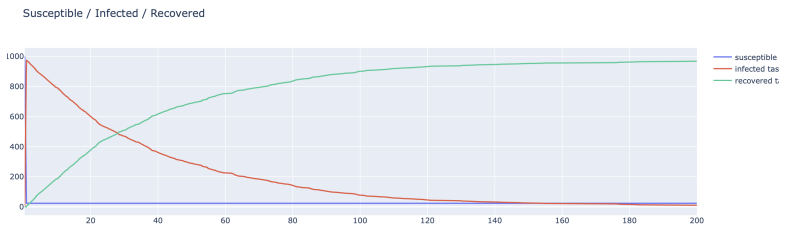

Figure 28: infectionRate=0.7, recoverRate=0.01

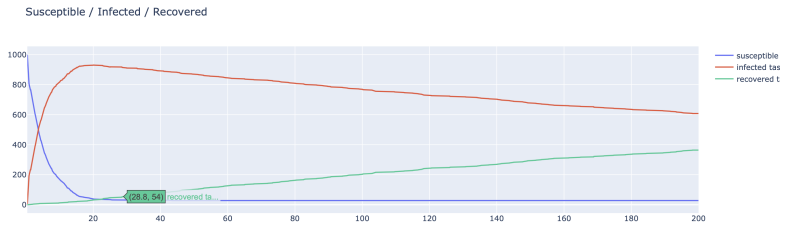

Figure 29: infectionRate=0.1, recoverRate=0.01

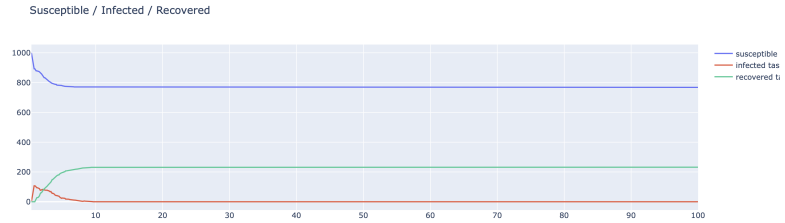

Figure 30: infectionRate=0.1, recoverRate=0.3

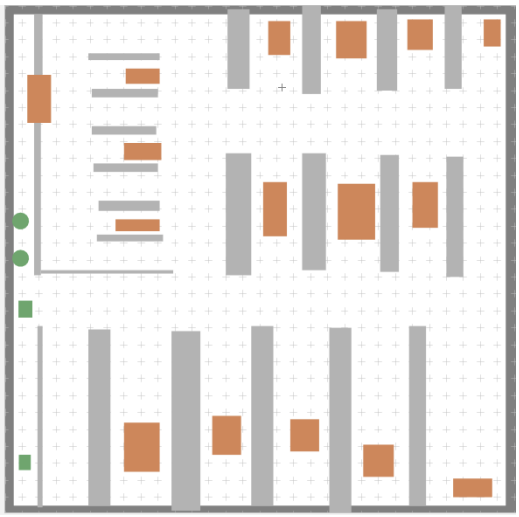

Figure 31:

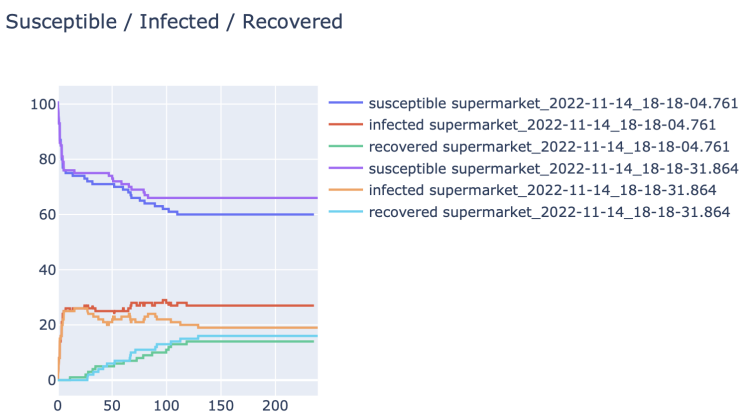

Figure 32: infectionRate=0.01, recoverRate=0.005, pedPotentialPersonalSpaceWidth=1.2 and 2.4

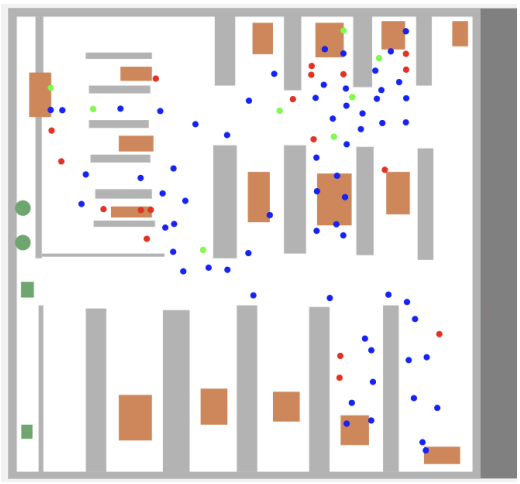

Figure 33:

References

[1] Download the latest java lts free.

[2] What is vadere?

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

[4] Dirk Helbing and Peter Molnar. *Social force model for pedestrian dynamics*. 05 1995.

[5] Gerta Köster, Franz Treml, and Marion Gödel. Avoiding numerical pitfalls in social force models. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 87:063305, 06 2013.

[6] von RiMEA, Von Angelikakneidl, and Von Anwinkens. Rimea e.v., Oct 2022.

[7] Michael Seitz and Gerta Köster. Natural discretization of pedestrian movement in continuous space. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 86:046108, 10 2012.

[8] Michael J. Seitz, Felix Dietrich, and Gerta Köster. The effect of stepping on pedestrian trajectories. *Physica A: Statistical Mechanics and its Applications*, 421:594–604, 2015.

[9] Michael J Seitz and Gerta Köster. How update schemes influence crowd simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2014(7):P07002, jul 2014.