Report for exercise 2 from group C

Tasks addressed:	5
Authors:	Aleksi Kääriäinen (03795252)
	Danqing Chen (03766464))
	Julia Xu (03716599)
	Joseph Alterbaum (03724310)
Last compiled:	2024–09–27

The work on tasks was divided in the following way:

Aleksi Kääriäinen (03795252)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Danqing Chen (03766464))	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Julia Xu (03716599)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Joseph Alterbaum (03724310)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%

Report on task 1, Setting up the Vadere environment

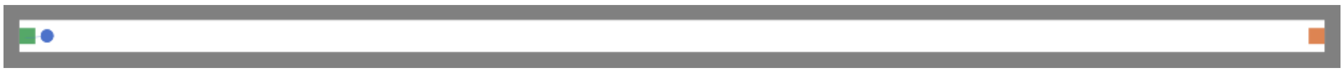
1 Vadere setup

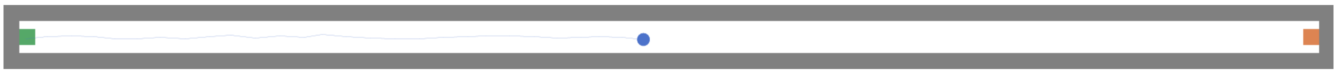
The first task mainly concerns running first simulations in Vadere with the Optimal Steps Model (OSM) and comparing them to the simulations with our cellular automaton in the first exercise. For this purpose we used the already compiled Vadere-version downloaded from their website. The executable JAR-file of the Vadere GUI could be started immediately after making sure the right Java-version (11 or higher) was installed, so no further setup was necessary.

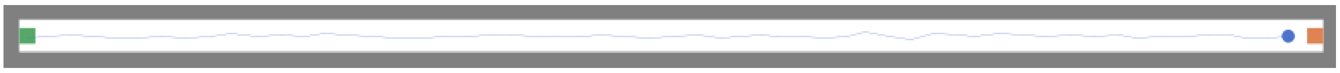
1.1 RiMEA-scenario 1 – Straight Line

First of all, the behaviour of the standard OSM in Vadere for RiMEA-test 1 (straight line) was observed. For reference we want to provide the textual description of the test from the RiMEA-guideline [1]: "*It is to be proven that a person in a 2 m wide and 40 m long corridor with a defined walking speed will cover the distance in the corresponding time period. If 40 cm (body dimension), 1 second (premovement time) and 5% (walking speed) are set as imprecise values, then the following requirement results with a typical pedestrian speed of 1.33 m/s: the speed should be set at a value between 4.5 and 5.1 km/h. The travel time should lie in the range of 26 to 34 seconds when 1.33 m/s is set as the speed.*"
The scenario was implemented in Vadere by an area of 2 m height and 42 m width, where the latter compensates for the extent of the source and target fields. The source was placed as a 1m x 1m rectangle on the left side while the target area is a 1m x 1m field on the right which absorbs pedestrian upon arrival, cp. Fig. 1a. Apart from the spatial arrangement of the test, the speed and premovement time have to be set. We modeled the premovement time of the single pedestrian by setting the source (or "spawner" in Vadere) attributes "constraintsTimeStart" & "constraintsTimeEnd" both to 1.0, which corresponds to the source area spawning the pedestrian after 1s of the simulation. Furthermore, by editing the "attributesPedestrian"-entry in the topology dictionary we can specify the speed behaviour of the spawned pedestrian. In order to achieve the speed of 1.33 m/s with an imprecision of 5%, we set the attributes [in m/s]: "speedDistributionMean" to 1.33, "minimumSpeed" to 1.2635 & "maximumSpeed" to 1.3965. Converted to km/h this equals to a minimum of 4.5386 km/h and a maximum of 5.0274 km/h and although contradicting the statement in the RiMEA-guideline, we chose them because they are the exact values for a 5% imprecision. The body dimension of 40 cm, corresponding to a radius of 20 cm, is the Vadere standard setting and can be changed in the "radius"-value within "attributesPedestrian". Importing the standard template of the optimal steps model in Vadere is done via the "Model"-tab & "Load template" where the OSM can be found under "org.vadere.simulator.models.osm.OptimalStepsModel". The corresponding scenario-file is stored in the repository under "/scenarios/task_1/RiMEA_1_straight_line_OSM.scenario".

Figure 1: RiMEA-test 1 within Vadere


(a) Initial setup of RiMEA-test 1 in Vadere


(b) Simulation of RiMEA-test 1 after 17s with an OSM


(c) Final state of the simulation of RiMEA-test 1 (right before absorption of pedestrian)

When running the simulation with the described setup, we can observe that the test is completed successfully. We found that after a time of 34 seconds, which is 86 simulation steps, the pedestrian reaches the target, cp. Fig. 1c, and stayed within the 2m corridor in the meantime, cp. Fig. 1b. In the 87th step, after 34.126 s, the pedestrian is finally absorbed by the target. The fact that it took the pedestrian 0.126 s more than implied by the RiMEA-guideline is insignificant since the pedestrian is spawned slightly farther than 40 m from the target. In comparison with our cellular automaton, the test results are similar: Both models achieve the pedestrian covering the distance of 40 m within the provided time frame. However, a few differences occur when looking at the OSM-setup in Vadere. First of all, the trajectory of the OSM-simulation is more fine-grained and not as straight as in the cellular automaton. This also reflects the design choices we made in the first exercise: We assumed the pedestrian to fill one cell and therefore had to make the cell size 0.4 m. In Vadere the extent of a pedestrian is decoupled from the discretisation of the grid which allows for movement smaller than a full body size and a round model of the 2D-extent instead of the quadratic model in our implementation. Both models work with discretisation which can be seen by the incremental movement in Fig. 1c.
Moreover, many differences regarding the visualisation, user interface & general functionalities in Vadere are evident which were observed in all three test scenarios of the first task. Looking at the visualisation of Vadere, all elements seen in our cellular automaton implementation are found in a quite similar way again, e.g., visualisation of target, source & obstacles, the grid, etc. However, Vadere offers far more possibilities for the user to visualise the simulation and its results. One of the most useful features is the depiction of the trajectory as a thin line behind the pedestrian. Further features include the direction of movement, extensive post-processing capabilities, automatic result exports or the ability to take screenshots and videos directly. So while the two implementations obviously share similar roots, the visualisation options in Vadere are far more professional. The same can be said about the user interface, where core functions such as the starting, stepping and stopping of the simulation are the same as in the cellular automaton. Again, the Vadere user interface is far more advanced since the full scenario can be specified graphically in contrast to us specifying .json-files manually in the first exercise. Furthermore, shortcuts such as Strg+C exist and the setup can be viewed as plain text, in a table of attributes and as a tree view of the system. Additionally, the setup offers many more functionalities such as pedestrian spawners, extensive setting options & attributes for all objects, a 3rd dimension in the sense of stairs and meta-attributes of the simulation, e.g. a finish time. In conclusion, the visualisation & user interface of Vadere and the cellular automaton from exercise 1 share basic functionalities but it is clear that Vadere offers a much wider variety of features and is intended as a more professional tool.

1.2 RiMEA-scenario 6 – corner

The second scenario to look at is the RiMEA-test 6 where pedestrians need to round a corner succesfully. The textual specification is the following: "*Twenty persons moving towards a corner which turns to the left will successfully go around it without passing through walls.*" [1]. A graphic depicted in Fig. 2 shows the spatial extents of the test scenario.

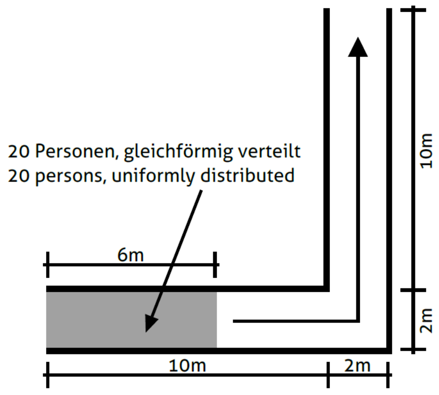
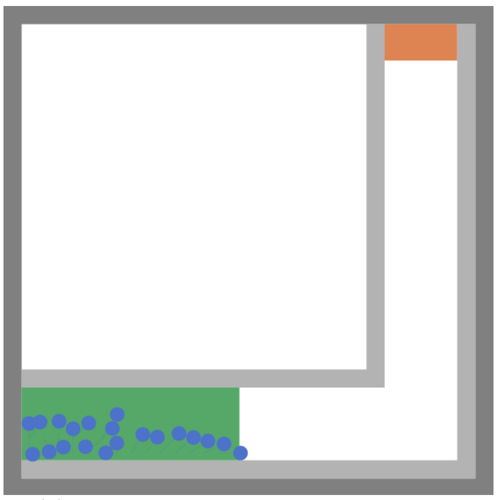
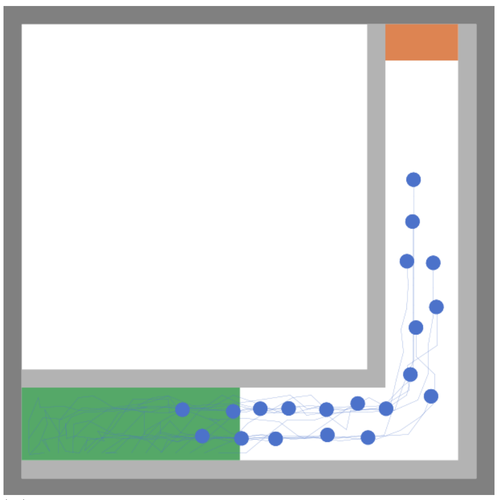

20 Personen, gleichförmig verteilt
20 persons, uniformly distributed
6m
10m
2m
10m
2m

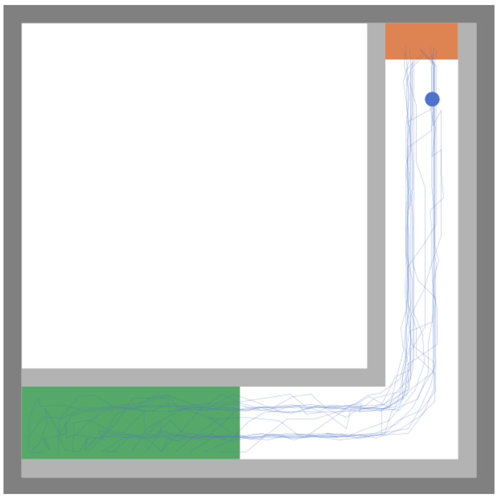
Figure 2: Specification of RiMEA-test 6 from the guideline [1]

The scenario was realised in Vadere by a 12.5 m x 12.5 m grid to compensate for the lower wall which was assumed to have a thickness of 0.5 m, cp. Fig. 3a. While the target area was assumed to be absorbing and at the end of the corridor, the source area was set as specified in RiMEA-test 6 to a 6 m x 2 m rectangle on the lower left side. Although technically the lower wall could be neglected, we decided to include it in our scenario since there might be effects in some models preferring to keep a distance to obstacles. Regarding the pedestrians, the only constraint was to sample 20 people which can be achieved in the ”eventElementCount”-attribute of the source/spawner. The distribution of pedestrians was set to a constant value of 1.34 m/s which is the preset in Vadere. Setting OSM as the model is done analogously to the first test. The corresponding scenario-file is stored in the repository under ”/scenarios/task_1/RiMEA_6_corner_OSM.scenario”.

Figure 3: RiMEA-test 6 within Vadere


(a) Initial setup of RiMEA-test 6 in Vadere


(b) Simulation of RiMEA-test 6 after 10s with an OSM


(c) Final state of the simulation of RiMEA-test 6 (right before absorption of the last pedestrian)

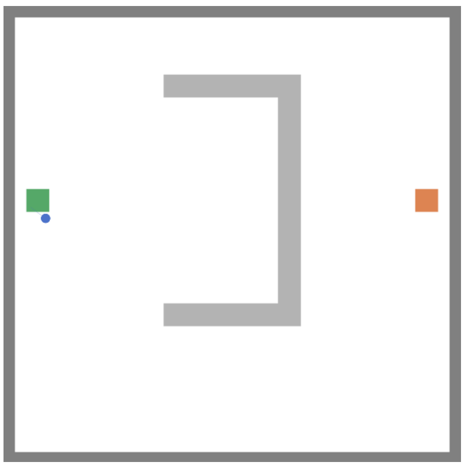
After running the simulation we can see that the test is passed successfully. This is especially easy in the GUI by looking at the trajectories and checking that no walls were skipped, cp. Fig. 3c. A notable observation with the Optimal Steps Model is that pedestrians sometimes walk away from the target area which can be visualised by showing the walking directions in the GUI.
When comparing the results to the cellular automaton, the overall test result is quite similar since both models passed and no pedestrian walked through a wall. However, the model trajectories are quite different: While for naive distance & Dijkstra’s algorithm in the first exercise the pedestrians gathered along the upper wall in the first part of the corridor, with the Optimal Steps Model in Vadere they walk rather uniformly towards the corner before rounding it. In the second part of the corridor after the corner the trajectories look very similar for both simulations. An interesting observation with the OSM is the formation of two main paths where pedestrians walk. This can be seen in the visualisation of model trajectories in the final state, Fig. 3c, where two parallel

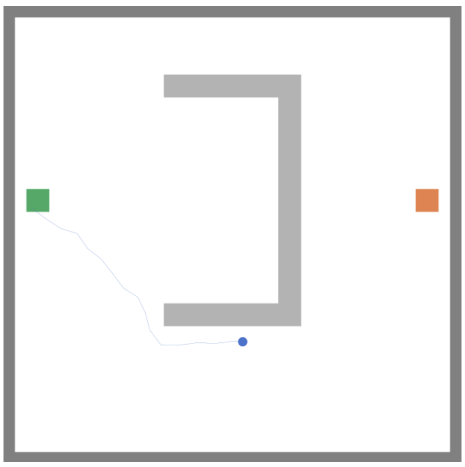
lines emerge from the accumulation of trajectories. Even within the corner these two paths remain segregated although weaker than before and after the turn. As before, the movement is generally more fine-grained hinting at the smaller discretisation within Vadere. The visualisation of trajectories in Vadere is particularly helpful for this test since with our cellular automaton we had to check manually if the pedestrians step into a wall. Furthermore, the movement is calculated more efficiently in Vadere which looks more realisitc and helps with a smooth visualisation. Apart from these two differences the general observations for visualisation and user interface from the first test apply.

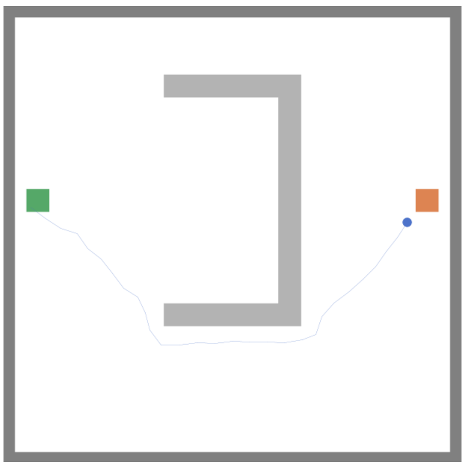
1.3 Chicken Test

Lastly, the behaviour of an OSM in Vadere regarding the "Chicken Test" should be tested. The setup can be seen in Fig. 4a and models one pedestrian trying to reach a target that is positioned on a straight line from the source. However, a U-shaped obstacle is placed on the straight line blocking the pedestrian completely if no obstacle avoidance is implemented in the simulation, e.g., a Euclidean distance as only cost-function is used. In our case, we used a 20 m x 20 m grid with a pedestrian spawner on the left and the target on the right. The U-shaped obstacle is 6 m wide and 11 m high. Goal of the test scenario is that the pedestrian reaches the goal area and does not get stuck anywhere. Since the speed of the pedestrian is not important for this test it was again set to 1.34 m/s and the OSM was included analogously to the previous subtasks. The scenario-file can be found in the repository under "/scenarios/task_1/Chicken_Test_OSM.scenario".

Figure 4: Chicken Test within Vadere


(a) Initial setup of the chicken test in Vadere


(b) Simulation of chicken test after 10s with an OSM


(c) Final state of the simulation of the chicken test (right before absorption of the pedestrian)

After running the simulation we observed that the pedestrian chose a trajectory that avoided getting stuck right from the start. This can be seen by the diagonal movement starting at the source in Fig. 4b. The simulation passed the test and produced a trajectory that minimised the walked distance while keeping a certain clearance to the obstacle. The final state can be seen in Fig. 4c.
When comparing an OSM in Vadere and a cellular automaton with Euclidean distance cost, the biggest difference is that the cellular automaton gets stuck due to missing obstacle avoidance which is implemented in the OSM. For Vadere with an OSM & our cellular automaton with Dijkstra's algorithm the chicken test is passed but a few differences occurr. The cellular automaton with Dijkstra chose the path that minimised covered distance and in consequence its pedestrian went right along the wall of the U-shaped obstacle. This indicates that the standard OSM also keeps some kind of cost for being too close to an obstacle and therefore avoids going right along walls. Again, the trajectory in Vadere is more fine-grained and smaller movements to each side of the optimal path occur. Apart from this, the behaviour is very similar. Furthermore, the described differences between the visualisations and user interfaces (cp. Chap. 1.1) have no particular advantage when observing the chicken test.

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

2 Social Force (SFM) & Gradient Navigation Model

2.1 Social Force Model (SFM)

To repeat the three scenarios with the Social Force Model, the scenarios were copied and the model changed to the template "org.vadere.simulator.models.sfm.SocialForceModel" in the Model-tab.

2.1.1 RiMEA-test 1 – Straight Line

The scenario-file for RiMEA-test 1 with an SFM is stored under "/scenarios/task_2/RiMEA_1_straight_line_SFM .scenario" in the repository. When running the simulation we can observe a very straight movement towards the target area, visible in the trajectory within Fig. 5. Furthermore, the RiMEA-test is passed since it took 31.6 s or equivalently 79 steps to finish the 40 m corridor.
Compared to the Optimal Steps Model the trajectory is much smoother but overall the path is similar and both models pass the RiMEA-test goal. Additionally, the time behaviour of the SFM (31.6 s) is much closer to the expected time behaviour of the scenario, which is 40 m / 1.33 m/s = 30.08 s, than the OSM time behaviour (34.126).

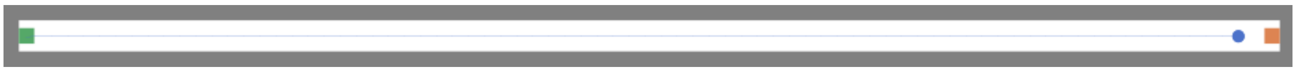


Figure 5: Final state of the SFM-simulation of RiMEA-test 1 (right before absorption of the last pedestrian)

Possible reasons for the observed behaviour in different models is most probably because of modeling assumptions & their mathematical expression in the corresponding model. The OSM updates the pedestrians position by maximising its utility which depends negatively on the distance to the target, negatively on the proximity to other pedestrians & obstacles [2]. This means that it is assumed that pedestrians want to go the shortest way avoiding pedestrians & obstacles. The same assumptions are made in the SFM, where the update decision is made on top of forces instead of utilities for every position. The forces include a repelling force for obstacles & other pedestrians as well as a force drawing the pedestrian towards the target [4]. The similar path in OSM & SFM is probably due to the same assumptions underlying a pedestrians walking decisions which are especially easy in the case without obstacles and other participants. However, the differences in the fine-grained trajectory are most probably based on the different underlying mathematical formalisation of the assumptions. A force drawing the pedestrian model in a certain direction can be much more fine-grained than a utility grid which has to be computed for every possible position x. Furthermore, since distance to target is the only relevant parameter in this scenario the utility differences between two vertically adjacent positions is small which could ultimately cause the zig-zag-movement in the OSM simulation.

2.1.2 RiMEA-test 6 – Corner

To recreate the second test, we ran the scenario-file "/scenarios/task_2/RiMEA_6_corner_SFM.scenario" in the repository. Most of the pedestrians in the simulation finished within 20 s (50 steps) while one pedestrian remained until 29.2 s (73 steps). In general, the pedestrians were drawn towards the upper wall in the first corridor segment and crowded around the corner , cp. Fig. 6, where a lot of overlapping of pedestrians occurred. After turning the corner, the inidivudal trajectories spread around the center of the corridor. Since no obstacles were skipped, the RiMEA-test 6 was passed successfully. An interesting observation is that once the last single person had a higher distance to the rest of the crowd, its speed was reduced drastically.
First of all, both models passed the RiMEA-test but with very different trajectories. In the OSM-scenario the pedestrians do not crowd around the corner and walk close to the upper wall in the first segment but go rather uniformly distributed, cp. Fig. 3c. Furthermore, the second corridor segment is passed differently with a split into two main paths while in SFM pedestrians walk in the middle of the corridor. The finish time for both simulations is similar (OSM 28 s/70 steps & SFM 29.2 s/73 steps) however, in OSM no pedestrian is left behind or equivalently turns the corner especially slowly. Moreover, the overlapping of pedestrian models in SFM does not occur and in general a bigger distance between single participants can be observed.

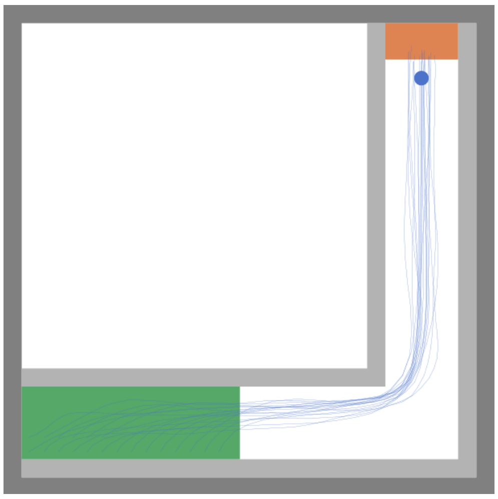


Figure 6: Final state of the SFM-simulation of RiMEA-test 6 (right before absorption of the last pedestrian)

Again, the reasons for the shown behaviour most probably lie in the formalisation of modeling assumptions. As discussed in the first scenario, SFM uses forces to model a fast way to the target. This could explain the influence of the target in the first segment of the corridor such that the pedestrians in the SFM want to be closest to the target and therefore go along the upper wall and crowd in front of the corner. In OSM, where a utility grid is used, this might not be the case because the utility of each vertical position for a given horizontal place is equal since the wall blocks direct movement to the target. The increased avoidance of other pedestrians in OSM might be due to a higher weighting of this aspect in the utility function rather than mathematical differences to modeling forces in the SFM. This could also explain why in the OSM two main paths in the second corridor segment emerge while for the standard SFM the crowd moves spread along the center of the corridor. The issue of a pedestrian getting stuck for some time at the corner in SFM might be due to the appealing forces of the target and the repelling forces of the wall exactly cancelling out. Therefore, the difference of forces determining the direction and acceleration of movement becomes small and the pedestrian walks only very slowly.

2.1.3 Chicken Test

The scenario-file for the chicken test can be found in the repository as "/scenarios/task_2/Chicken_Test_SFM .scenario". The simulation finished within 19.2 s or 48 steps and the pedestrian went around the obstacle to the target, cp. Fig. 7, which means that the chicken test is passed. In general, the path is similar to the OSM but the trajectory is much more smooth. One notable observation is that the path comes very close to the lower obstacle boundary at first and then does an evasion movement of the obstacle corner, as seen in the lower left corner of the obstacle in Fig. 7.

When comparing the simulation to the OSM, both model reach the goal and therefore pass the test. Overall, the OSM-simulation finishes slightly faster (17.6 s) and the distance to the obstacle is a bit more uniform which results in a more symmetric path of the pedestrian, cp. Fig. 4c. Furthermore, the OSM produces rather sharp corners in the trajectory which do not occur in the SFM.

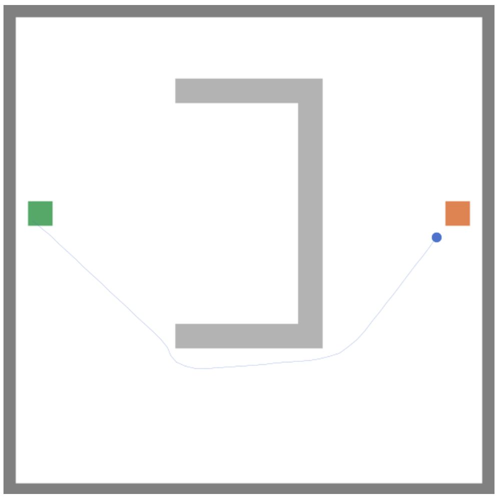


Figure 7: Final state of the SFM-simulation of the "chicken test" (right before absorption of the last pedestrian)

Similarly to the first two scenarios in this chapter, the smooth movement is most probably due to the modeling difference of forces in SFM vs. utility grids in OSM. Because forces in general produce a more smooth sequence of directions compared to a discretised grid of utilities the trajectory of SFM is smoother than the one from OSM. The sharp corners in the OSM-trajectory are probably due to the discretised utility grid and sudden changes of utility. Again a similar overall path is chosen due to the same underlying assumptions of avoiding obstacles and reaching the target fast in both models. The assumption of avoiding other participants has no influence since only one pedestrian exists.

2.2 Gradient Navigation Model

Again, the scenarios were copied but the model was changed to the template "org.vadere.simulator.models.gnm .GradientNavigationModel" in the Model-tab to repeat the scenarios with the Gradient Navigation Model.

2.2.1 RiMEA-test 1 – Straight Line

Here, the scenario file of the first RiMEA-test can be found as "/scenarios/task_2/RiMEA_1_straight_line_GNM .scenario" in the repository. After simulating the scenario, we observed a very straight trajectory with a slight but constant vertical disposition from a third of the horizontal distance until nearly the end, cp. Fig. 8. The pedestrian took exactly 30 s or 75 simulation steps to reach the target and therefore the RiMEA-test is passed successfully.
First of all, it has to be emphasized that all three inspected models pass the test conditions. Overall, the trajectories produced by SFM & GNM are very similar and only differ from the OSM-trajectory regarding the smoothness of movement, which tends to show more erratic deviations from the straight line in OSM. While the GNM hits the target timing exactly (30 s), the SFM (31.6) is still better than the OSM (34.126).

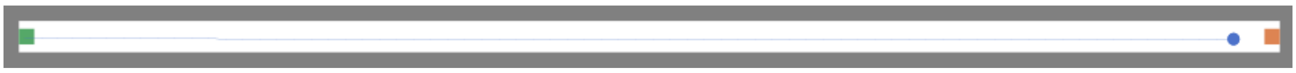


Figure 8: Final state of the GNM-simulation of RiMEA-test 1 (right before absorption of the pedestrian)

Possible reasons for the similarity of SFM & GNM in this scenario can be found when looking at the assumptions of both models: GNM assumes that pedestrians want to reach their goal in the fastest way &

that they alter their velocity as a reaction to obstacles & other pedestrians [3]. From this assumption a set of differential equations is derived which in turn is solved to determine the trajectory of the pedestrians. SFMs however model the pedestrians movement with the impact of forces including a repelling force from other pedestrians, another repelling effect when close to obstacle borders and a force to emphasize the shortest possible way towards a target [4]. In our situation where no other pedestrians and obstacles are considered the assumptions are similar and therefore the corresponding trajectories are almost the same.

2.2.2 RiMEA-test 6 – Corner

The scenario file of this test is saved under "/scenarios/task_2/RiMEA_6_corner_GNM.scenario" in the repository. We observed the simulation successfully passing the test and not skipping any obstacles. Most of the pedestrians in the scenario finished within 32 s or 80 steps but one remaining pedestrian took 40.4 s/101 steps to reach the target. In the first segment of the corridor the pedestrians were drawn heavily towards the upper wall and built a dense crowd in front of the corner while not overlapping each others space. Finally, one after another goes around the corner and walks towards the target on the very left of the corridor all on the same path, which can be seen by the strong trajectory on the left in the upper corridor in Fig. 9. The last pedestrian waits long before rounding the corner and therefore gets separated from the rest of the group.
On the one hand, all three models passed the RiMEA-test successfully. On the other hand, the trajectories of the three model differ quite a bit. While GNM & SFM show the same trajectories in front of the corner (close to upper wall), OSM is different for this segment (uniform movement forwards). Furthermore, in SFM & GNM the pedestrians crowd around the corner where additionally in SFM overlapping of pedestrian models occurs. All three models display different behaviour after rounding the corner (two paths in OSM, one central path SFM, the same path along the left wall in GNM). While the GNM simulation takes the longest time to reach the goal (40.4s), the timing in SFM & OSM is similar (SFM 29.2 s vs. OSM 28 s). Finally, in SFM & GNM simulation one pedestrian gets separated by walking around the corner especially slow.

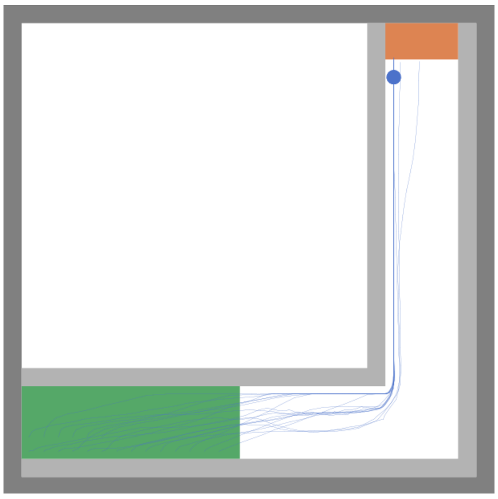


Figure 9: Final state of the GNM-simulation of RiMEA-test 6 (right before absorption of the last pedestrian)

The similarity in modelling assumptions between SFM & GNM was stated before and is a possible reason for the similar behaviour before turning the corner. Since the fastest path is a subgoal in SFM & GNM for each pedestrian, going along the upper wall minimises the distance. The observations in the second segment of the corridor cannot be fully explained by this. Here, the weighting, e.g. for avoiding pedestrians, staying off obstacles and going the fastest way, & exact mathematical formulation of assumptions (forces in SFM [4] & differential equations in GNM [3]) could be reasons for the differing behaviour. As a consequence, the weighting for the fastest approach is assumed to be higher than for the distance to obstacles since the trajectory along the left wall is the shortest path. In contrast, it appears that distance to obstacles in the standard SFM is weighted rather high (central path in second segment) while the weighting for avoiding other pedestrians is so low that even overlapping is a viable option within the simulation.

2.2.3 Chicken Test

For the final test, the scenario-file is saved under "/scenarios/task_2/Chicken_Test_GNM.scenario" in the repository. The simulation finished after 18 s or 45 steps and passed the test successfully since it went around the obstacle and reached the target. Noticeable is the very smooth and straight trajectory, cp. Fig. 10. Furthermore, the path was chosen very close to the obstacle but with a distinct and uniform distance along the obstacle border.
Overall, all three models got to the target and passed the chicken test successfully. Moreover, the finishing times and the general path below the obstacle was very similar. However, the behaviour close to the obstacle differed: SFM & OSM showed similar trajectories with the first being much more smooth while the path in the GNM simulation was more symmetric and close to the obstacle than with SFM & GNM.

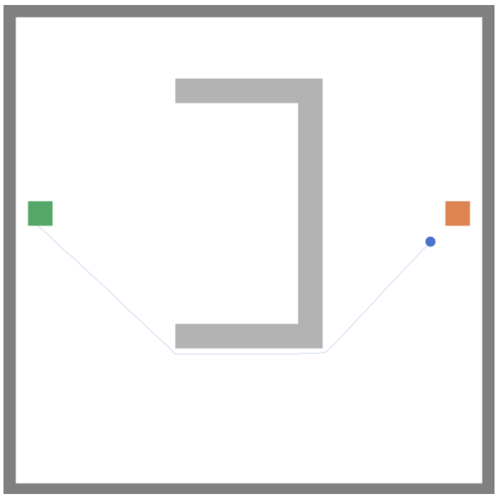


Figure 10: Final state of the GNM-simulation of the "chicken test" (right before absorption of the pedestrian)

Similar to the scenarios before, possible reasons could lie in the modeling assumptions of SFM & GNM. In this case, we face only obstacles and no influence of other pedestrians so the difference in the trajectory can probably be traced back to different assumption regarding obstacle avoidance. While one possible explanation for the more symmetric trajectory in GNM is the weighting of obstacles, we think it is most probably the mathematical formulation with differential equations that give rise to the very symmetric and smooth path.

Report on task 3, Using the console interface from Vadere

3 Vadere Console

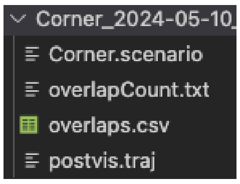
There are two ways to interact with the Vadere software: 1) through its given GUI (as used in the other tasks) or 2) through its console (as used in this task).
To run a scenario via the console use the following command (see exercise sheet):

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

3.1 Comparing Outputs from Vadere GUI and Console

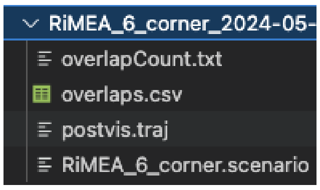
This subsection briefly discusses the similarities and differences of the output files generated by running the corner scenario from the previous task via the Vadere GUI and its console. Both output folders save the scenario file as well as a text file, a CSV file and a traj file containing all the simulated pedestrians' data.
Both methods generate the same amount of files. By running the corner scenario both procedures produce four files as shown in figure 11.

Figure 11: Generated output files from running the corner scenario with (a) GUI and (b) console.



Corner_2024-05-10_
Corner.scenario
overlapCount.txt
overlaps.csv
postvis.traj

(a) GUI output



RiMEA_6_corner_2024-05-
overlapCount.txt
overlaps.csv
postvis.traj
RiMEA_6_corner.scenario

(b) Console output

There were no significant differences found in the output folders. However, without the GUI there is no graphical simulation visible and only the resulting data can be interpreted.

3.2 Adding new pedestrian via code

This task focuses on writing a script to generate a scenario file with an additional pedestrian from the given corner scenario. Here, the programming language Python is used.

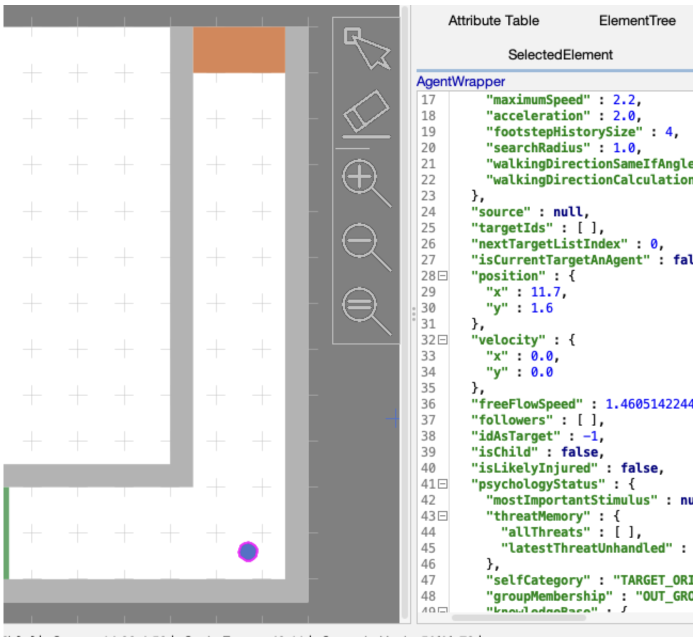
The idea is to first read the given corner file, then extract all its components into an implemented scenario structure, and implement a method that adds a new pedestrian to the scenario. In the end, the created scenario structure is saved into a json-structured file (with .scenario ending!) which can then run via the Vadere console as shown above.

3.2.1 Code Implementation

There are two Python files created to implement this task: (1) `generate_scenario.py` and (2) `elements.py`. The latter consists of multiple classes that represent elements in an usual scenario file such as `Topography` or `Pedestrian` for example. The first Python file is used to extract a scenario/json file into instances from `elements.py` as well as an implemented scenario instance. At the end of `generate_scenario.py` all parts from a scenario instance are then being written in a new json file.

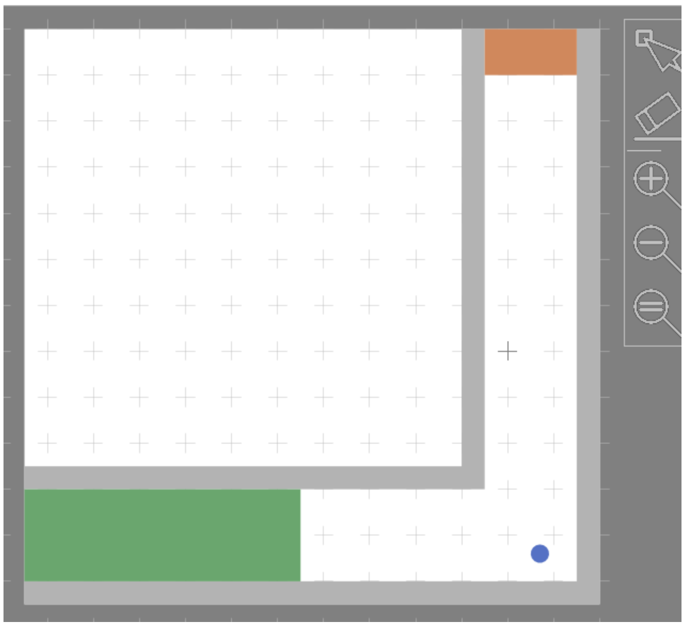
To get the right coordinates, the given (original) corner scenario is run in the GUI and a new pedestrian is manually added in the cornern. Then the coordinates from the newly added pedestrian can be collected from the topography as shown in figure 12a. Figure 12b illustrates a scene after loading the newly created scenario scene from the code. This visualization is here to check if the pedestrian is correctly instantiated.

Figure 12: Adding a new pedestrian in the corner.



```
      "maximumSpeed" : 2.2,
      "acceleration" : 2.0,
      "footstepHistorySize" : 4,
      "searchRadius" : 1.0,
      "walkingDirectionSameIfAngle
      "walkingDirectionCalculation
    },
    "source" : null,
    "targetIds" : [ ],
    "nextTargetListIndex" : 0,
    "isCurrentTargetAnAgent" : fal
    "position" : {
      "x" : 11.7,
      "y" : 1.6
    },
    "velocity" : {
      "x" : 0.0,
      "y" : 0.0
    },
    "freeFlowSpeed" : 1.4605142244
    "followers" : [ ],
    "idAsTarget" : -1,
    "isChild" : false,
    "isLikelyInjured" : false,
    "psychologyStatus" : {
      "mostImportantStimulus" : nu
      "threatMemory" : {
        "allThreats" : [ ],
        "latestThreatUnhandled" :
      },
      "selfCategory" : "TARGET_ORI
      "groupMembership" : "OUT_GRO
```

(a) Newly added pedestrian (blue dot in the corner) and its coordinates (data on the right).



(b) Simulation scene after loading newly created corner scenario.

It is crucial to add the new pedestrian before generating the new corner scenario. Listing 1 below shows a summarized version of how a pedestrian is added. The fields that are commented out (marked with #) are usually in seen in the GUI when inserting a new pedestrian but with these lines the console would not run the created scenario which is why they are removed.

Listing 1: Method for adding a new pedestrian into a scenario.

```
def add_pedestrian(self, id: int, x: float, y: float, targetIds: list,
    freeFlowSpeed: float = 1.182125, likelyInjured: bool = False):
        # Create a new pedestrian
        pedestrian = {
```

```
        "source" : None,
        "targetIds" : targetIds,
        "position" : {
            "x" : x,
            "y" : y
        },
        # "NextTargetListIndex" : 0,
        "freeFlowSpeed" : freeFlowSpeed,
        "attributes" : {
            "id": id,
            "radius": 0.2,
            "densityDependentSpeed": False,
            "speedDistributionMean": 1.34,
            "speedDistributionStandardDeviation": 0.26,
            "minimumSpeed": 0.5,
            "maximumSpeed": 2.2,
            "acceleration": 2.0,
            "footstepHistorySize": 4,
            "searchRadius": 1.0,
            # "angleCalculationType": "USE_CENTER",
            # "targetOrientationAngleThreshold": 45.0
        },
        "idAsTarget": -1,
        "isChild": False,
        "isLikelyInjured": likelyInjured,
        # "mostImportantEvent": None,
        # "salientBehavior": "TARGET_ORIENTED",
        "groupIds": [ ],
        "trajectory": {
            # "footsteps": [ ]
        },
        "groupSizes": [ ],
        "modelPedestrianMap": { },
        "type": "PEDESTRIAN"
    }

    # Append the pedestrian to the list of pedestrians
    self.dynamicElements.append(pedestrian)
```

3.3 Evaluation

In this subsection it is to be evaluated how long it takes for the newly generated pedestrian to reach the target compared to the others. This can be done by assessing the output `postvis.traj` file which contains data about the starting time and end time of each simulated pedestrian. Therefore, simply subtracting one value with the other results in time taken to reach the target. The Python file `evaluate_scenario_output.py` extracts data from a given output file and computes the pedestrian times. In the end, the results are printed out in the console.

Figure 1 shows the calculated times from the corner pedestrian versus all the others given the simulation scenario described in this chapter.

Table 1: Times taken for corner pedestrian and other pedestrians to reach the target.

Pedestrian	Time Taken (mean)	Max	Min
corner	0.6119		
others	0.6066	0.7352	0.4952

An interesting finding occurs when comparing the mean of the time taken by all the other pedestrians to reach the target with the time of the corner pedestrian: the mean is lower than the corner pedestrian's time meaning overall other pedestrians are faster to reach the target than the corner one even though he is geographically closer to the target. However, by comparing the slowest pedestrian with the corner one, it can be seen that the corner pedestrian is not the last to reach the end of the corner.

Report on task 4, Integrating a new model

4 SIR Model in Vadere

4.1 Description of the setup

The main code base of vadere is contained within the vadere folder of the git repository. It contains several components of the java application in the form of high level classes. The main function of the application responsible to start the application in a GUI is written in the class `VadereApplication` in `org.vadere.gui.projectview`.

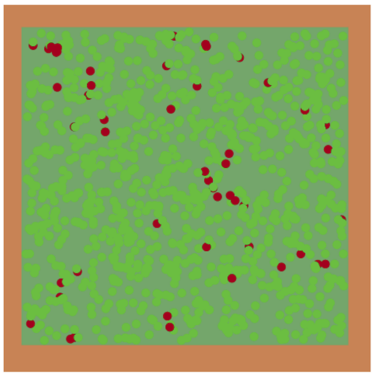
The `SIRGroup` class belongs to the package `org.vadere.simulator.models.groups.sir` and is inherited from the `AbstractGroupModel` class. It represents a group of pedestrians in the simulation that are subject to the SIR infection dynamics. The class manages group members and supports basic group functionalities such as adding and removing members. Key methods in this class include `addMember(Pedestrian ped)`, which adds a pedestrian to the group, `removeMember(Pedestrian ped)`, which removes a pedestrian from the group, and `getMembers()`, which returns the list of group members. The `SIRGroupModel` class also belongs to the package `org.vadere.simulator.models.groups.sir`. This class manages the simulation of the spread of infection among pedestrians using the SIR model, specifically handling transitions between susceptible and infective states. It does not include removal/recovery state of the pedestrians initially but is added by us.

4.2 Optimizing `update` method in the SIR Group Model class

We made all of the changes in the update method of the `SIRGroupModel` class. This function moves the simulation one step ahead in time. The size of the time step is passed as a parameter to the function. We made significant changes to the update method like making accurate simulation of infection rate and removal rate along with making it more efficient. The `LinkedCellsGrid` is a spatial data structure provided to make the method more efficient. Currently, the method looks at all pedestrians for each pedestrian which is $O(n^2)$ complexity. The linked cells enables to use spatial information to pick the neighbours. We did other optimizations like unnecessary loops and better checks to improve the efficiency of the code to pass the remote tests on Artemis. The linked cells grid was not used directly because of an issue with a wrong return-type in a getter-function that we were unable to solve. In consequence, we could not use it's constructor and did not want to change the signature of the class.

4.3 Experiments

We designed two experiments. First one is visualised in figure 13. We generated 1000 pedestrians as instructed in the task 4 of exercise sheet. The target was set to not absorbing. The target is placed on top of the source so all the pedestrians are already over the target and hence do not move. The number of infections at start is 10 out of a 1000 and infection rate is 0.01. Within a few time steps the pedestrians get infected based on the infection rate. The figure 14 shows how the number of infections grow over time for each infection rate. For infection rate 0.01 it took about 8 seconds to infect half the population, for a rate of 0.02 it took about 4 seconds and for a rate of 0.1 it takes less than a second.


(a) Initial setup of the 1000 stationary pedestrian test

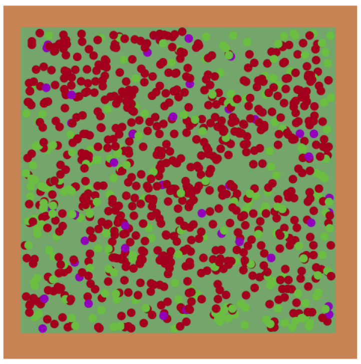

(b) State of 1000 stationary pedestrian test after several time steps

Figure 13: 1000 pedestrian test. Red color represents infected pedestrian, green represents susceptible and purple represents recovered/removed pedestrians.

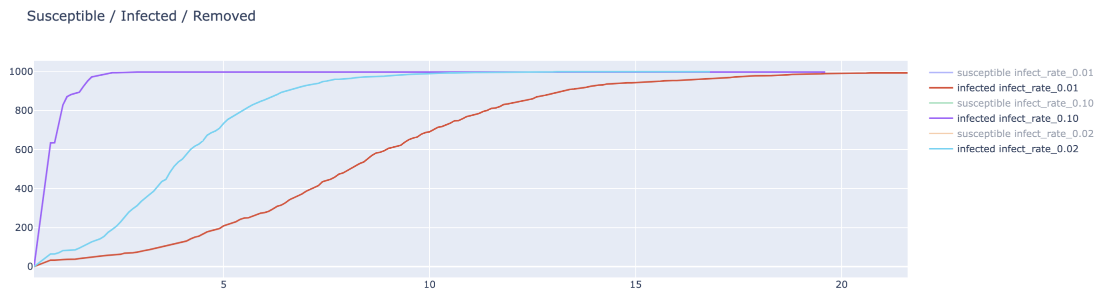
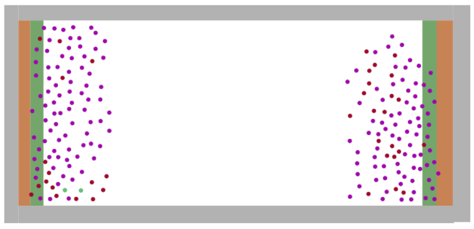
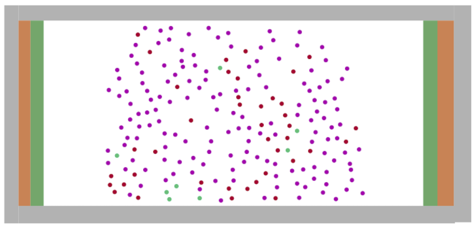


Figure 14: Number of infections plotted for different infection rates for the 1000 pedestrian test

Second experiment was to generate a corridor scenario and observe the pedestrians. The infections were not as high as the 1000 pedestrian test. It might be because of the large amount of space that the pedestrians have to move across the corridor. It is visualised in figure 15. Two groups of 100 pedestrians are moving in a corridoor. One from left to right and one from right to less. There infections are observed.


(a) Initial setup of the corridor scenario


(b) State of corridor scenario test after several time steps

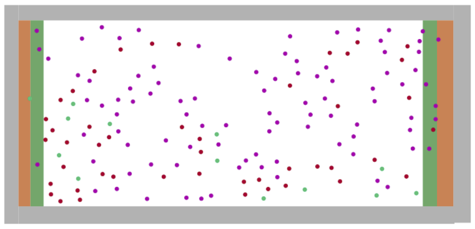

(c) State of corridor scenario test later

Figure 15: Corridoor scenario 40m x 20m. Purple represents susceptible pedestrians, Red represent infected pedestrians and green represent recovered/removed pedestrians

4.4 Decoupling the Infection Rate and the time step

Issues with the original approach: Time Step Dependency: The original infection probability calculation does not consider the length of the simulation step (simTimeStepLength). This is problematic because inf_rate was intended to represent the probability of infection per second. If your simulation updates every fraction of a second (say every 0.1 seconds), then using inf_rate directly means you are effectively multiplying the probability of infection tenfold over one second, resulting in a much higher rate than intended. Inconsistent Behavior: As you change the step size of your simulation (say from 0.1 seconds to 0.05 seconds per step), the behavior of your simulation regarding infection dynamics changes dramatically. This inconsistency can lead to incorrect interpretations and predictions, particularly when trying to simulate and analyze an infectious disease spread. Difficulty in Interpretation: When inf_rate is directly used as the infection probability per step without adjustment for the actual duration of each step, it becomes difficult to interpret what inf_rate really means in the context of different time steps. With this consideration, the infection rate now is calculated as the probability of infection a pedestrian per second instead of per time step, in another word, for each pedestrian, the probability of an infection per second is the sum of the infection probability for every time step in that second.

Algorithm 1 Update infection and recovery status of pedestrians

```
inf_rate := attributesSIRG.getInfectionRate() × this.simTimeStepLength;
if (this.random.nextDouble() < inf_rate) then
    elementRemoved(p);
    assignToGroup(p, SIRType.ID_INFECTED.ordinal());
    count_inf += 1;
    break;
end if
```

4.5 Possible Extensions of the model

Different recovery rate: The likelihood of recovery can be tailored to reflect individual differences among pedestrians, particularly considering factors like age . Younger people between the age of 18 to 40, typically recover more quickly and are more resilient, whereas older people of the age 60 and above may face slower recovery rates.

Health regulation: In the current model, a pedestrian doesn't change their path regardless of how many people or whether there are anyone infected near him or her, this is not usually the situation in real life where there are health regulations such as quarantine or where the healthy people are often suggested to stay away from the infected.

Different infection rate for pedestrians: in real life people have different immune system, therefore not everyone is susceptible to an infection, introducing this to the current model would also make this model more realistic.

Report on task 5, Analysis and visualisation of results

5 Analysis & Visualisation of Results

5.1 Implementing the 'recovered' state

This tasks goal is to add a 'recovered' state to the pedestrians, and analyze and visualize the results obtained. The first sub-task was to add the 'recovered' state. Recovered pedestrians cannot get re-infected, and cannot infect other pedestrians. The implementation for the SIR model can be found in the Java package `VadereSimulator.src.org.vadere.simulator.models.groups.sir`. The states of the pedestrians in the `SIRGroupModel` are implemented with an enum named `SIRType`. In the `SIRType` enum, there are 3 states: `ID_INFECTED`, `ID_SUSCEPTIBLE` and `ID_REMOVED`. The 'removed' state was not used at all in the SIR model source code, so in the implementation, the state `ID_REMOVED` should be interpreted as the 'recovered' state. When a pedestrian recovers, they get assigned the 'recovered' state, and thus continue to exist in the simulation, but do not interact with other pedestrians, i.e. they can't get re-infected and can't infect other pedestrians.

In Vadere, in the simulation model's configuration JSON-file, the user has to specify attributes regarding the actual simulation. These attributes include the key-value-pair 'recoveryRate', which expects a value between 0 and 1. This value is loaded into the simulation, and used as the probability for an infected pedestrian to recover from the infection in a given time window. The task was to calculate the recovery rate to reflect the probability of a pedestrian getting recovered after one second in simulation time. Similarly to the infection rate calculation, the updated recovery rate r' can be calculated with:

$$r' = r \cdot \Delta t,$$

where r is the original recovery rate, and Δt is the time step length of each update call. This updated probability is used to determine whether a pedestrian recovers in a time step or not. For each infected pedestrian in the simulation, the following recovery algorithm is executed in each update of the simulation state.

Algorithm 2 Calculate the updated recovery rate and determine if a pedestrian recovers

```
rec_rate := attributesSIRG.getRecoveryRate() · this.simTimeStepLength;
if (this.random.nextDouble() < rec_rate) then
  elementRemoved(p);
  assignToGroup(p, SIRType.ID_REMOVED.ordinal());
end if
```

In algorithm 2, first the recovery rate is updated. Then, a random number is produced, and if it is less than the updated recovery rate, then the infected pedestrian is removed from the group of infected and assigned to the recovered group. This algorithm is independent of where the infected pedestrian is, and how many other pedestrians are nearby.

A test scenario for validating the functionality of the 'recovered' state and a pedestrian recovering can be found in the exercise folder in './scenarios/task_5_recovery_test.scenario'. The aim of the test is to see that a pedestrian is assigned the 'recovered' state during the simulation. The test is a simple test, where a single infected pedestrian is placed on the grid. The test's duration is 10 seconds, and the recovery rate has been set to a high number, so that the probability of the pedestrian not recovering during the simulation is very low.

Running the scenario, and then analysing the output file 'SIRinformation.csv' shows, that in the beginning, the pedestrian has the groupId '0', i.e. it is infected. During the simulation, the pedestrian's groupId changes from '0' to '2', indicating that the pedestrian has indeed recovered. This simple test leads to believe that the 'recovered' state is properly implemented.

Furthermore, to validate that the implemented recovery rate reflects the probability of a pedestrian getting recovered after one second in simulation time, a second test was implemented. The number of recovered pedestrians after time t follows the function:

$$f(n,t) = n \cdot (1 - (1 - r_{rec})^t),$$

where n is the number of infected pedestrians in the beginning, t is the length of the simulation in seconds and r_{rec} is the probability of a pedestrian recovering in one second of simulation. This model assumes that the infected pedestrians do not infect other pedestrians during the simulation.

To validate the implementation of the recovery rate, we want to find the recovery rate that recovers circa half the population after 10 seconds of simulation. To achieve this, we solve the function for r_{rec} when $f(1000, 10) = 500$. The function yields $r_{rec} \approx 0.067$.

In line with the calculations, we set 1000 infected pedestrians in the simulation and run it for 10 seconds. The recovery rate is set to 0.067. The test scenario can be found in './scenarios/task_5_prob.scenario'.

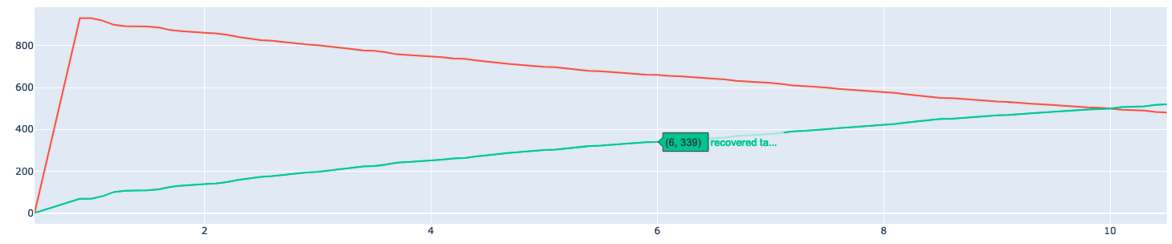


Figure 16: Results of probability test

In figure 16, the red line shows the number of infected pedestrians and the green line the number of recovered pedestrians over time. After running the simulation for 10 seconds, the lines are really close to meeting at $y = 500$, which can be interpreted as that the recovery rate is correctly implemented. The visualization software initializes all pedestrians as susceptible, and only updates them after the first time step, hence the infected graph looks a bit weird, but in the simulation, all pedestrians are infected from the beginning.

5.2 Extending the visualization application to work with the 'recovered' state

Provided with the exercise files was an application that visualizes the number of infected and susceptible pedestrians as a function of time. The application expects to find a file called 'SIRinformation.csv' in the output folder of the Vadere software. The file has the columns 'PedestrianId', 'timeStep', and 'GroupId-PID5'. Each row in the file describes the state of one pedestrian in the given time step. The application transforms the data in such a way, that each row contains the timestamp of the simulation, and the number of infected and susceptible pedestrians.

To be able to use the application in the following tasks, it had to be modified slightly, so that the application would also visualize the number of recovered pedestrians. Most of the changes to the application are done in the function `file_df_to_count_df`, which performs the aforementioned data transformation. The function works by iterating the pedestrian ids and checking if the state of the pedestrian has changed from susceptible to infected over time, updating the counts as a change of state is found. To properly track the number of recovered pedestrians, a second check was added. The newly added check detects the change from the state infected to the state 'recovered' in a time step, and updates the counts respectively. Finally, in the function `create_folder_data_scatter`, a third plot was added, which plots the number of recovered pedestrians as a function of time.

5.3 Analysis and visualization

5.3.1 Setup

The setup for this experiment is illustrated in figure 17. 1000 pedestrians are randomly placed inside a 20 × 20 meter grid. 10 of them are infected from the start. In the next part, we run the same setup multiple times with different recovery and infection rates, analyse and visualize how the pedestrians' states change over time. This scenario is found in './scenarios/task_5_2_1.scenario'.

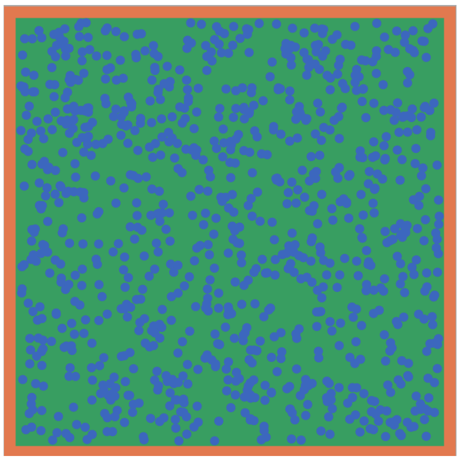


Figure 17: Setup for experiment

5.3.2 Testing, analysis and visualization

As stated previously, in this section, the same scenario is being run with different infection and recovery rates. Each simulation is 50 seconds long, and the parameter infection and recovery rates can be found in the caption of each simulation's graph. In each of the graphs, the blue line shows the number of susceptible pedestrians, red the number of infected and green the number of recovered pedestrians as a function of time.

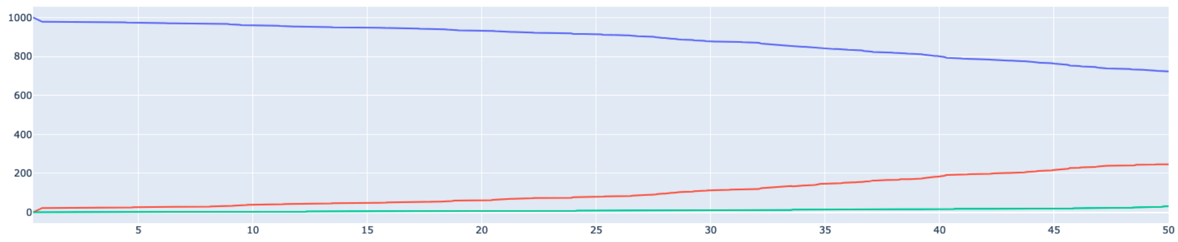


Figure 18: Infection rate and recovery rate both set to 0.01

Figure 18 shows that when both the infection and recovery rate is set to 0.01, the growth of the infected population is slow but steady. The recovered population also grows steadily, but even slower, since the population that can recover is significantly smaller than the population than the population that can be infected.

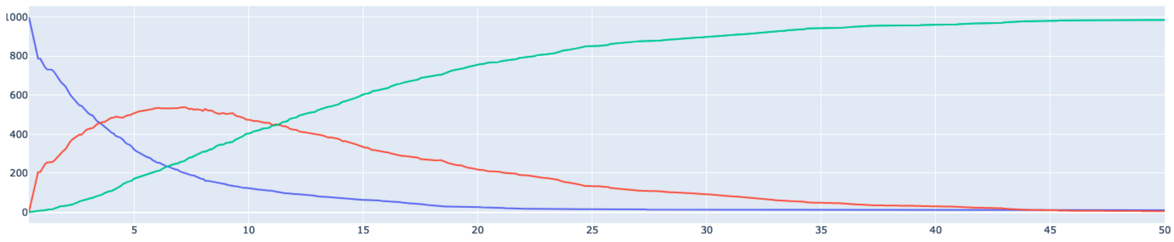


Figure 19: Infection rate and recovery rate both set to 0.1

Setting both the infection and recovery rate to 0.1, on the other hand, shows a fast growth in the infected population. It can be interpreted from figure 19, that the number of infected pedestrians peak at around 600 pedestrians. Afterwards, since the recovery rate is also high, the infected population slowly declines, but still infecting new pedestrians, since the number of recovered pedestrians keeps growing. After 50 seconds of simulation, the infection has spread almost throughout the whole population, and the population is almost completely recovered from the breakout.

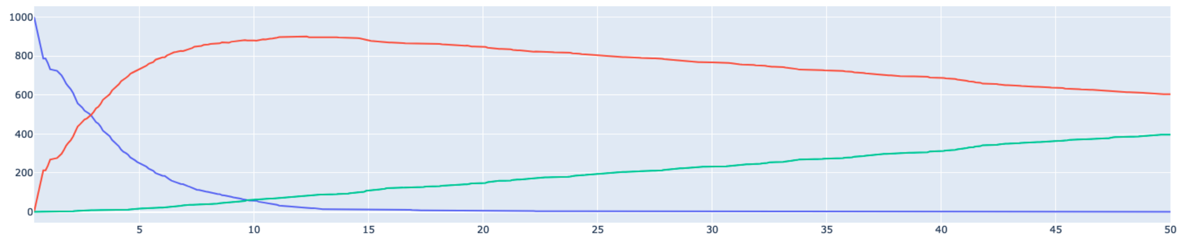


Figure 20: Infection rate set to 0.1 and recovery rate set to 0.01

Figure 20 shows the spread of the infection when the infection rate is set to 0.1 and the recovery rate to 0.01. The epidemic naturally lasts longer, and the infected population peaks higher, at almost 900 pedestrians, and at almost the same time point than the previous example. The infected population recovers from the infection much slower.

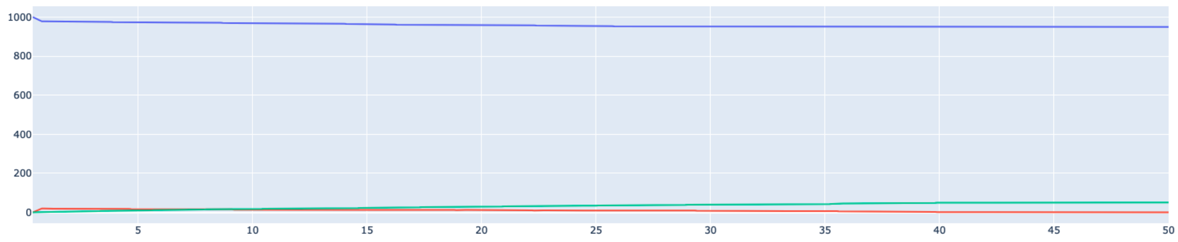


Figure 21: Infection rate set to 0.01 and recovery rate set to 0.10

Figure 21 shows the spread of the infection when the infection rate is set to 0.01 and the recovery rate to 0.1. The graph shows, that when the recovery rate is much higher than the infection rate, the spread of the disease ends before it can even begin properly. Only a small percentage of the population actually gets infected, while most of the population stay susceptible to the disease, without ever getting infected.

5.4 Supermarket Scenario

5.4.1 Experiment Setup

The experimental setup for the supermarket scenario is illustrated in Figure 22. In this simulation, customers are represented as pedestrians entering the simulation from a source point, marked in green. The layout includes vertical barriers that mimic the grocery sections typical of a standard supermarket. Customers are directed towards various objectives including aisles, cashiers, and exits, which are represented by orange squares at designated points within the simulation space, including an exit at the top of the layout.

Each source point is associated with all target IDs, allowing for random movement paths through the aisles to the cash registers and exits, simulating the shopping process. Initially, we introduce a single infected individual among the customers (‘infectionAtStart‘ = 1), with an ‘infectionRate‘ of 0.005. The ‘recoveryRate‘ is set to 0 to monitor the spread of infection without recovery interference (also since the goal here is to find the relationship between social distancing and infection bahavior, we decided to take recovery rate out of the equation for the purpose of easier interpretation of the results).

To mimic a realistic scenario, considering the number of customers visit a supermarket (at least of the size 30m by 30m) daily, and the duration of the time in real life which customer spend in the supermarket, we scale the simulation to a slightly densely populated area. We model 20 customers within a 10cm × 10cm area, reflecting a typical customer density during normal operations. The personal space for each pedestrian (‘pedPotentialPersonalSpaceWidth‘) is set to 1.0, and the total number of customers in the simulation is fixed at 20.

In our supermarket scenario simulation, we aim to provide a realistic representation by considering both the customer density and the average time customers spend in the supermarket. We simulate a condensed period within a busy supermarket environment to analyze the dynamics of customer interaction and potential virus transmission under high-density conditions.

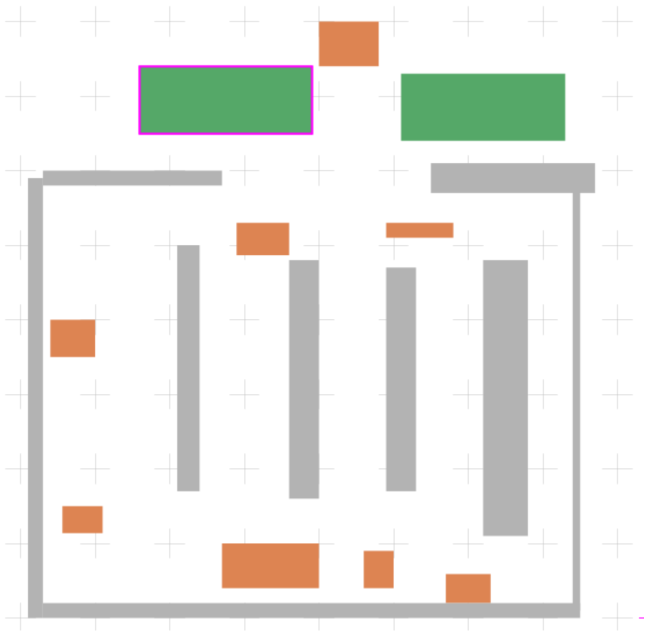


Figure 22: Supermarket layout

The simulation visualizations, which explore different social distancing parameters, are discussed below. These visualizations adjust the ‘pedPotentialPersonalSpaceWidth‘ to 0.5 23, 1.024, and 2.025 respectively, allowing us to observe the impact of varying personal space widths on virus transmission among customers in a simulated supermarket environment.

- For a personal space width of 0.5, the customer density is higher, potentially leading to quicker spread of infection but also faster stabilization of infection rates due to limited movement space.
- With a personal space width of 1.0, the simulation reflects a moderate approach to social distancing, balancing movement and infection spread.
- At a personal space width of 2.0, the simulation showcases extensive social distancing, resulting in a significant reduction in the rate of infection transmission. Notably, the infection dynamics continue to change over a longer duration as the customers have more space to avoid contact.

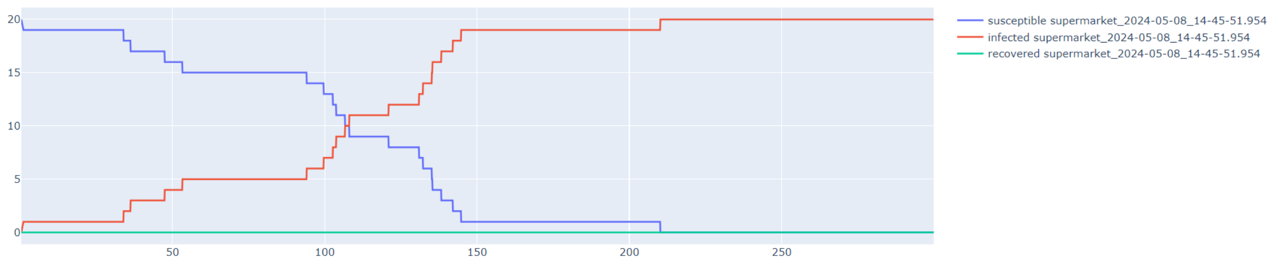


Figure 23: Scenario with 20 customers, 1 infected person in the beginning and pedPotentialPersonalSpaceWidth 0.5

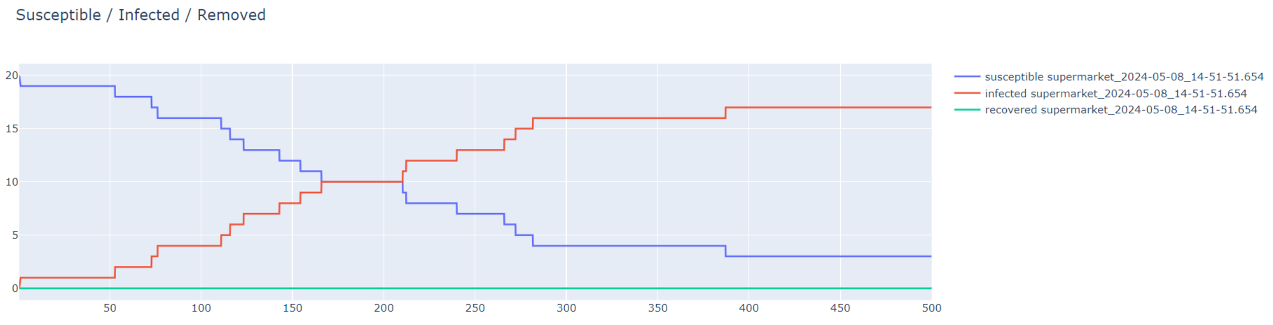


Figure 24: Scenario with 20 customers, 1 infected person in the beginning and pedPotentialPersonalSpaceWidth 1.0

Due to the variations in social distancing measures, the simulation time was adjusted accordingly: shorter durations were used when the number of infected individuals stabilized quickly, and longer durations were applied when social distancing was extensive enough to delay the stabilization of infection rates.

5.4.2 Results and interpretation

Analysis of Simulation Results

The outcomes of the simulation distinctly demonstrate a robust correlation between the `pedPotentialPersonalSpaceWidth` and the rate of infection among the customers. It is evident that increased social distancing, represented by larger personal space widths, significantly slows the rate of infection transmission. These observations are particularly pertinent under the controlled conditions of our simulation,

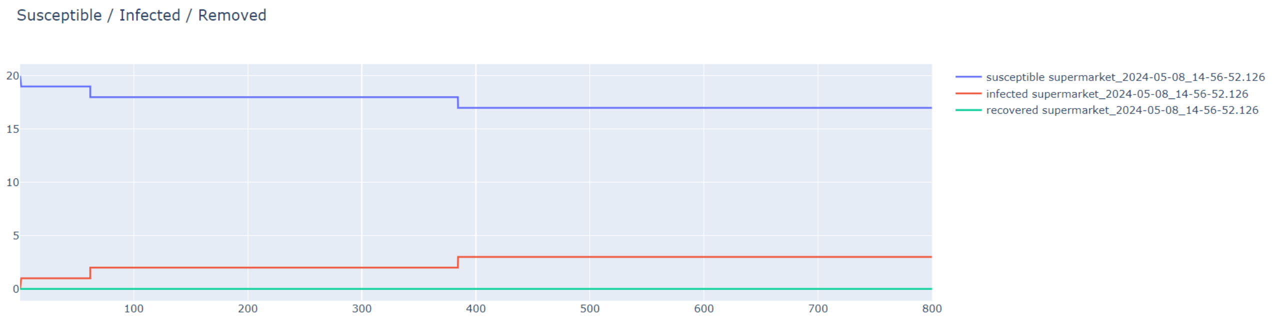


Figure 25: Scenario with 20 customers, 1 infected person in the beginning and pedPotentialPersonalSpaceWidth 2.0, note that the spread of the infection is slower in this case

where the number of customers was fixed at 20 per simulation iteration, and the infection rate was maintained at a relatively low level of 0.005.

Implications of Increased Density and Infection Rates

However, it is important to consider the limitations of the current simulation parameters. The effect of social distancing diminishes under conditions of higher crowd density or increased infection rates. In scenarios where either the number of customers is significantly higher or the infection rate is elevated, the impact of increased personal space on slowing down the infection rate is notably reduced. This phenomenon is illustrated in the figures presented below 26 27 28, which highlight the reduced efficacy of social distancing measures in denser or more highly infectious environments.

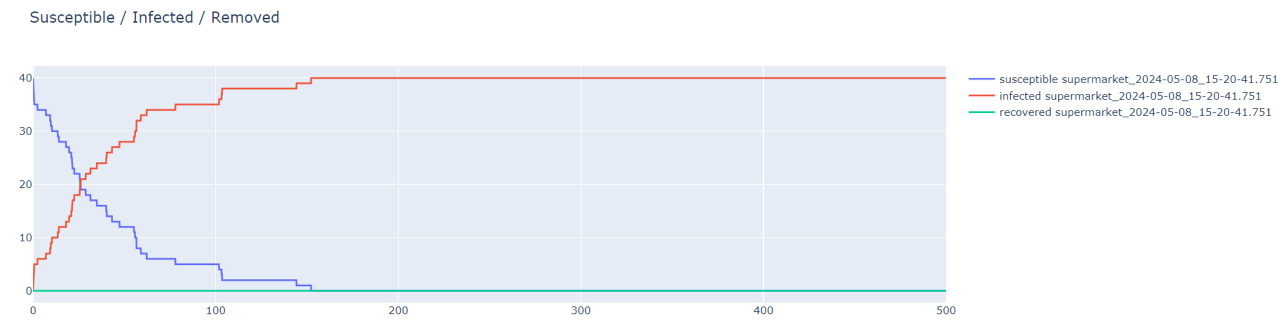


Figure 26: Scenario with 40 customers, 5 infected at the beginning and pedPotentialPersonalSpaceWidth 0.5

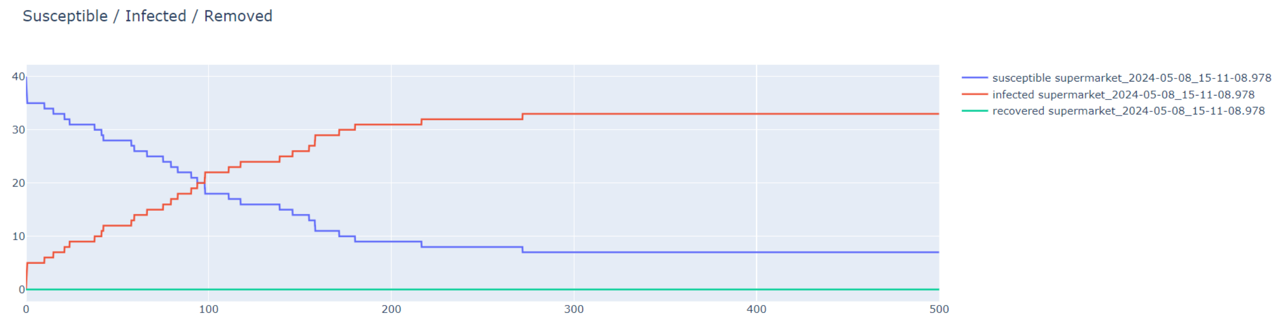


Figure 27: Scenario with 40 customers, 5 infected at the beginning and pedPotentialPersonalSpaceWidth 1.0

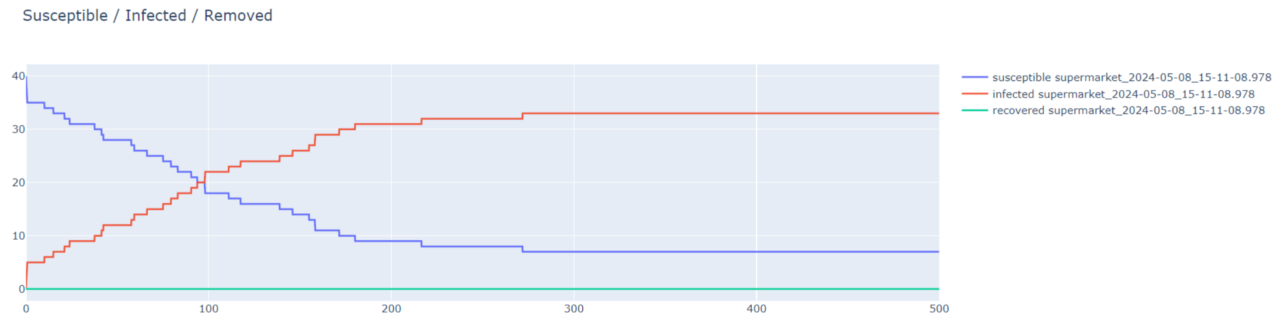


Figure 28: Scenario with 40 customers, 5 infected at the beginning and pedPotentialPersonalSpaceWidth 2.0

Conclusion

These findings underscore the need for a multi-faceted approach to managing infection spread in public spaces like supermarkets, especially under varied and dynamic conditions. While social distancing remains a critical measure, its effectiveness is contingent upon the specific characteristics of the crowd and the virulence of the pathogen involved. As such, reliance solely on social distancing may not suffice in all situations, particularly in highly congested settings or with pathogens that have a higher transmission rate.

References

[1] RiMEA. (2016) *Guideline for Microscopic Evacuation Analysis.*, RiMEA e.V., 3.0.0 edition.

[2] Christina Mayr and Gerta Köster. Social Distancing with the Optimal Steps Model. *ArXiv Pre-Print*, 2022.

[3] Felix Dietrich and Gerta Köster. Gradient navigation model for pedestrian dynamics. *Physical Review E*, 89(6):062801, 2014.

[4] Dirk Helbing and Péter Molnár. Social Force Model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995.