Report for exercise 2 from group A

Tasks addressed:	5
Authors:	MELIH MERT AKSOY (03716847)
	ATAMERT RAHMA (03711801)
	ARDA YAZGAN (03710782)
Last compiled:	2022–05–25
Source code:	https://github.com/mlcms-SS22-Group-A/mlcms-SS22-Group-A/tree/exercise-2

The work on tasks was divided in the following way:

MELIH MERT AKSOY (03716847)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
ATAMERT RAHMA (03711801)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
ARDA YAZGAN (03710782)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%

Report on task Task 1/5: Setting up the Vadere environment

AdoptOpenJDK	11.0.15
IntelliJ IDEA	2022.1.1

Table 1: Software versions.

In this task we have set up the `Vadere` software by installing the compiled `JAR` files from `http://www.vadere.org/releases/` (master branch version). We have also installed AdoptOpenJDK to be able to run the software (see Figure 1 above for the version we have downloaded for this exercise).

As described in the exercise sheet, we have recreated the straight line and corner scenarios from RiMEA guidelines (to be exact scenarios 1 and 6 from [5]) and the "chicken test" scenario, and as suggested, we did use the standard template of Optimal Steps Model (OSM, [6]) to simulate the scenarios. We have used the exact same configurations as in our last exercise (such as number of pedestrians, length and height of obstacles, source and target locations etc.) to be able to compare the Vadere software with our cellular automaton implementation.

Visualization has differences between our implementation and Vadere software. In Vadere users are given the option to visualize the trajectory or the look direction of the pedestrian, they can also do a post-simulation visualization where they can scroll through the timesteps, other than that both implementations create somewhat similar outputs. There are sources where pedestrians start to walk (coloured green in Vadere) and targets they want to reach (coloured orange in Vadere), users can also add obstacles to the scenarios (coloured gray in Vadere) in different shapes, which are inaccessible by pedestrians, pedestrians are depicted by blue circles that move around the scenario.

Another key difference between the implementations is that Vadere software creates a much more continuous visualization output than our own output. In our optimization the scenario consists a grid and this grid is split into discrete cells, where each cell can have one of the four following values at a time: empty (E), pedestrian (P), obstacle (O) and target (T). In Vadere, however, pedestrians can have positions that are described by floating point values, which means the discretization in Vadere software is much more relaxified than in our implementation.

Coming to the user interface and scenario editing, our implementation gave the users an option to create their scenarios using `.txt` files, where they can input for each cell in the grid, whether they want it empty, place a pedestrian, target or an obstacle in it. Vadere software uses mainly JSON file type to save the topography of a specific scenario, where the objects themselves and their important attributes are located. It is also possible to use the visualization of the scenario itself to add items into the scenario (using the buttons in the toolbar, users can add sources, single pedestrians, obstacles and targets).

Users are also able to setup a speed distribution for the pedestrians, where in our implementation users are able to set specific speed values for each pedestrian and if they do not choose to do so, a default value of 1.6 will be assigned to each pedestrian (as it is the average walking speed of people [4]). We did not change the template values in our simulations, which are explicitly given in the following:

- `speedDistribtuionMean` : 1.34
- `speedDistributionStandardDeviation` : 0.26
- `minimumSpeed` : 0.5
- `maximumSpeed` : 2.2
- `acceleration` : 2.0

The output files that Vadere software creates at the end of a simulation run is a feature that is not implemented in our cellular automaton software. The output directory for a scenario contains a .traj file, where for each timestep in the simulation, the starting and end position are given for each of the pedestrians. We also did use a clock in the background to measure how much time it takes for pedestrians to reach the target, but Vadere offers more information in each time-step.

In the following subsections instead of analyzing and stating the general similarities and differences between our cellular automaton implementation and Vadere software, we will go into more detail about the results of different scenarios run in both simulation softwares.

Straight line scenario

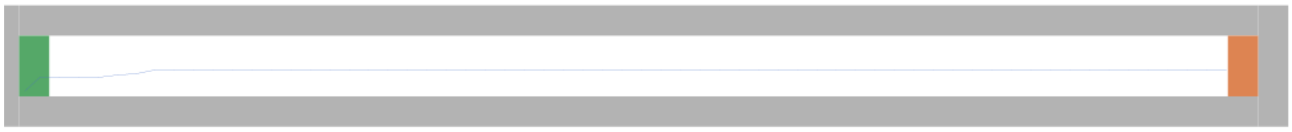


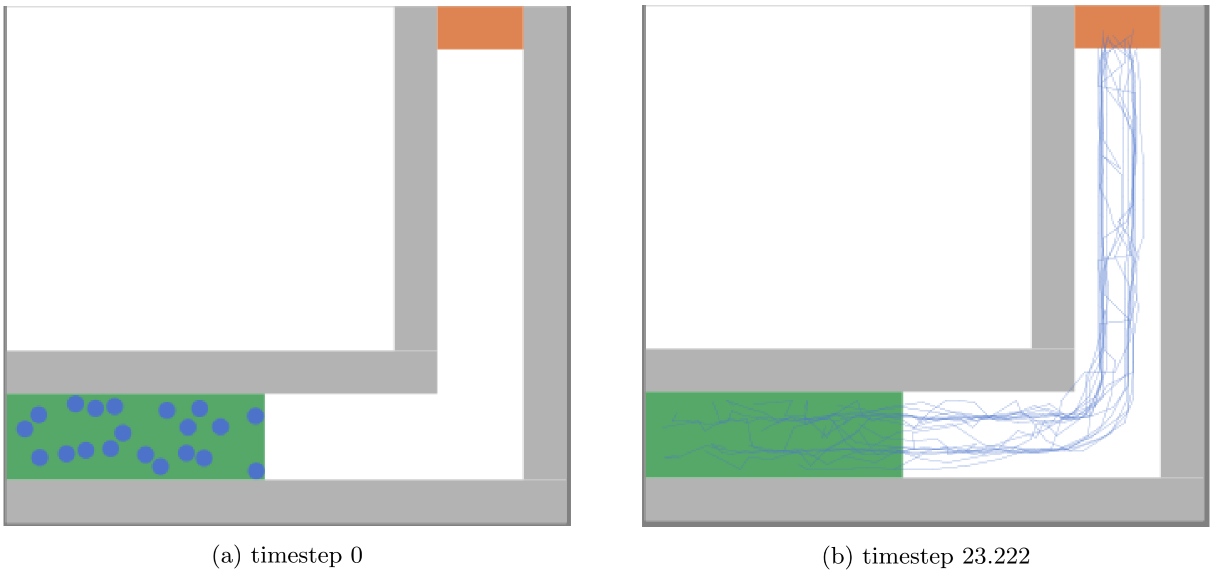
Figure 1: RiMEA straight line scenario simulated by `Vadere` software. The blue line represents the trajectory of the pedestrian and the target is reached after 42.502 seconds.

This is the first RiMEA scenario, where a pedestrian walks in a straight hall that is 40m wide and has a height of 2m. A single pedestrian is created from a source, the area of the source or the exact start position of the pedestrian is not given in RiMEA guidelines, so in this scenario, a 1 by 2 meters source is used, where pedestrians spawn location is randomised. Target area's left edge is at meter 41 so that pedestrian walks approximately 40m (if it was placed at 39, pedestrian would walk around 39 meters, since she stops after reaching the target).

In Figure 1 a snapshot of the last time-step is given. The resulting visualization looks similar to the one that we have provided in our last report, where here we can also see the trajectory of pedestrian and make observations about her pathing. Looking at the trajectory we can say that pedestrian spawned at the lower half of the source and aimed to finish her walking right in the middle of the target area.

One thing to be noticed is that the time that it takes for the pedestrian to walk towards the target is about %25 more than the one she needed in our simulation, which was approximately 31 time-steps, and since we counted every time-step as a second, 31 seconds to be precise. This difference is mainly caused by the implementation decisions of speed in our software versus Vadere. As explained earlier Vadere does not assign a specific speed value that is either given by user or a default value to pedestrians. It distributes speed across pedestrians according to the given variables such as mean value, standard deviation, minimum value and maximum value, acceleration etc. Since the speed of pedestrian is not exactly 1.33 m/s in each simulation run, it is expected that we get faster or slower completion times compared to our implementation and test results. What matters is the average completion time, which checks out with the given RiMEA guidelines and test results in the case of using Vadere software as well.

Corner scenario



(a) timestep 0

(b) timestep 23.222

Figure 2: `RiMEA` corner scenario simulated by `Vadere` software. On the left figure we can see the initial setup of the scenario, where each pedestrians starting location is also shown (blue circles). The blue lines in the right figure represents the trajectory of the pedestrians. The target is reached by all of the pedestrians after 23.222 seconds.

This is the sixth RiMEA scenario, where multiple pedestrians try to move around a corner, the main aim of this scenario is to observe the behaviour around a corner point. In this configuration 20 pedestrians are created from a source, the source is 6m wide and has a height of 2m, where pedestrians spawn location is randomised using a distribution. Pedestrians walk in a hall that is 2m wide, which is shown in figure 2 in more detail. The target is at the end of the hall, and to be able to reach it, all pedestrianns must successfully move around the corner without passing through obstacles.

In Figure 2 a snapshot of the first- and last time-step of the simulation is given. Here we can see a different approach that pedestrians take in comparison to our cellular automaton implementation: In our implementation all of the pedestrians were forming a single line of people where there were sometimes even spaces between pedestrians, however as depicted in the figures as well, in the OSM model simulation, pedestrians seems to split into two different groups, where one group walks on the left side of the hallway and the other one on the right side. This behaviour does not change even during the movement around the corner and afterwards. It is again to be stated that in Vadere software the simulation took a longer time to finish in this example run, however as stated above in the first comparison, this is mainly due to the speed distribution feature of Vadere.

Chicken test scenario

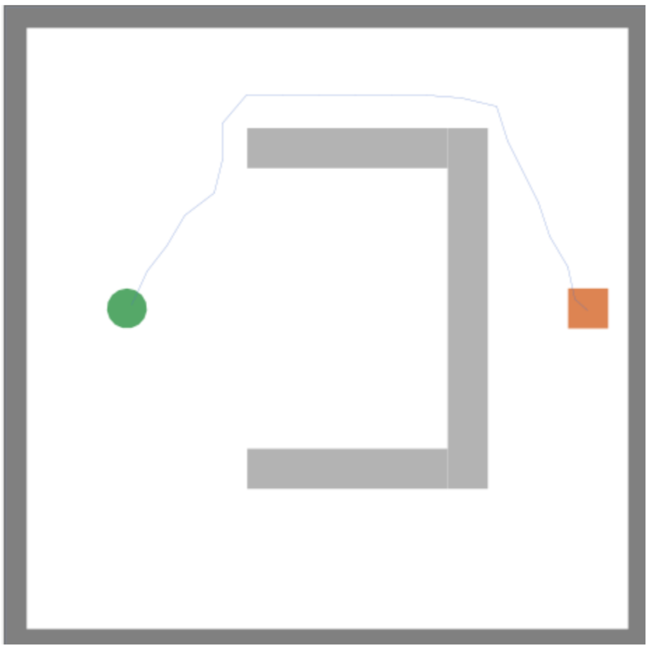


Figure 3: Chicken test scenario simulated by `Vadere` software. The blue line represents the trajectory of the pedestrian and the target is reached after 11.537 timesteps.

This is a test scenario that we self have created, where a single pedestrian tries to move around a U-shaped obstacle, the main aim of this scenario is to observe the behaviour of pedestrian and make sure he does not get stuck in the wrong places. In this configuration pedestrian is placed on the left side of the scenario, target is on the direct opposite of the pedestrian and in between there is a U-shaped obstacle where the opening of U is pointing towards the pedestrian (see Figure 3). In Vadere we have recreated the exact same configuration to see whether the behaviour around a U-shaped obstacle stayed the same.

The figure 3 shows a snapshot of both the scenario setup and the trajectory of the pedestrian. We have used Optimal Steps Model (OSM) during this run as all the other scenarios and we can clearly see that the pathing of the pedestrian is very similar to the one that we have observed during our tests in the cellular automaton implementation. Pedestrian does not directly move to the space between the obstacles, where she would actually get stuck, but rather moves upwards to be able to move around the whole obstacle. It is to be noted that in strictly turns right after he realizes that he has left the obstacle behind, which also shows that he is not so uncomfortable moving close to the obstacles. The target is reached by the pedestrian in 11.537 seconds, which is similar to the one that we have recorded on our implementation.

Report on task Task 2/5: Simulation of the scenario with a different model

Optimal Steps Model vs. Social Force Model

We do not observe any difference between the two models in the straight line scenario. The reason for that is probably the missing interaction between pedestrians as there is only one pedestrian that moves towards the target. In addition, as there is no obstacle in this scenario, the walking trend of the pedestrian is not changed during the simulation, which again contributes to the result that both scenarios end up in the (almost) same trajectory with the close simulation time.

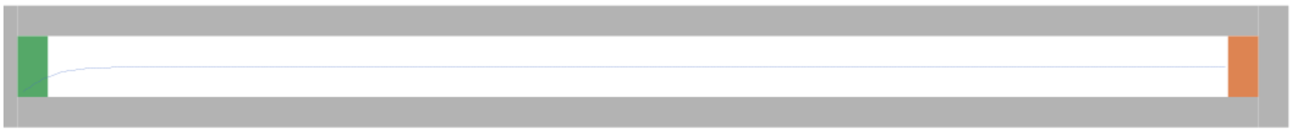
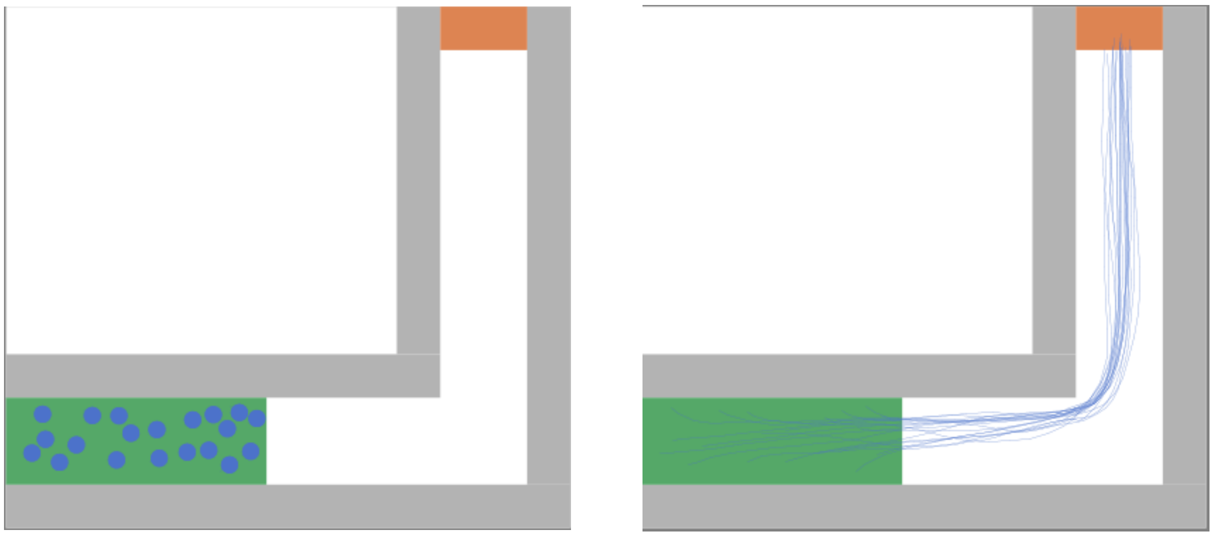


Figure 4: RiMEA straight line scenario simulated by `Vadere` software using social force model. The blue line represents the trajectory of the pedestrian and the target is reached after 41 timesteps.



(a) timestep 0 (b) timestep 22.4

Figure 5: `RiMEA` corner scenario simulated by `Vadere` software using social force model. The blue line represents the trajectory of the pedestrians the target is reached after 22.4 seconds.

When we compare the trajectories of both Optimal Steps Model and Social Force Model for the corner scenario the most significant difference is that in the Social Force Model, a more dense trajectory is observed where pedestrians move closer to each other compared to optimal steps model. The tendency to move closer especially in the beginning of the trajectory can be possibly explained with the attractiveness effect in social force model. As all of the pedestrians are moving to the same target which then leads to the same direction of the movement causes pedestrians to be "attracted" (friends etc.) of each other. This attractiveness but declines with the increasing time which explains the loosened density of the pedestrians at the end. The increased density at the beginning decreases the velocity of the pedestrians in contrast to optimal steps model and results in a longer simulation time.

The trajectory of social force model with the chicken scenario seems to be more rounded compared to the optimal steps model. As the social force model assumes that the pedestrian feels more uncomfortable when they are closer to wall/obstacle. That's why the pedestrian can not go at the direct path when they arrive at the top right corner of the obstacle. Instead they prefer a longer way to reach the target which then in turn

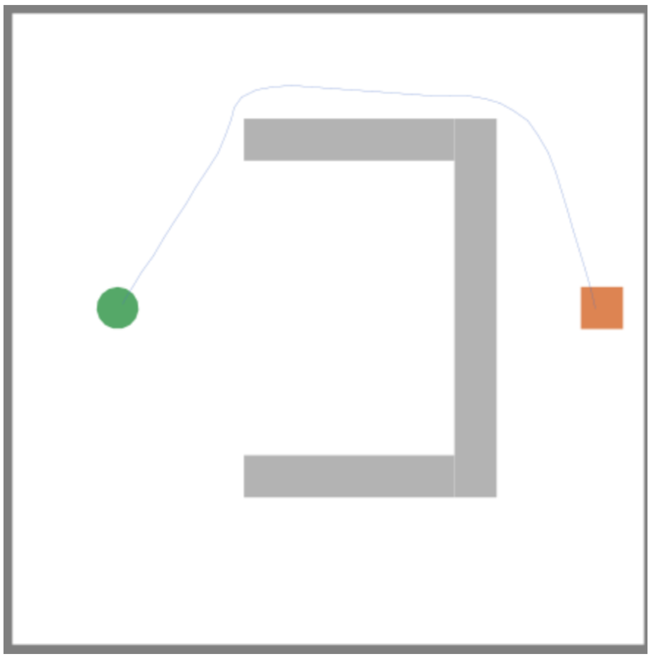


Figure 6: Chicken test scenario simulated by `Vadere` software using social force model. The blue line represents the trajectory of the pedestrian and the target is reached after 16 seconds.

make the simulation time again longer.

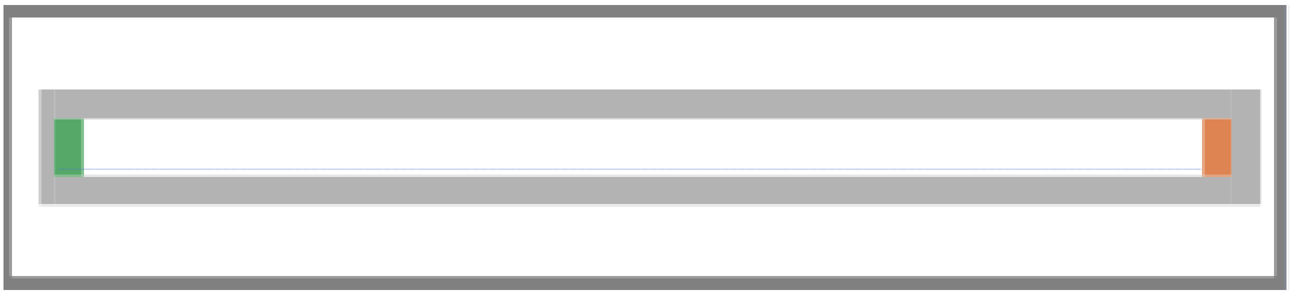


Figure 7: RiMEA straight line scenario simulated by `Vadere` software gradient navigation model. The blue line represents the trajectory of the pedestrian and the target is reached after 41.7 seconds.

As in the Gradient Navigation Model forces are not considered [1], the pedestrians probably do not feel uncomfortable when they move near to the obstacles/wall. Besides they still want to reach the target in the shortest path and in the exercise-paper it is not mentioned that the pedestrians want to keep a space for movement for themselves. This explains the shape of the trajectory in all scenarios in which the pedestrians' positions are in general closer to the obstacles. Although the trajectory seems to be the shortest one among all other models, one can think that the scenarios with Gradient Navigation Model should have the shortest simulation times, but there is an important thing that change in position of pedestrians is not instantaneous and it is assumed that there is a certain time delay [2] when pedestrians change their positions. Another reason for longer simulation times is that as the social forces are not considered in this model they follow a dense trajectory throughout the whole simulation which then decelerates the pedestrians and result in a slower average speed.

Report on task Task 3/5: Using the console interface from Vadere

Figure 10 shows the result of the simulations (pedestrian trajectories). Figure (a) is run using the GUI and figure (b) using the CLI of the `Vadere` software. We have used the same corner scenario that we have used before with the Social Force Model.

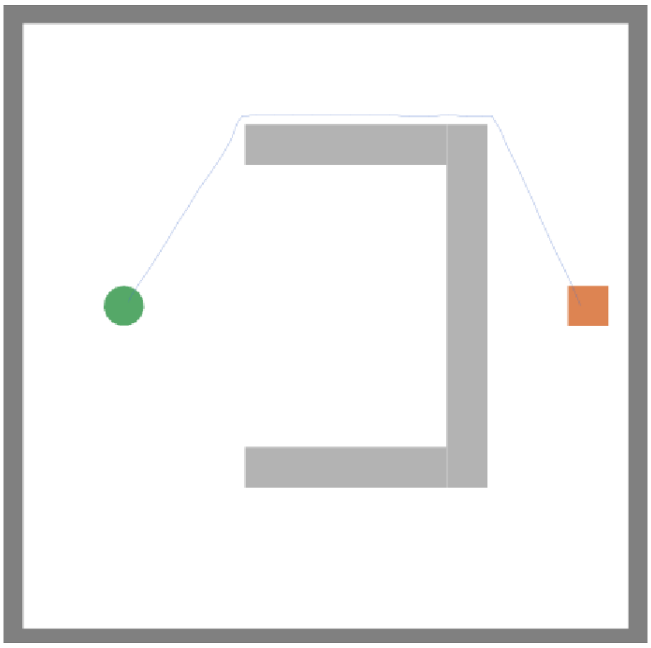
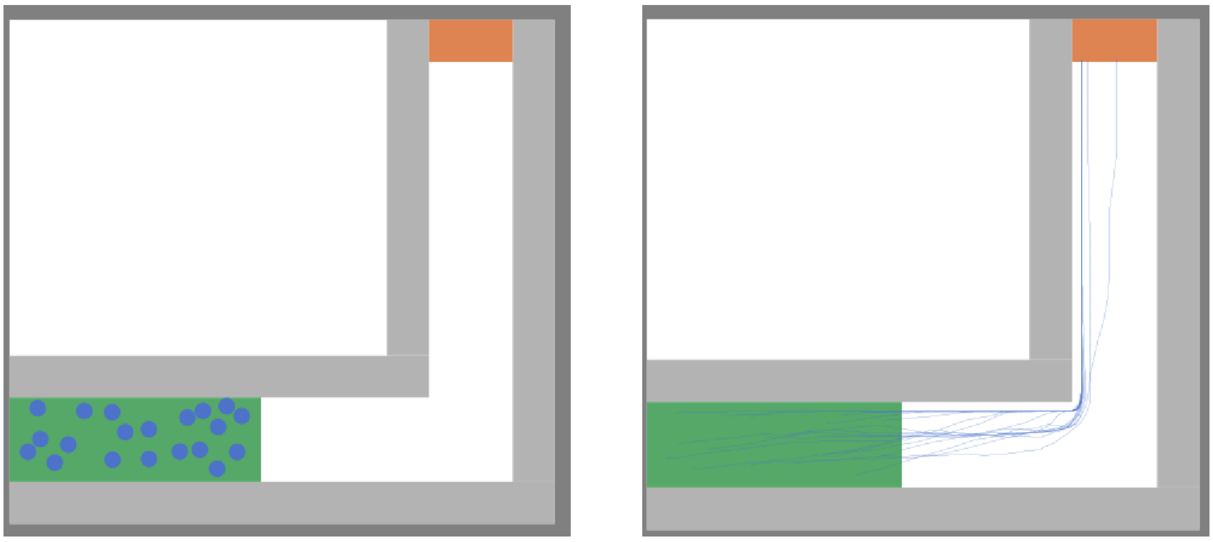


Figure 8: Chicken test scenario simulated by `Vadere` software using Gradient Navigation Model. The blue line represents the trajectory of the pedestrian and the target is reached after 15.6 seconds.



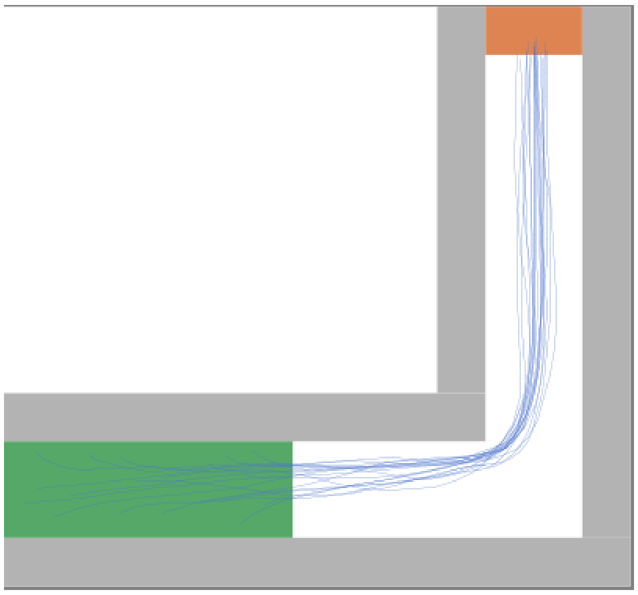
(a) timestep 0

(b) timestep 23.222

Figure 9: `RiMEA` corner scenario simulated by `Vadere` software using Gradient Navigation Model. The blue line represents the trajectory of the pedestrians the target is reached after 30 seconds.

Using the CLI is a bit more abstract since one does not have the option to visualize the topography directly, as one can do in the GUI. In the CLI the user should give the scenario file that she wants to simulate, as well as one output directory. The scenario is then simulated and the output files are stored in the output directory as usual.

As one can see in the figure, they both look similar. We have used the `diff` command in the command line to compare the output files *(postvis.traj)* line by line. The number of differing lines of the output files of the GUI and CLI runs can be seen in figure 11 below. As expected both of the runs produce the same pedestrian trajectories.


(a) timestep 22.4

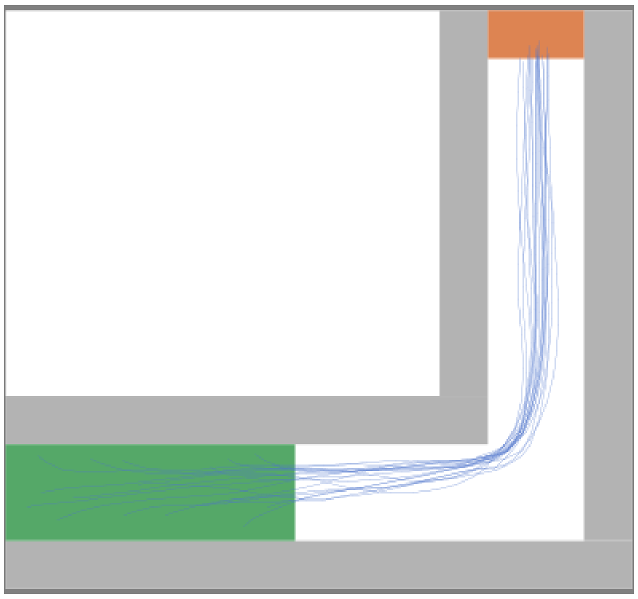
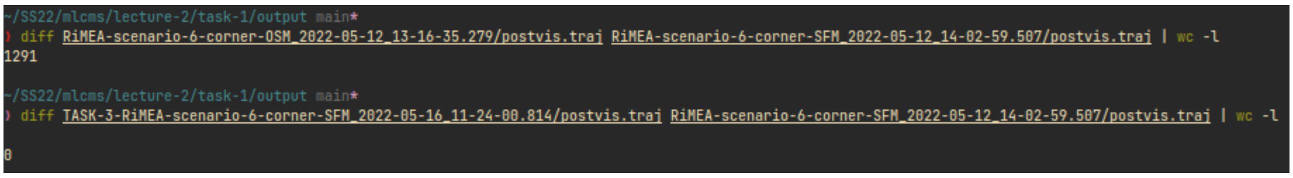

(b) timestep 22.4

Figure 10: `RiMEA` corner scenario simulated by `Vadere` software using social force model. The blue line represents the trajectory of the pedestrians. The target is reached after 22.4 seconds. Figure (a) is run using the GUI. Figure (b) is run using CLI.



```
~/SS22/mlcms/lecture-2/task-1/output main*
❯ diff RiMEA-scenario-6-corner-OSM_2022-05-12_13-16-35.279/postvis.traj RiMEA-scenario-6-corner-SFM_2022-05-12_14-02-59.507/postvis.traj | wc -l
1291

~/SS22/mlcms/lecture-2/task-1/output main*
❯ diff TASK-3-RiMEA-scenario-6-corner-SFM_2022-05-16_11-24-00.814/postvis.traj RiMEA-scenario-6-corner-SFM_2022-05-12_14-02-59.507/postvis.traj | wc -l

0
```

Figure 11: 1291 is the number of differing lines of the outputs of the corner scenarios that are run using the Optimal Steps Model and Social Force Model. There are no differing lines between the GUI and CLI runs of the corner scenario when using the same model.

In order to add a pedestrian to the corner scenario in task 1 we wrote a small python script `add_pedestrian.py`. The script takes the scenario file location as a absolute or relative path as an argument, as well as the x and y-coordinates of the pedestrian to be added. It requires the base `pedestrian.json` file to modify, which can be obtained from the `dynamicElements` field of a random scenario created by the graphical user interface. This file and the script should be placed into the same directory. The script follows the following steps to insert a pedestrian at the given location into the given scenario:

1. Parses the input arguments, and checks them for validity (arguments are mandatory). A usage message is printed if this is not the case.

2. Finds a unique id for the pedestrian to be added. For this we save the ids of obstacles, targets, sources and other dynamicElements instances into a list and sort this list in increasing order. Then we take the last element of this list and add 1 to get a unique id for the new pedestrian.

3. Lastly we assume that the scenario has a single target, and read this target idea from the targets field. We then set the target field of the pedestrian in the `pedestrian.json` file and add the JSON object into the dynamicElements list of the scenario.

4. The given scenario is not modified in the end, but we create a new scenario with a new name – `"_pedAdded"` string appeneded to the end of the given scenario name.

We have used this script to add a new pedestrian at position $(11.4, 2.3)$ which is roughly the location of the red point shown in the exercise sheet. One can see this outlying pedestrian in figure 19.

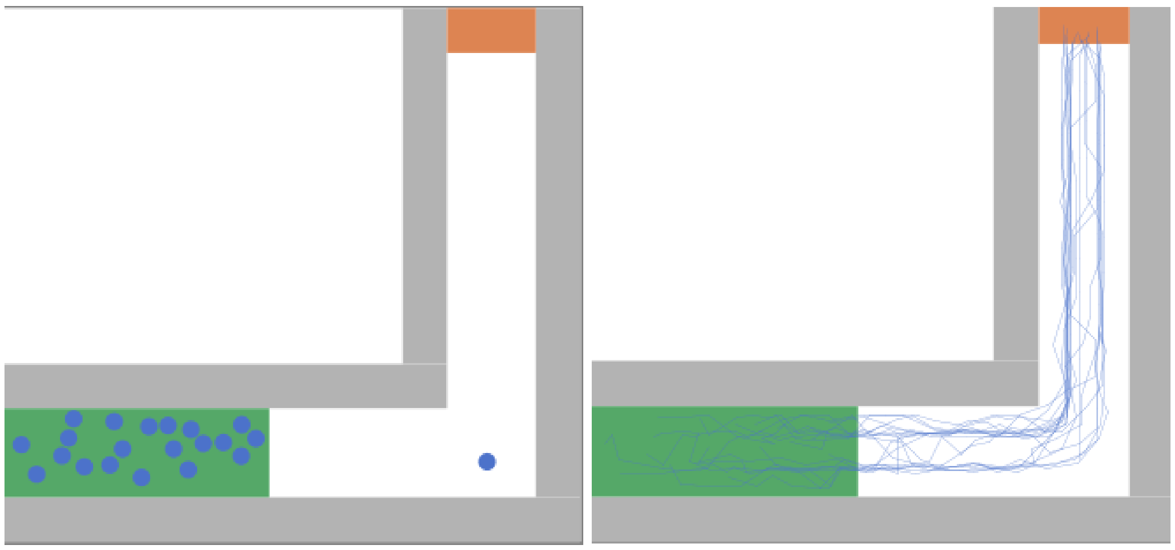

(a) timestep 0 (b) timestep 27.146

Figure 12: `RiMEA` corner scenario with added pedestrian simulated by `Vadere` software using optimal steps model (same scenario that is used in task 1). The blue line represents the trajectory of the pedestrians. The target is reached after 27.146 seconds.

The pedestrians starting in the source field (green area in the figure) have reached the target after 13.44 seconds in average. The outlying pedestrian has reached the target in 6.8 seconds since its initial position has a lower distance to the target.

Report on task Task 4/5: Integrating a new model

For this task we have cloned the Master branch of the `Vadere` software [3]. In the first part of this we have prepared an UML Digram of the SIR model that we have to integrate into `Vadere`. The integration of this model on top of the master branch was very straightforward:

1. modify the Model JSON file and add `SIRGroupModel` to the submodels list of the main model (Optimal Steps Model is used in our case)
2. modify the Model JSON file again and add the following attributes to the `AttributesSIRG` object:
   - `infectionsAtStart` (initial number of infected pedestrians in the scenario)
   - `infectionRate` (the probability of a pedestrian to get infected if there is another infected pedestrian with a smaller distance than the `infectionMaxDistance`)
   - `infectionMaxDistance` (the maximum distance in which the disease may spread)
3. Link a new output file `SIRinformation.csv` to the output processor `FootStepGroupIDProcessor`. The new output processor creates a table consisting 3 values at the columns: `pedestrianId`, `simTime` and `groupID-PID5`. This table contains for each pedestrian that has the given id, the group id of her in the given simulation time, where `groupID-PID5` variable is an integer value, where a 0 indicates susceptible, 1 indicates an infected pedestrian. Using the `python` web application that was provided in Moodle (`python app.py` as console input then opening up a local ip-address on browser) we were able to visualize data that is stored in the stated `.csv` file, where the application depicts in a graph the number of infected and susceptible pedestrians at each time step, or it can even compare different test cases with each other by plotting the results on the same graph.

The diagram is depicted in figure 13. SIR Model models S (susceptible), I (infected) and R (recovered) groups which can be used to simulate the spread of a disease among a group of people. In the `sir` package an

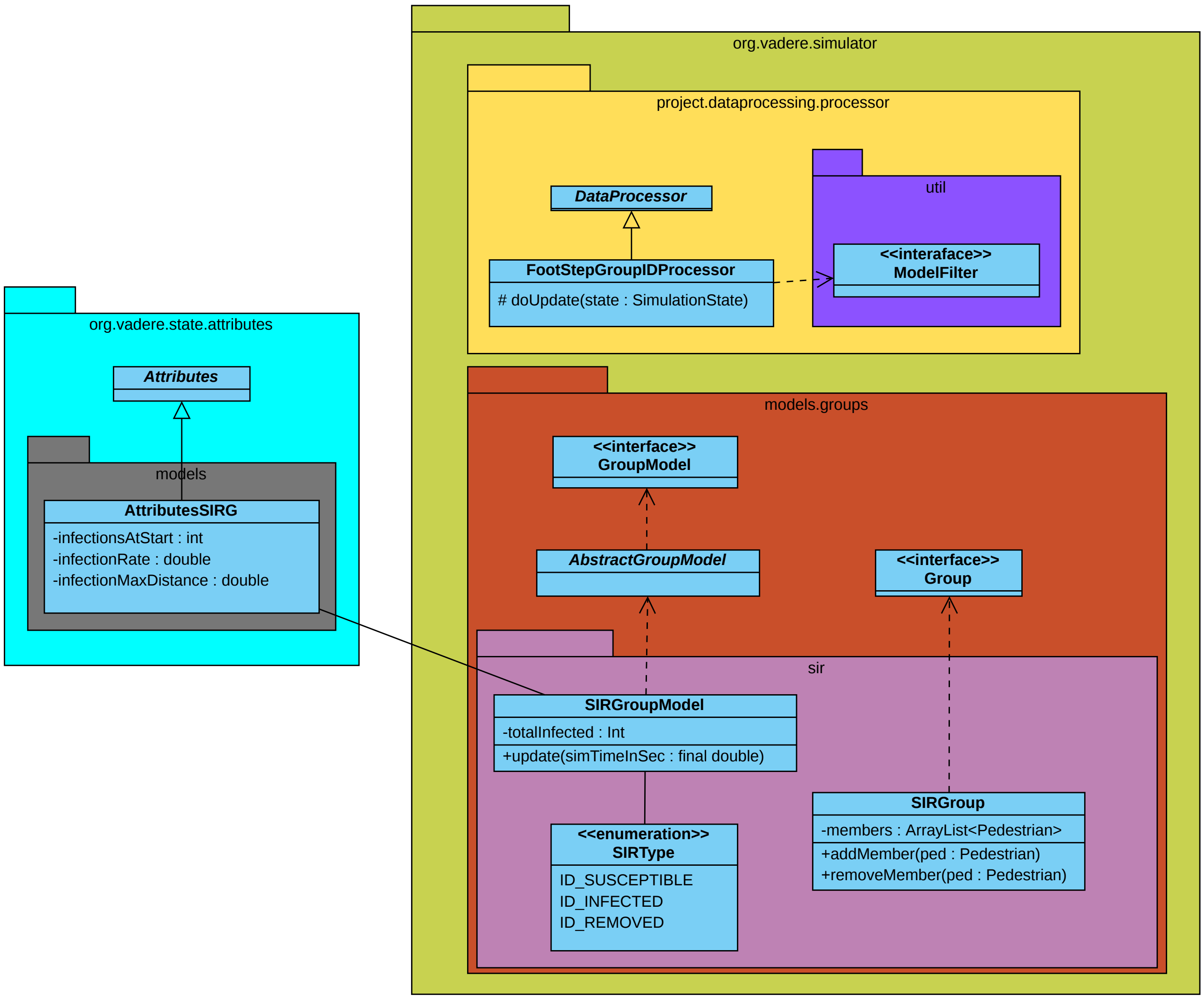


Figure 13: uml diagram of the provided SIR model extension on `Vadere` software.

Enum type is used in the file `SIRType` to present these groups. In the `update` function of the `SIRGroupModel` in the `sir` package all the pedestrians are updated accordingly as infected as explained above. The required attributes are stored in file `AttributesSIRG` in the `attributes` package which are read from the JSON file of the mode. Currently the model does not implement the `recovered` state. So all the pedestrians stay infected forever if they become infected at some time during the simulation. Additionally the number of infected pedestrians are stored in the variable `totalInfected`.

We have modified the `getGroupColor` in the file `SimulationModel.java` as seen in figure 14. We include the figures of the visualization of the group coloring in the next task. We have also removed the rounding of the simulation time in the `doUpdate` function in file `FootStepGroupIDProcessor.java` since rounded times do not work in the post-visualization.

To find neighbours in a more efficient way (rather than checking all other pedestrians for each pedestrian) we have used the suggested `LinkedGridCell` class provided to us in the project. We have extracted all pedestrian objects from the topography, the method we used for this operation (`getSpatialMap()` method of topography attribute in the SIRGroupModel class) already returns an iterable LinkedGridCell object, using that iterable object we were able to access the pedestrians, that are in a circle centered around the pedestrian to whom we want to find neighbours, with a specified radius, in this case "infectionMaxDistance", in a less time and performance consuming implementation (according to the documentation of the class in $O(1)$).

```
int groupId = ped.getGroupIds().getFirst();
Color c = colorMap.get(groupId);
if (c == null) {
  switch (groupId) {
    case 0: c = Color.RED; break;
    case 1: c = Color.BLUE; break;
    case 3: c = Color.GREEN; break;
    default: c = new Color(...));
  }
  colorMap.put(groupId, c);
}
return c;
```

Figure 14: Sets different colors (`RED` for infected, `BLUE` for susceptible and `GREEN` for recovered). The recovered state will be added in the next task.

We have constructed the scenario given in the task 4.5 (but without the recovered state yet). The scenario consist of the following:

- 60 by 60 meters area,
- 1000 static pedestrians (no movement) are randomly distributed in the scenario,
- initially there are 10 infected pedestrians in the scenario.

Visualization of this scenario with the `infectionMaxDistance` and `infectionRate` pairs: $(1, 0.01)$, $(1, 0.03)$, $(1, 0.05)$, $(2, 0.06)$, $(2, 0.09)$ can be seen in figure 15. As one can see, only for the pairs $(2, 0.09)$ and $(2, 0.06)$ the half of the population (500 pedestrians) can get infected in about 3.5 and 7 seconds respectively. So as expected, if we increase the `infectionRate` it takes less time for the half of the population to get infected. And the for the other pairs the number of the infected pedestrians does not increase significantly because they include very low `infectionMaxDistance` or `infectionRate` values.

We have also constructed the following 'corridor' scenario:

- 40 by 20 meters area with two groups of people, one on the right and one on the left,
- one group of pedestrians moves from left to right, and the other group moves from right to left,
- each group consists of 100 pedestrians, generated over time (they do not spawn at the same time).

Figure 16 depicts the number of infected and susceptible pedestrians over time. As one can see, the number of infected pedestrians increase and the susceptible decrease throughout the simulation. In the end of the simulation about 120 people got infected. This means 60% of the population got infected.

If the discretization of the simulation time is too high meaning `simTimeStepLength` is closer to 1, we increase the `maxDistance` of the `SIRGroupModel` in order to regenerate the effect of the pedestrians passing by. For example if the `simTimeStepLength` is too high and two pedestrians are not in the range of `maxDistance` to each other. If the `simTimeStepLength` causes them to pass by without entering their `maxDistance` ranges although they would have been in this range if discretization is low, then we try to compensate this by simply increasing the `maxDistance` attribute of `SIRGroupModel`. With the following example, one can see that how the `maxDistance` is changed depending on the discretization:
If the `simTimeStepLength` is set to 0.77 and the `maxDistance` is 1.0, assuming the average speed of 1.42 m/s of a pedestrian, the distance in one time step the pedestrian takes is 1.0934 which is greater then the given `maxDistance`. Therefore we increase the `maxDistance` to $(1.42 * 0.77)$ include them in infection range.
For possible extensions of the current SIR model we have the three following suggestions:

1. The infection rate of the disease describes a feature of the that disease but this does not tell anything about the one's immunity. This model makes a pedestrian sick depending on the infection rate without considering the immunity of that person. For example from Corona-Virus we know that the older people may be get easier infected and affected relative to younger people. Therefore adding an immune attribute for each pedestrian because even if a person is young, they can have pre-conditions which potentially weaken their immune systems, would be a good idea. In a simplified version, the immune attribute can be directly set according to the person's age without considering special cases.

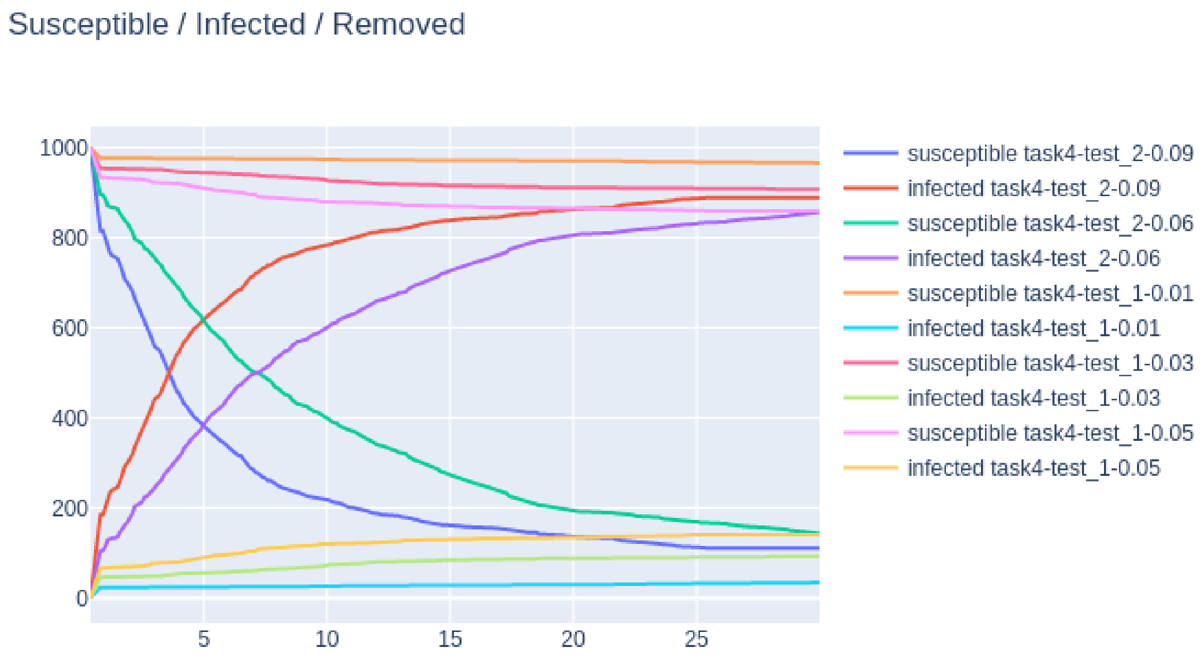


Figure 15: On the `x` axis we see simulation time in seconds and on the `y` axis the number of pedestrians. Lines represent the number of corresponding pedestrians throughout the simulation. For example the `susceptible task4-test_2-0.09` means the number of susceptible pedestrians with `infectionMaxDistance` equals 2 and `infectionRate` equals 0.09.

2. Another extension can be adding environmental factors to the model. In real-life scenarios how many walls surround the room or the air quality (open or closed with no air-conditioner) may make a difference in the spreading rate of a disease, since the disease that we try to simulate is assumed to be spread through air. Therefore we argue that this extension may make the SIR Model more realistic. This can be e.g. added by checking the distance of the pedestrian to the walls and assume that a smaller room would make the disease spread more easily.

3. Protections against the spreading may also be added to the model (e.g. which pedestrians are covering their mouth and nose with a mask, points in the topology for hand sanitisers that the pedestrians may use and the simulation may dynamically change the infection rate for these pedestrians by checking a variable `lastTimeHandsCleaned` etc.).

Report on task Task 5/5: Analysis and visualization of results

In order to add a "recovered" state that represent the recovered persons to our SIR Model, we add an *Enum* item `ID_RECOVERED` to the file `SIRType.java`. We have also added a variable `recoveryRate` to the file `AttributesSIRG.java`. One should add this variable to the Model JSON file as shown in figure 17. This is the probability for an "infected" pedestrian to become "recovered" at every time step.

Recovery of a person happens independent of where they are, or how many persons around them are "infected" or "recovered". And recovered persons cannot get re-infected or cannot infect susceptible persons. We have implemented these features by adding additional implementation in the function `update` of file `SIRGroupModel.java` as shown in figure 18. In the first if-statement we check if the current considered pedestrian is "recovered", and if so, we skip this pedestrian since it cannot become "infected" again. In the

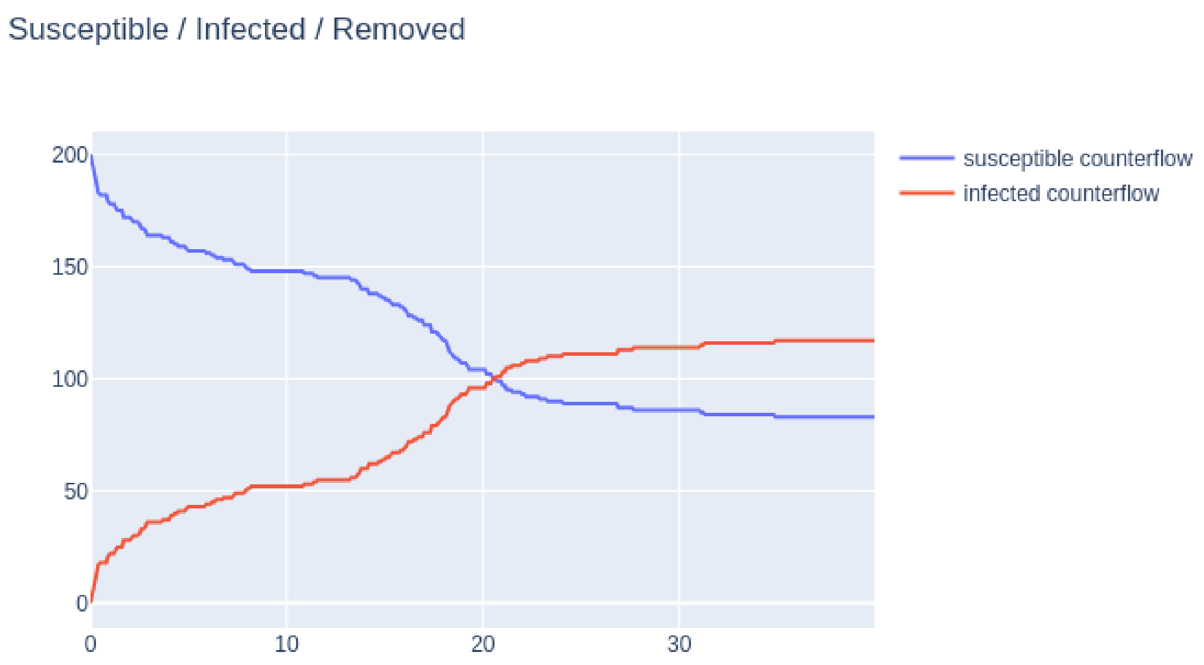


Figure 16: On the x axis we see simulation time in seconds and on the y axis the number of pedestrians. Red-line is the number of infected pedestrians and blue-line is the number of susceptible pedestrians throughout the simulation.

```
"org.vadere.state.attributes.models.AttributesSIRG" : {
  "infectionsAtStart" : 10,
  "infectionRate" : 0.75,
  "recoveryRate" : 0.02,
  "infectionMaxDistance" : 1.0
},
```

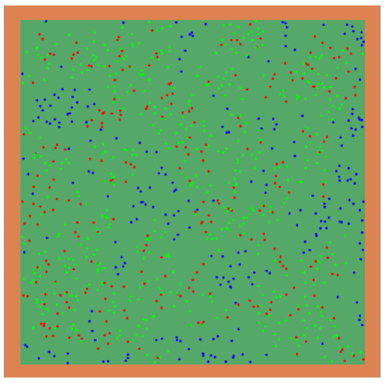
Figure 17: `org.vadere.state.attributes.models.AttributesSIRG` field in the JSON file of the Model.

second if-statement we make the pedestrian "recovered" with the probability `recoveryRate`. In the current setup recovered pedestrians cannot infect susceptible ones since it only considers the "infected" pedestrians already. We have also modified the `utils.py` function in the given visualization web-application to include the "recovered" pedestrians.

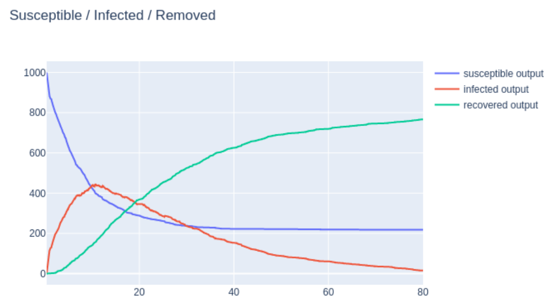
```
if (getGroup(p).getID() == SIRType.ID_RECOVERED.ordinal()) {
  continue;
}
if (getGroup(p).getID() == SIRType.ID_INFECTED.ordinal() &&
    this.random.nextDouble() < attributesSIRG.getRecoveryRate()) {
  elementRemoved(p);
  assignToGroup(p, SIRType.ID_RECOVERED.ordinal());
  continue;
}
```

Figure 18: Added if-statements to the pedestrian loop in the `update function` in `SIRGroupModel.java`

We have experimented with the infection rate and the recovery rate of pedestrians as well to see their effect



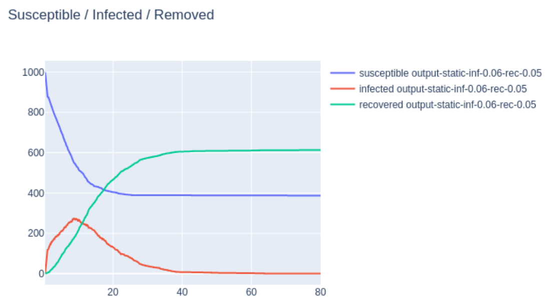
(a) Static scenario with infection rate 0.06 , recovery rate 0.02 and the snapshot is taken at timestep 26.4 s



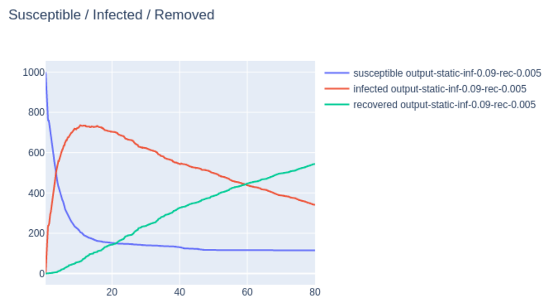
(b) Static scenario run with infection rate 0.06, recovery rate 0.02, max distance 2m

Figure 19

on the number of susceptible, infected and recovered pedestrians. In Figure 20 there are two different setups that we have run Vadere simulation software is depicted: Both of these simulations were run in the static pedestrian scenario, where there are 1000 pedestrians, 10 start with an infection and 990 are susceptible. The caption of both graphs state the different infection rate and recovery rates that are used in that specific scenario, the max distance value is not changed during these tests, to only visualize the effect of the rates. The sub-figure (see Figure 20) on the left depicts a case where recovery rate and infection rate are close to each other, which means the chance of a person getting infected and the same person getting recovered is nearly the same, which results in a situation, where people cannot get infected too much and even the people who has an infection gets recovered quickly, so the peak number of infected pedestrians is much lower than on the scenario in the right sub-figure (see Figure 20). In the right sub-figure the output of a scenario is depicted, where the recovery rate is much lower than the infection rate and the infection rate is also increased in comparison to the first scenario. This results in a more steep increase of infected pedestrians at the start of the scenario and because of the little recovery rate, it takes more time to reach the same number of recovered pedestrians in comparison to the left sub-figure.



(a) Static scenario run with infection rate 0.06, recovery rate 0.05, max distance 2m



(b) Static scenario run with infection rate 0.09, recovery rate 0.005, max distance 2m

Figure 20

The following scenario is used to create a so called supermarket scenario.There are two sources to which the targets are given in different orders to realise natural behavior of customers in supermarket. However the last target for both of the sources is the one on the top right in order to represent the exit and entrance of the supermarket.

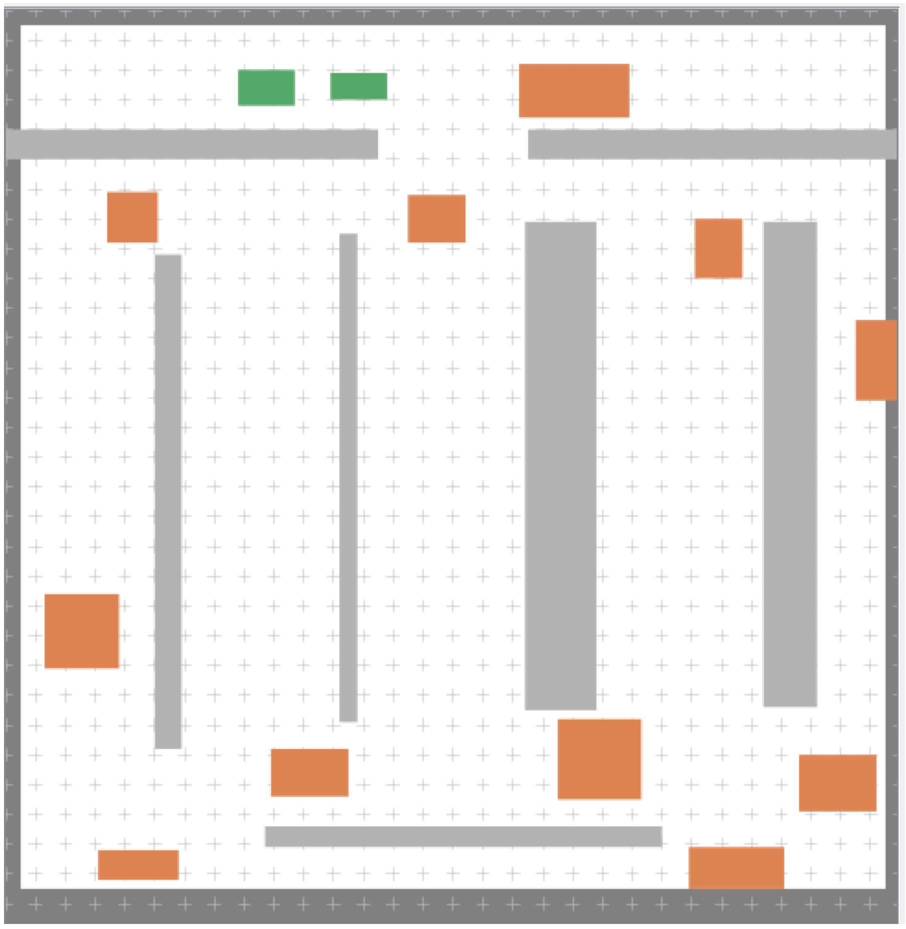


Figure 21: Supermarket scenario

In figure 22 we see the results of first case of supermarket scenario where the infection rate is 0.06 , recovery rate is 0.005 and the max-distance is 0.7.For comparison reasons these parameters are kept same in the scenario of 23.However in this scenario the attribute `pedPotentialPersonalSpaceWidth` is doubled which means the pedestrians want to keep more distance to each other compared the first case where `pedPotentialPersonalSpaceWidth` is 1.2 . As a result of this the spread of the disease is drastically reduced. In the first case more than half of the customers get infected while in the second scenario only a few new customers get infected (initial infection start is 3 customers) and the spread of disease is stopped very quickly.

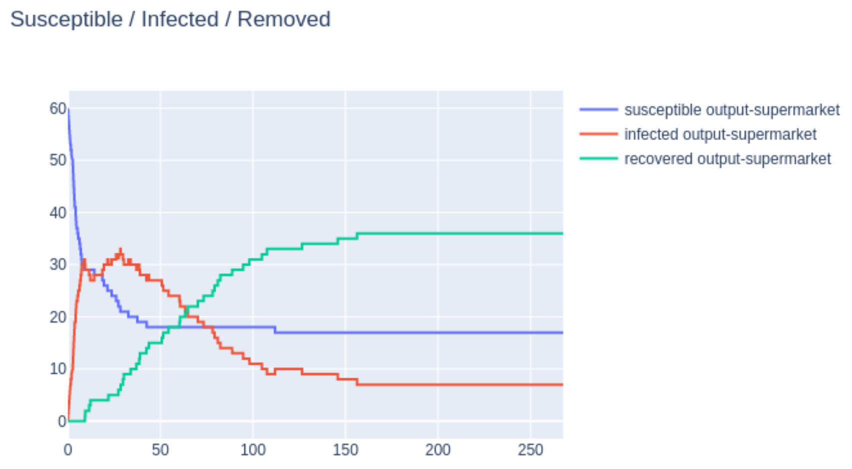


Figure 22: Supermarket scenario run with infection rate 0.06, recovery rate 0.005, max distance 0.7

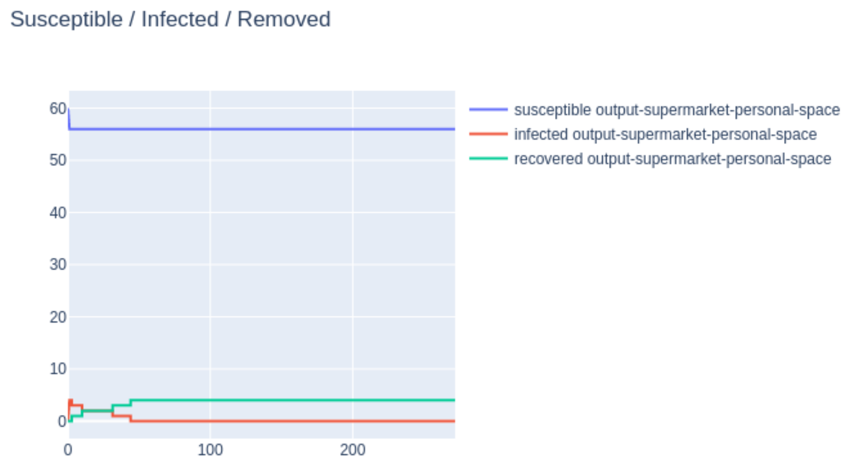


Figure 23: Supermarket scenario run with infection rate 0.06, recovery rate 0.005, max distance 0.7m and pedPotentialPersonalSpaceWidth 2.4m (doubled compared to initial value)

References

[1] Felix Dietrich and Gerta Köster. “Gradient navigation model for pedestrian dynamics”. In: *Physical Review E* 89.6 (June 2014).

[2] Felix Dietrich et al. “Bridging the gap: From cellular automata to differential equation models for pedestrian dynamic”. In: *Journal of Computational Science* 5.5 (2014), pp. 841–846.

[3] https://gitlab.lrz.de/vadere/vadere. *Vadere*. May 2022.

[4] Murtagh, Elaine M., Mair, Jacqueline L., Aguiar, Elroy, Tudor-Locke, Catrine, Murphy, Marie H. “Outdoor Walking Speeds of Apparently Healthy Adults: A Systematic Review and Meta-analysis”. In: 51 (Jan. 2021), pp. 125–141.

[5] RiMEA. *Guideline for Microscopic Evacuation Analysis 3.0.0 edition.* www.rimea.de. 2016.

[6] Michael Seitz and Gerta Köster. "Natural discretization of pedestrian movement in continuous space". In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 86 (Oct. 2012), p. 046108. DOI: `10.1103/PhysRevE.86.046108`.