Report for exercise 2 from group K

Tasks addressed:	5
Authors:	Oliver Beck (03685783)
	Junle Li (03748878)
	Chenqi Zhou (03734992)
Last compiled:	2021–05–12
Source code:	https://github.com/Linnore/MLCMS-EX2-GroupK

The work on tasks was divided in the following way:

Oliver Beck (03685783)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%
Junle Li (03748878)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%
Chenqi Zhou (03734992)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%

Abstract

The Vadere software is a tool for the simulation and visualization of human crowds. It is open source with an active development team.

In this exercise, we learned how to use its graphical user interface, create, run, and modify simulation scenarios, and integrate a new SIR model into the software, which will be useful for working with Vadere and other simulation tools.

Vadere	master branch
Python	3.8.8
IntelliJ IDEA	2021.1.1 Community Edition
Jupyter Notebook	6.0.3
Vsiual Studio Code	1.55.2

Table 1: Software versions

Report on task 1, Setting up the Vadere environment

Task description

1. Download the Vadere software (not the source code version, just the compiled JAR files at *http : //www. vadere.org/releases/*).

2. Start the graphical user interface of the software, and re-create the RiMEA scenarios 1 (straight line) and 6 (corner), as well as the “chicken test” you had to implement in the first exercise. Use the Optimal Steps Model with its standard template. Report your findings and compare these scenarios with your own cellular automaton in the first exercise,

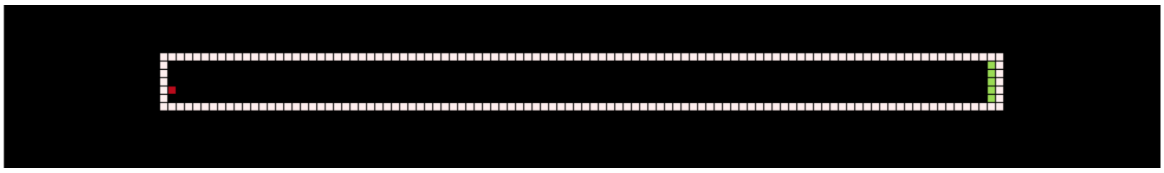
Results

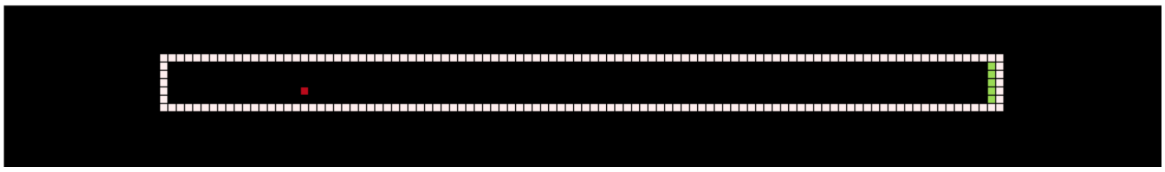
We will compare the test results of the Vadere Implementation individually, for the straight line, chicken test and, corner scenario. Afterwards we will evaluate the differences and similarities of the two implementations in more detail, regarding model trajectories, visualization, user interface and general observations.

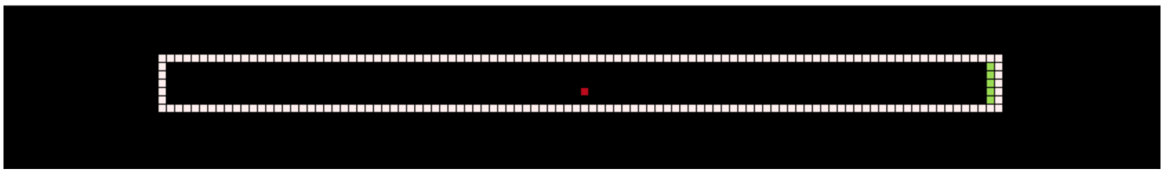
Comparing our implementation with Optimal Step Model in Vadere

RiMEA Test 1: Straight Line

Task description Simulate pedestrians with well defined speed. 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 percent (walking speed) are set as imprecise values, then the following requirement results with a typical pedestrian speed of 1.33 m/sec: 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/sec is set as the speed. Compare your Implementation to the Vadere Implementation.


(a) t=0s


(b) t=5s


(c) t=15s

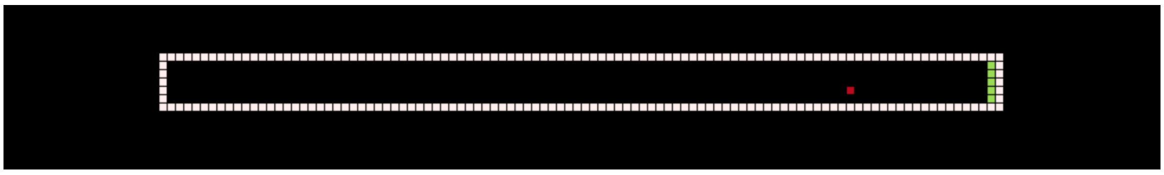
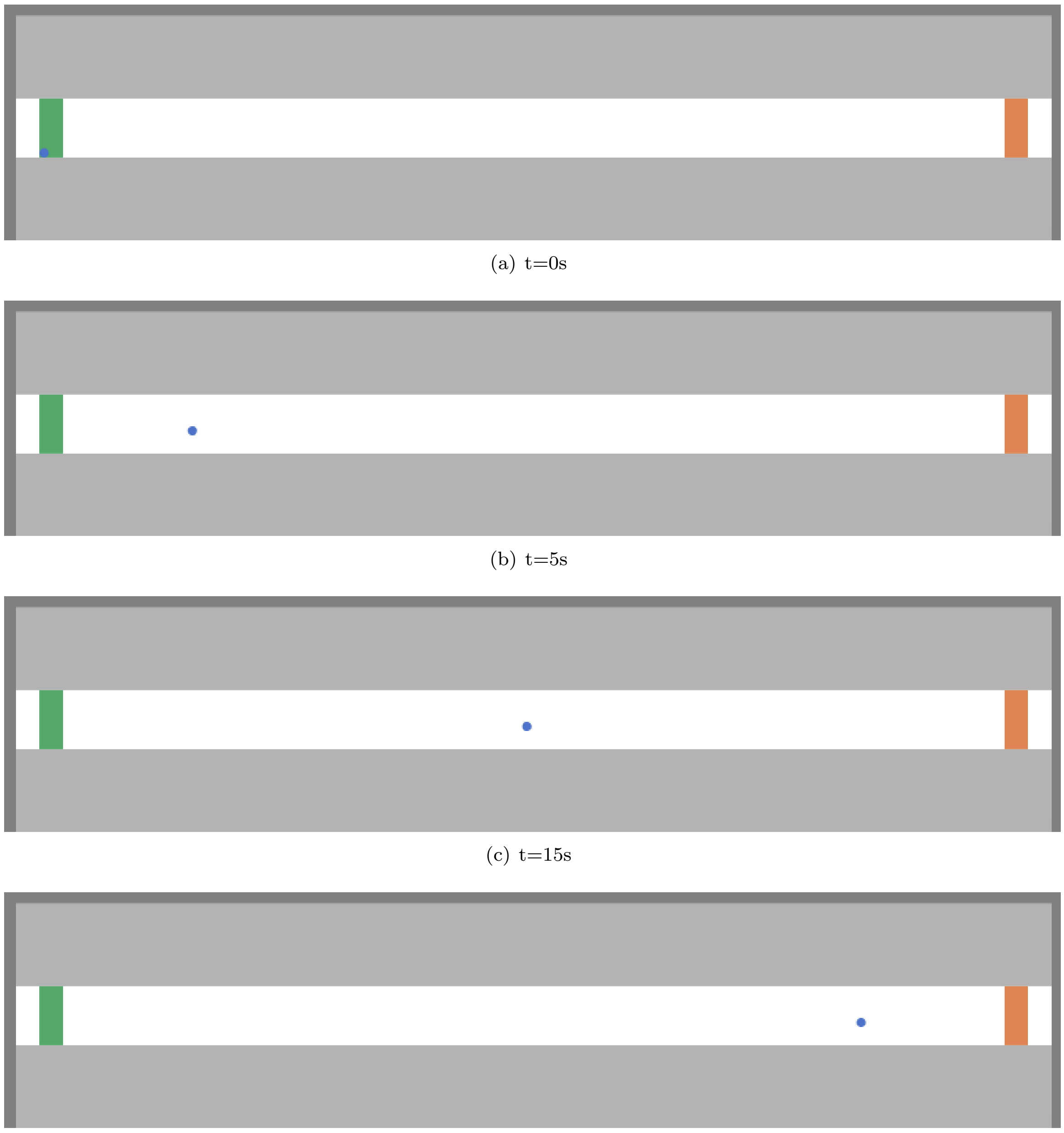

(d) t=25s

Figure 1: A Run of our Cellular Automaton Implementation of RiMEA Test1 in Exercise1


(a) t=0s

(b) t=5s

(c) t=15s

(d) t=25s

Figure 2: A Run of the Optimal Steps Model in Vadere

The results of the comparison As can be seen in the visual timestamps the overall results for the RiMEA test 1 scenario of the Vadere Simulation and our implementation in exercise 1 nearly doesn't differ at all for the actual test results. The results, in other words the elapsed time, in that the pedestrians reach the target in 40m distance is in the required interval for the test (26 to 34 seconds). This isn't surprising since the task is very simple and the simple nature of a constant time moving pedestrian doesn't leave a lot of possibility for different results in combination with a functional simulation. One noticeable distance is that Pedestrians move to the center of the corridor and prefer that over walking alongside the wall. This is due to the fact that our implementation only accounts for avoiding obstacles, not for keeping distance from them, which also wasn't required or intended.

RiMEA Test 6: Corner

Task description Movement around a corner: Twenty persons moving towards a corner which turns to the left will successfully go around it without passing through walls. Compare your Implementation to the Vadere Implementation.

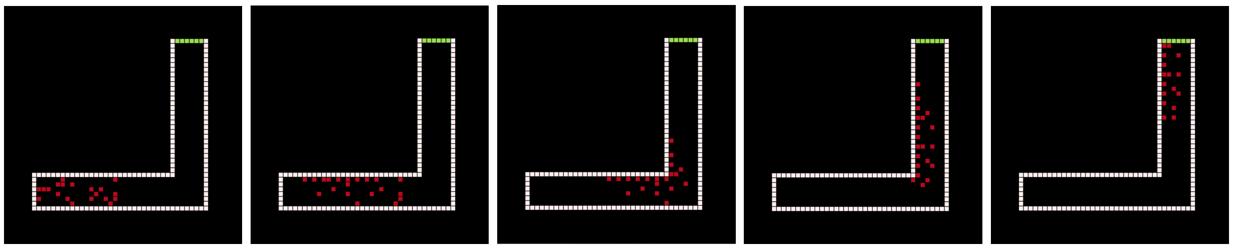

(a) t=0s (b) t=2s (c) t=5s (d) t=8s (e) t=11s

Figure 3: A time lapse of different stages of the test with shortest-path distance cost function.

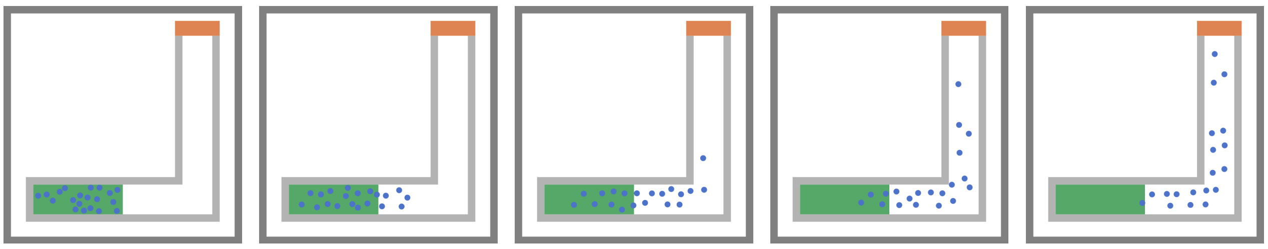

(a) t=0s (b) t=2s (c) t=5s (d) t=8s (e) t=11s

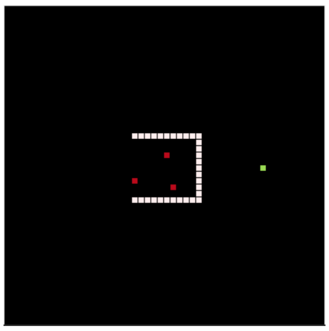
Figure 4: A time lapse of different stages of the test using Vadere OSM.

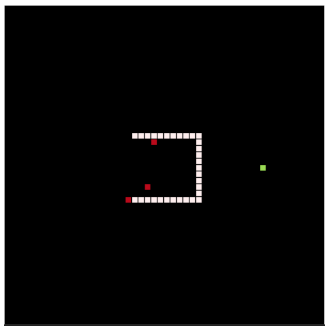
The results of the comparison Other than for the single pedestrian in RiMEA Test1, corner scenario is more complex. If is noticeable that the pedestrians keep more distance in the Vadere test runs compared to our implementation. This is likely due to a different distance cost function and avoidance weights for other pedestrians. When we changed the weighting of distance matrix and avoidance cost functions in our implementation, we observed different behaviours. We actually addressed something called dominance of the distance matrix or dominance of the avoidance in our first exercise. In the case of Vadere's simulation, it seems very similar to the avoidance dominant implementation we tried in Exercise 1. In both cases pedestrians might rather not advance and keep their distance to others than proceeding nearer to the target. This hold true for the Running Time and the Pedestrian distribution.

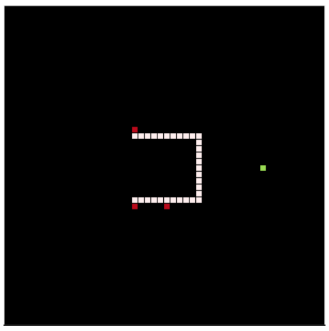
Figure 3 shows the one we finally implemented in exercise 1, in which we valued progression towards the target over social distancing. The Result is that the progression of the Pedestrians that are nearest to the target in our implementation is similar to the one in Vadere (since they aren't influenced a lot by nearby others). However, when the last pedestrian in Vadere Simulation is still in the spawning area, the last pedestrian is about to cross the corner in our implementation. What is also very noticeable is that our simulation fits up to 4 Pedestrians in parallel, while the Vadere Simulation seems to proceed into a "civilized" pair of rows.

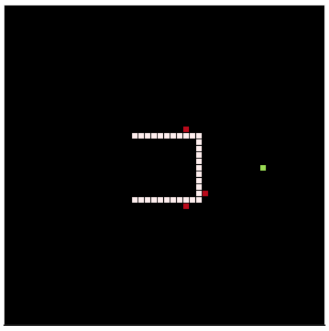
Chicken Test in Exercise 1

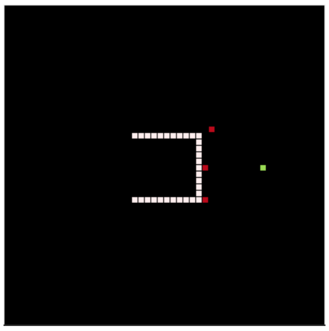
Task description Implement rudimentary obstacle avoidance for pedestrians by adding a penalty (large cost in the cost function) for stepping onto an obstacle cell. Determine the impact of obstacle avoidance on whether pedestrians can reach the target in the "chicken test" scenario with U-shaped obstacles.

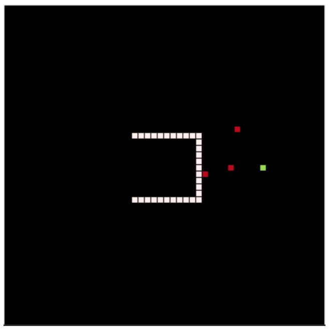

(a) t=0.6s

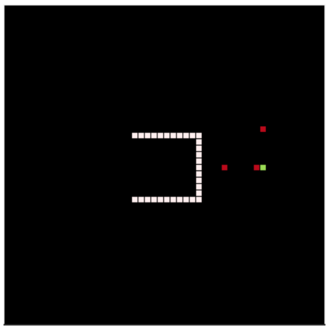

(b) t=1.5s

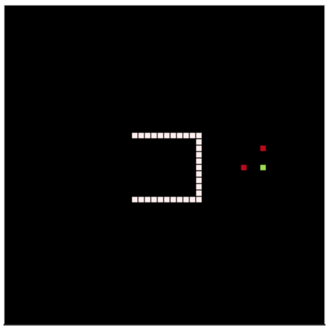

(c) t=3.0s


(d) t=5.1s


(e) t=5.9s


(f) t=6.9s


(g) t=7.6s


(h) t=8.4s

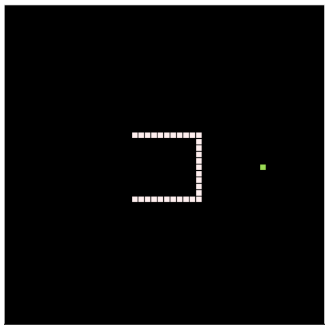
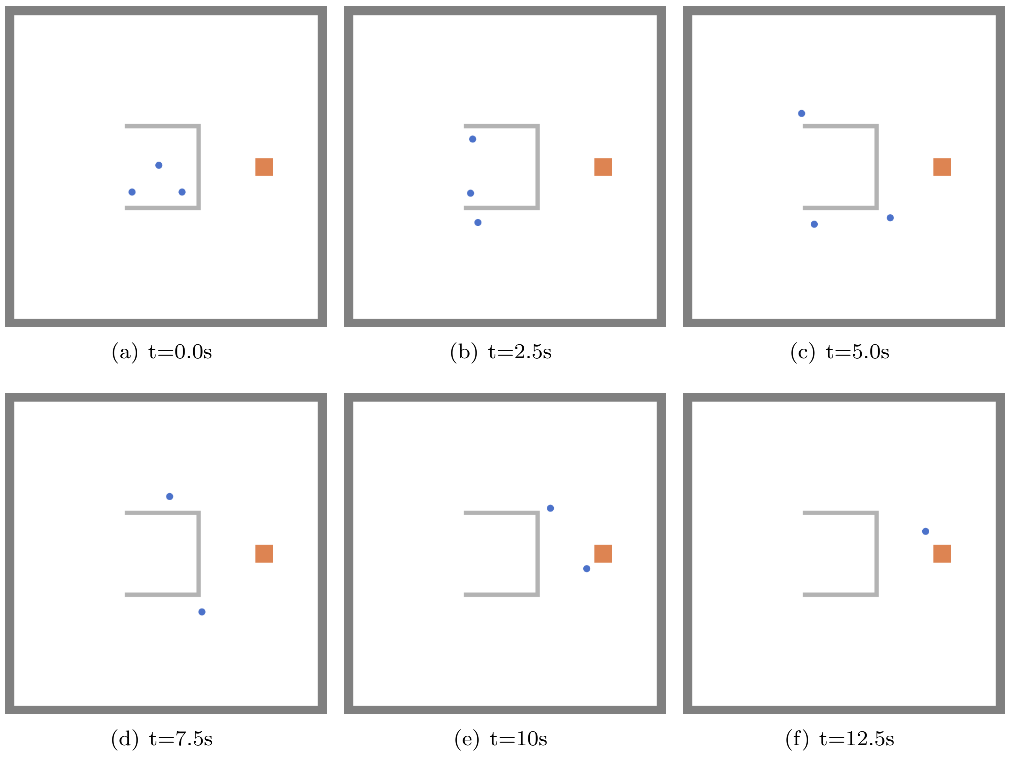

(i) t=8.9s

Figure 5: With obstacle avoidance


(a) t=0.0s (b) t=2.5s (c) t=5.0s

(d) t=7.5s (e) t=10s (f) t=12.5s

Figure 6: With obstacle avoidance

The results of the comparison Again results are fairly similar. One thing we repeatedly noticed was that the pedestrians in the Vadere simulation try to keep distance from the walls and other pedestrians, which costs them more time as a shorter way theoretically would be possible when walking near a wall.

Conclusions for the comparison between our Cellular Automaton Implementation and the Optimal Steps Model in Vadere

Model Trajectories Vadere implements diagonal movement, avoidance between pedestrians and obstacles, and larger avoidance between pedestrians compared to our simulation. It seems that the Vadere simulation has some kind of "foresight" and anticipates when multiple pedestrians move towards the same area.

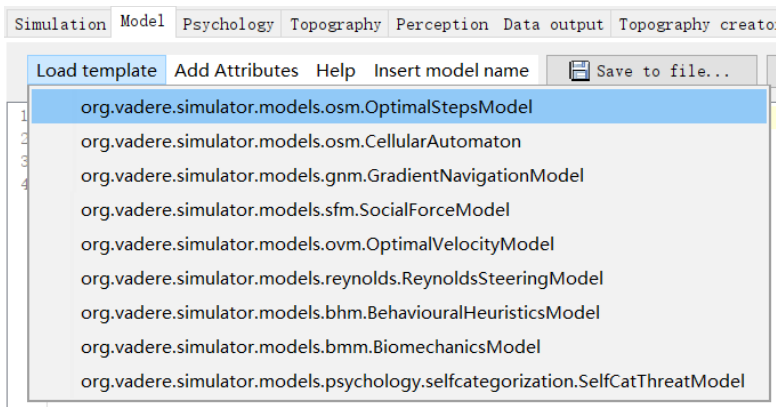
Visualization The simulation and the virtualization of our implementation is bound by the granularity of the grid we are using usually 0.33 meters. The visualization happens after discrete time steps due to the Cellular Automaton property. The "only" things we visualize are pedestrians, obstacles targets and their interactions, whereas Vadere offers multiple options regarding step size, trajectories, movement directions, different grouping, heat maps and many more.

User Interface Our implementation uses classic black and neon design, similar to popular dark mode. We also provides option to draw freely. Other than these, Vadere has much more functionalities.

General Observations Vadere provides more accurate visualizations and provides obviously more functionalities.

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

Task description



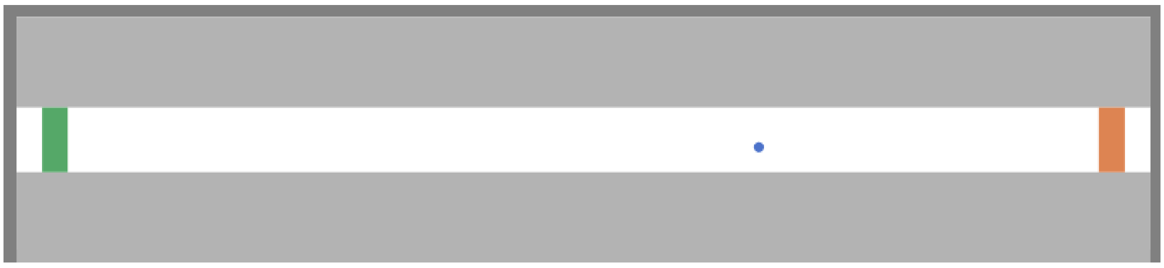
org.vadere.simulator.models.osm.OptimalStepsModel
org.vadere.simulator.models.osm.CellularAutomaton
org.vadere.simulator.models.gnm.GradientNavigationModel
org.vadere.simulator.models.sfm.SocialForceModel
org.vadere.simulator.models.ovm.OptimalVelocityModel
org.vadere.simulator.models.reynolds.ReynoldsSteeringModel
org.vadere.simulator.models.bhm.BehaviouralHeuristicsModel
org.vadere.simulator.models.bmm.BiomechanicsModel
org.vadere.simulator.models.psychology.selfcategorization.SelfCatThreatModel

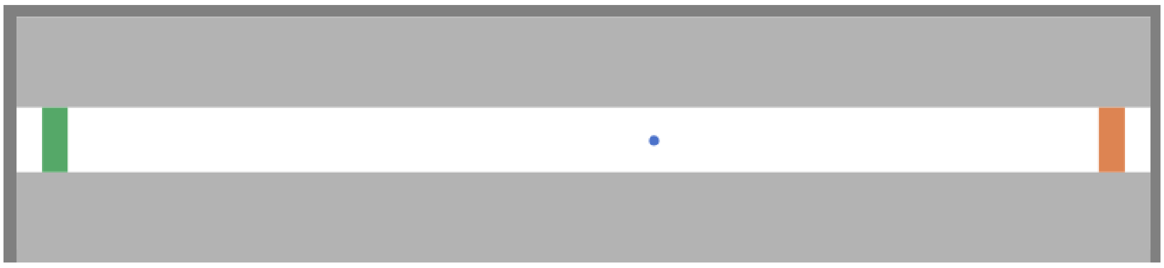
Figure 7: Changing the Model template in the Model tab

The Vadere software offers many different types of models for crowds. In this task, you have to change the model to the Social Force Model (SFM) from Helbing and co-authors. This model is already available as a template in the tab Model of a scenario. Re-run the three scenarios from the previous task, and report your findings. Compare the results between the two models and analyse the reason for the change in behavior. Do the same analysis with the Gradient Navigation Model (GNM) and compare all three models.

Result

Change the models of RiMEA scenario 1, scenario 6 and chicken test to “Social Force Model” (SFM), and “Gradient Navigation Model” (GNM) in turn.


(a) OSM (Simulation time:29.614s)


(b) SFM (Simulation time:34.4s)

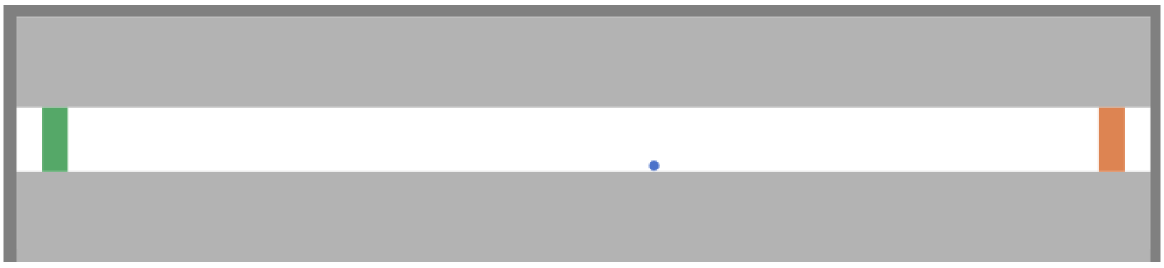
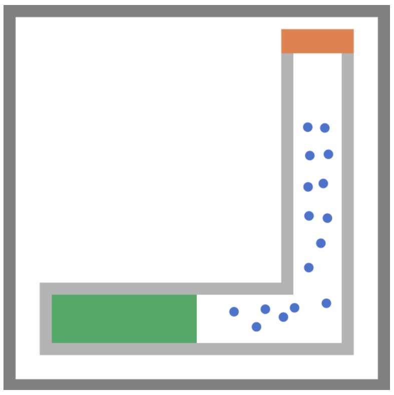
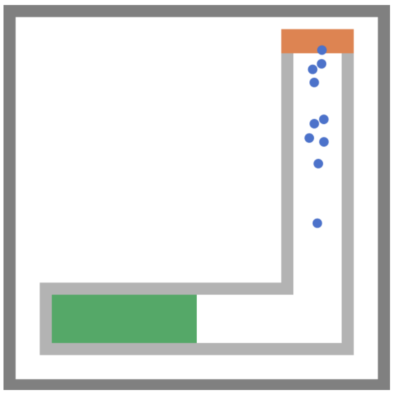

(c) GNM (Simulation time:34.4s)

Figure 8: RiMEA Test 1: Straight Line at t=20s

RiMEA Test 1: Straight Line The simulation time of Social Force Model and Gradient Navigation Model is the same, 34.4 seconds, which is slightly longer than the simulation time of Optimal Steps Model (29.614s). The pedestrian of OSM and SFM walks in the middle of the corridor, while the pedestrian of GNM walks along the wall.


(a) OSM (Simulation time:28.223s)


(b) SFM (Simulation time:20.4s)

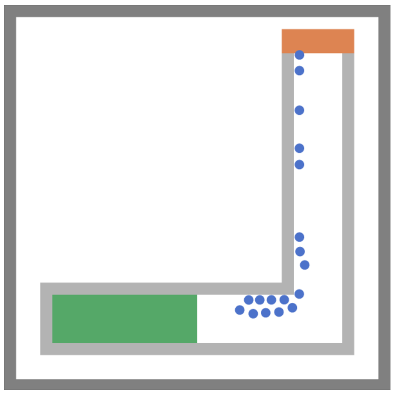

(c) GNM (Simulation time:33.2s)

Figure 9: RiMEA Test 6: Corner at t=14s

RiMEA Test 6: Corner The pedestrians of OSM and SFM follow their respective parallel trajectories, turning left and reaching the target, while the pedestrians of GNM approach the left wall at the beginning and walk along the left wall to reach the target. Therefore, the simulation time of GNM is the longest. In addition, the distance between pedestrians in OSM is larger than that in SFM, which also causes the simulation time of OSM to be a little longer than that of SFM.

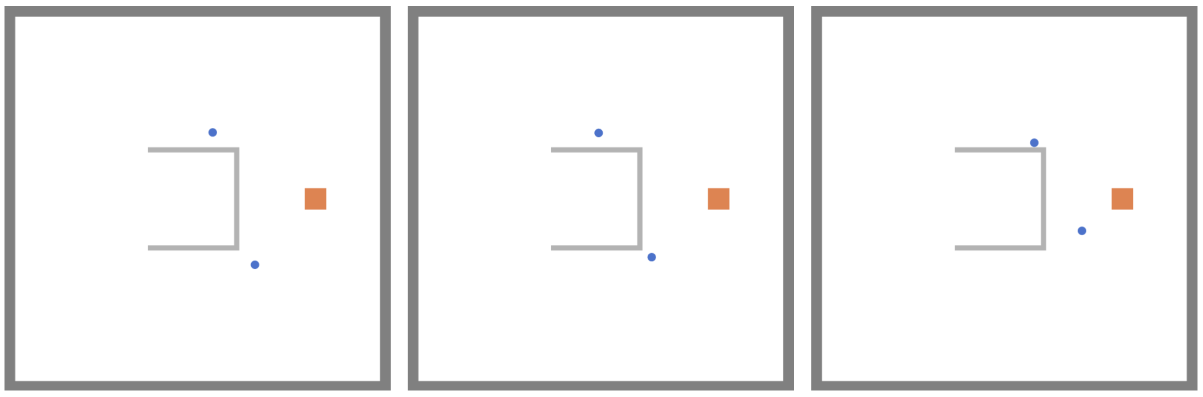

(a) OSM (Simulation time:13.766s) (b) SFM (Simulation time:14s) (c) GNM (Simulation time:12s)

Figure 10: Chicken test at t=8s

Chicken Test in Exercise 1 The simulation scenarios of OSM and SFM are basically similar, and the simulation time is also close. Pedestrians in GNM choose to walk along the wall when bypassing the U-shaped obstacle, so the simulation time is shortened.

Comparison and analysis of the three models It can be seen from the figures that the simulation scenarios of OSM and SFM are basically similar. Compared with SFM, OSM is relatively more capable of pedestrian avoidance, that is, the distance between pedestrians is larger. Pedestrians in GNM prefer to walk along the wall, which can shorten the simulation time to achieve the target.

In the Optimal Steps Model, the agent follows a greedy algorithm to approach the target, which is mathematically based on a utility function that maps each point in the plane to a utility value.

In the Social Force Model, it is recommended to describe the movement of pedestrians as being affected by "Social Force". These "forces" are not directly imposed by the pedestrian's personal environment, but a way to measure the internal motivation of individuals to perform certain actions. Therefore, SFM can very realistically describe the self-organization of the collective effects of several observed pedestrian behaviors.

Contrary to the four equations in the social force model, the gradient navigation model only needs three equations to get the ordinary differential equation system, which is fundamentally different from the force-based models. GNM uses the gradient superposition of the distance function to directly change the direction of the velocity vector. The velocity is then integrated to obtain the position. Likewise, pedestrians are no longer affected by inertia. There are several other advantages: because there is no actual force, the oscillation caused by the model is completely avoided. The derivative in the equation of motion is smooth, so a fast and accurate high-order numerical integrator can be used.

Report on task 3, Using the console interface from Vadere

Task description

In this task, we ran the Vadere through its console interface and compared the simulated results from the console with the ones from GUI. Other than that, modifying the scenario file, especially adding pedestrians as dynamic elements into the scenario through programs, is required.

Solutions

By specifying the path for scenario file and the output directory, it is not hard to use the console interface through command lines. To make it more convenient, we wrote python codes to generate the commands in format of “java -jar path/to/vadere/console scenario-run –scenario-file path/to/scenario/file –output-dir=path/to/output/directory”. The generated commands can be executed in python using the “os” library. To see more details about this part, please check task3.ipynb at the root of our GitHub repository.

To add pedestrians as dynamic elements into the scenario file through programs, we used the “json” library of python. Powered by this library, we managed to load the vadere scenario file from json format into python dictionary format. By simply converting the json codes from Vadere GUI for one pedestrian into python syntax as a dictionary, we successfully managed to add pedestrians into scenario file through programs. For this part of the task, we wrote a python class called “scenarioLoader” stored in ./pythonsrc/ folder of our GitHub repo. It enables us to load a vadere scenario file, add pedestrians, rename and output the modified scenario files easily. In the same notebook task3.ipynb, the procedure of adding an extra pedestrian to the RiMEA test 6 corner scenario using “scenarioLoader” we developed is shown in detains.

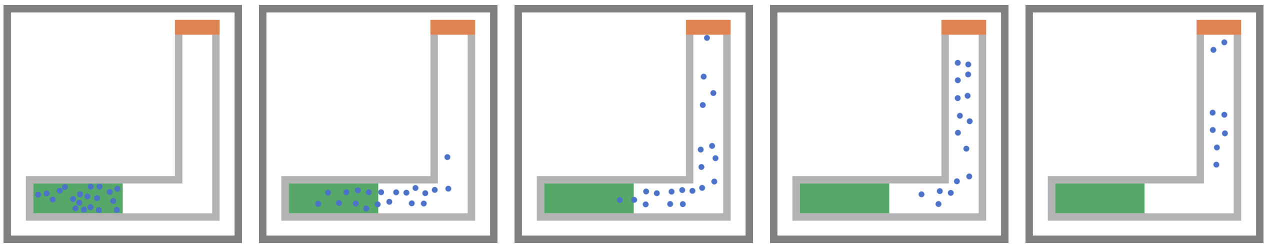

(a) t=0s (b) t=5s (c) t=10 (d) t=15s (e) t=20s

Figure 11: The corner scenario output by Vadere GUI.

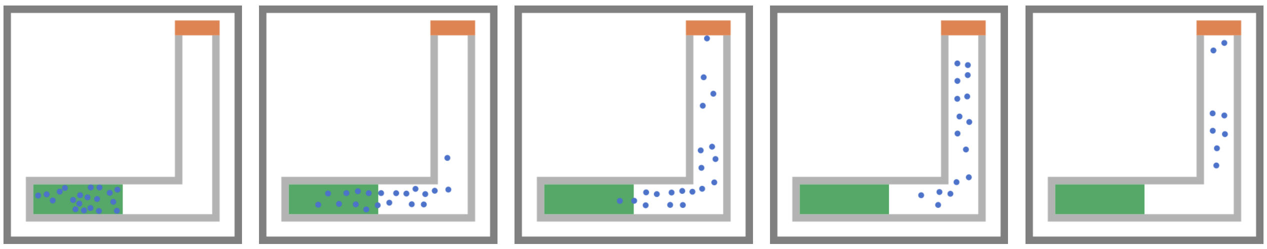

(a) t=0s (b) t=5s (c) t=10 (d) t=15s (e) t=20s

Figure 12: The corner scenario output by Vadere Console.

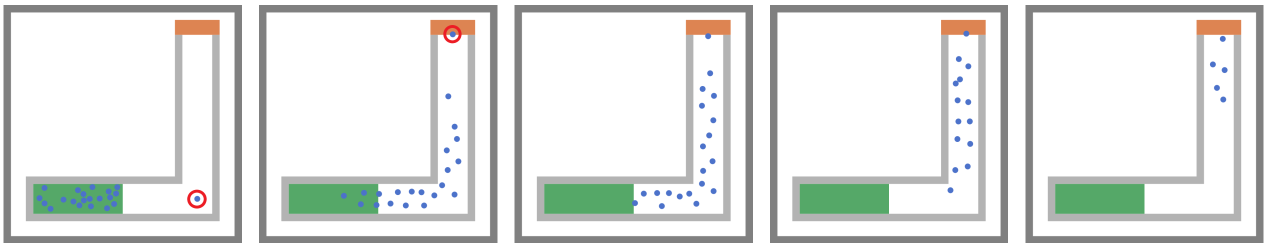

(a) t=0s (b) t=7.4s (c) t=10 (d) t=15s (e) t=20s

Figure 13: The corner scenario with an extra pedestrian output by Vadere Console.

Result

Firstly, by using the Vadere console interface, we re-simulated the corner scenario in task 1, and compared the output with the one from Vadere GUI. As Figure 12 and Figure 11 show, surprisingly, the results generated by the GUI and console are **exactly the same**. Note that the pedestrians are spawn at random positions in the source field in the scenario setting, but still the outputs are the same. Therefore, we concluded that Vadere has a special "random seed" that would be generated according to the scenario geometry.

Secondly, using the "scnearioLoader" we created, we successfully added an extra pedestrian at the right bottom in the corner scenario. Figure 13 shows the output of this new scenario. Comparing figure 13(a) and figure 12(a), we can see that after adding an extra pedestrian, the spawn positions of pedestrians in the source field changed. This confirms our conclusion that the "random seed" of Vadere depends on the scenario geometry. In this new scenario, the pedestrian we added reached the target at the first place (t=7.4s, figure 13(b)). Since the spawn positions are different, it is reasonable that the simulation results are different at other time points between figure 13 and figure 12.

Report on task 4, Integrating a new model

Task description

Now that you have used the Vadere software with its existing features, you have to integrate a new feature. This means you have to download the source code, add the new model, and then re-build the software using Java.

To re-build the software from its source code at the gitlab website, you need a proper IDE as well as the latest java development kit. Describe concisely and implement the following sub-tasks and tests.

1. Integrate the SIR model in Vadere.

2. To analyze the simulations, you have to get data about the "infective" pedestrians. You can use the modified output processor on Moodle to write the group information about the SIR model to a file.

3. Take a look at the method getGroupColor. You can define what color is assigned to what group by returning it depending on the current group of the given pedestrian.

4. Use the LinkedCellsGrid in org.vadere.util.geometry to improve efficiency of the distance computation for neighbors.

5. Run the following tests, and describe them together with their results:

Construct a scenario. Simulate the infection spreading through the static crowd, and visualize the results over time. Calculate how long it takes for half of the population to be infected.

Increase the infection Rate of the model using the GUI, and run the previous test again. Plot both results in one graph. Calculate how long it takes for half of the population to be infected again.

Construct a corridor scenario of 40m × 20m, where one group of 100 people moves from left to right, another one (also 100) moves right to left.

6. In the original implementation, the SIRGroupModel infection rate depends on the step size of the simulation. Suggest and implement a way to decouple the infection rate and the time step as far as possible.

7. Describe and motivate possible extensions of the model beyond what will be required in task 5. You do not need to implement these extensions, just describe them and argue why they would be reasonable.

Solution and Result

1 We used IntelliJ as the IDE to deal with Vadere source codes. After placing the SIR model files on Moodle to appropriate positions of the Vadere source codes, the SIR model is successfully integrated into Vadere. We wrote the method "initializeGroupsOfInitialPedestrians()" in SIRModel.java and modified the "update" method by using the LinkedCellGrid (see part 4 in this section). With these work been done, we managed to run the simulation of SIR model in Vadere GUI by adding SIR model as a submodel and specifying SIR attributes.

2 After integrating the SIR model, the simulation of a SIR model can be properly visualized during simulation. To also access to the SIR group information during the simulation, we integrated the FootStepGroupIDProcessor.java into Vadere. At the same time, to make the post-visualization of SIR simulation also works in Vadere GUI, we went through the codes of FootStepGroupIDProcessor and made it output the simulation time without rounded. Finally, it is important to link the group ID output processor to the original trajectory output file as well as creating a new output file called "groupID.txt". Linking the group ID processor to the trajectory output file makes the posi-visualization of SIR model in Vadere GUI work, and the "groupID.txt" file would be used by the Dash/Plotly application to visualize the infection progress (see part 5 of this section).

3 In this part, we modified the method "getGroupColor()" in the file "SimulationModel.java". By checking the group ID of the pedestrians, we managed to assign **Red** to infected group, **Green** to susceptible group, and **pink** to recovered group.

4 Instead of the pre-existing neighbor lookup that simply searched through all of the pedestrians in order to lookup neighboured ones for every single neighbor, we used the existing data structure LinkedCellsGrid in org.vadere.util.geometry. It has the advantage that a lookup for neighboured pedestrians doesn't happens in the order of all the pedestrians, i.e. O(n), but rather in constant time, comparable to the lookup in a predefined array. Although that in a static example the coordinates of the pedestrians doesn't change, we initialized the data structure inside the update method of the SIRGroupModel to make it also functions for a moving scenario. If we wanted to optimize only for the static scenario, we could have implemented it outside of the update method as a one-time initialization, but we made the decision to keep it more flexibly usable.

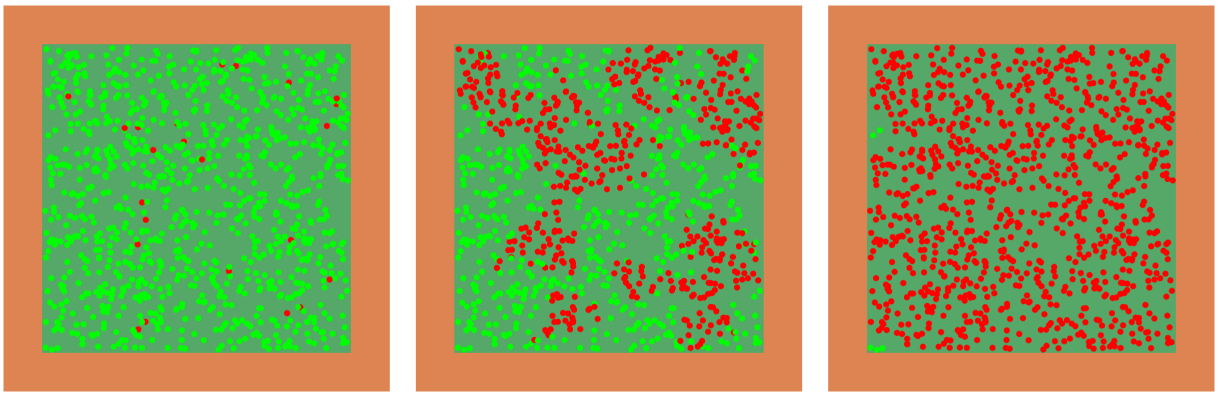

(a) t=0s. (b) t=86s. (c) t=500s.

Figure 14: The infection progress of a static group of pedestrians (10 initial infected people. Infection rate=0.01, infection distance=1m).

5.1 After we properly integrated the SIR model into Vadere, we managed to reproduce a static scenario of infection progress using SIR model. Figure 14 shows the simulation output of such an infection progress among a static population. It corresponds to "Static_infection_progress_without_recovery_rate1.scenario" in our repository's vadere project. Figure 14(a) to figure 14(c) are pictured from the post-visualization of Vadere GUI, which is an evidence that we managed to make the post-visualization work with SIR modelling simulation results (see part 2 of this section).

As shown in figure 14(a), we set 10 initial infected pedestrians at the beginning. Moreover, infection rate is set as 0.01 and infection distance is set as 1m. Figure 14(b) shows that half of the population are infected around time t=86s. Finally the simulation ends at t=500s (figure 14(c)), where some isolated small susceptible groups are observed (e.g. left bottom corner). Existence of such isolated susceptible groups is reasonable, since they are luckily spawn at the positions where no infected person are within their neighborhood.

Figure 16(a) is generated by the our modified Dash and Plotly application. It is stored at the folder "./pythonsrc/SIRvisualization/". It has been modified to plot the SIR information along time as a Stacked Area Chart. X-axis of the plot 16(a) represents the time in seconds, and the length of vertical segments in different colors at each given x represents the number of the corresponding group.

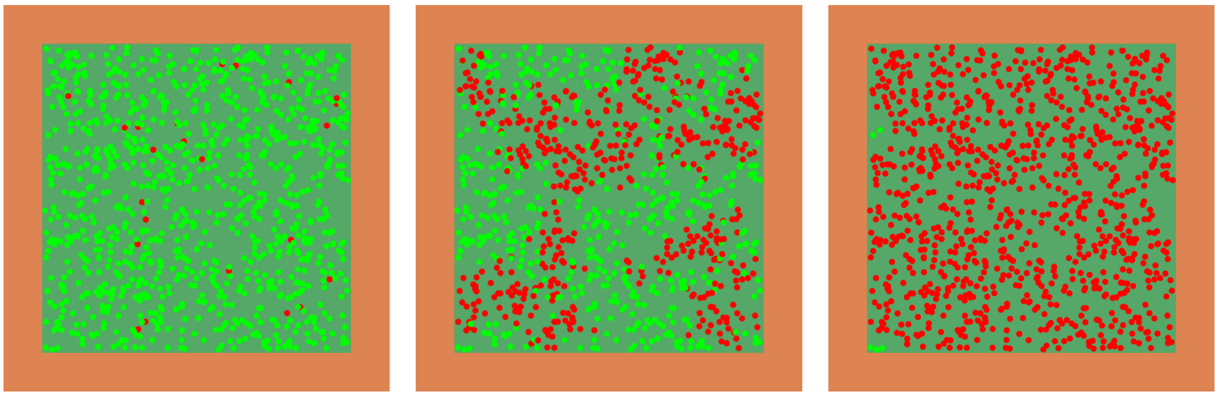
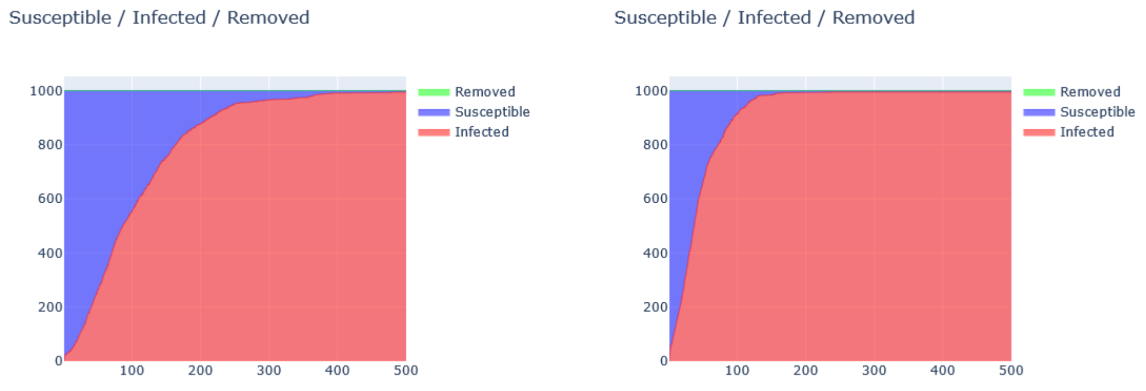

(a) t=0s. (b) t=37s. (c) t=500s.

Figure 15: The infection progress of a static group of pedestrians (10 initial infected people. Infection rate=0.02, infection distance=1m).

5.2 Based on the scenario we created for 5.1 in this section, we increased the infection rate from 0.01 to 0.02 and check the simulation results. This new scenario corresponds to "Static_infection_progress_without_recovery_rate2.scenario" in our repository's vadere project. Comparing figure 16(a) with figure 16(b), we can see that the infection spreads much faster. Half of the population are infected at around t=37s, which is approximately half of the time as the previous scenario.



(a) Infection rate=0.01 (b) Infection rate=0.02

Figure 16: The overall infection progress visualized by Stacked Area Chart.

5.3 For the counter flow situation, we created a corridor that has source field and target field at both ends. At each source field, 10 pedestrians will spawn per second and 10 seconds in total. Initial infected number is set to be 10. Moreover, we examined three settings and observed how many pedestrians are infected: 1. infection rate=0.01, infection distance=1m; 2. infection rate=0.02, infection distance=1m; 3. infection rate = 0.01, infection distance = 2m; 4. infection rate=0.02, infection distance=2m. Figure 17, 18 and 19 show the simulated results for these three setting respectively. Figure 21 summarises the infection spreading during the counter flow.

Given infection distance=1m, unlike the static cases, doubling the infection rate from 0.01 to 0.02 does not increase the infected number at all. It is reasonable as pedestrians are trying to avoid each other in the non-static cases. Therefore, if the avoidance behaviours overwhelm the infection distance, then the avoidance makes the increase of infection rate less effective under the circumstances that people are keeping distance .

We can further confirm that 1m infection distance is indeed overwhelmed by the avoidance. Comparing figure 18 and figure19, we could actually observe much more pedestrians get infected during the counter flow. Figure 21 shows that doubling the infection distance increases the final infected number from 10 to 30, which means infection distance=2m is now compatible with the avoidance behaviours.

Now if we further examine the effect of infection rate given infection distance=2m, we observe large increase of the infected number (comparing figure 21(c) and 21(d)) as we expected.

The above analysis shows that both of infection rate and infection distance are essential descriptors for the infection ability of a decease. Furthermore, the infection distance restricts the effect of infection rate when the infection distance is small.

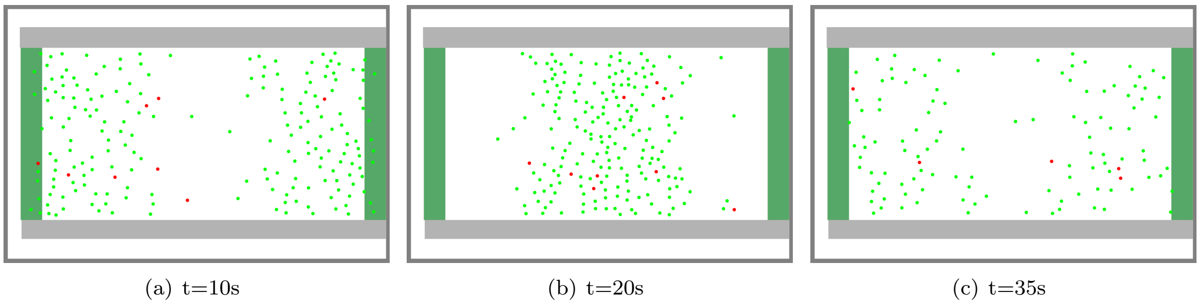

(a) t=10s (b) t=20s (c) t=35s

Figure 17: Counter flow scenario. Infection rate=0.01, infection distance=1m.

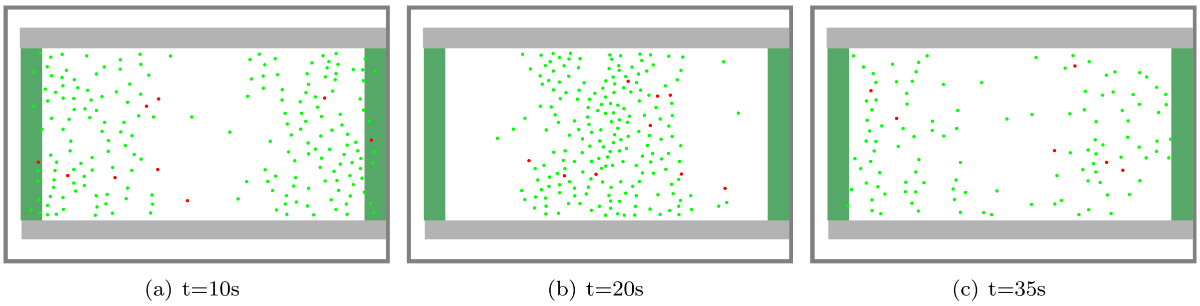

(a) t=10s (b) t=20s (c) t=35s

Figure 18: Counter flow scenario. Infection rate=0.02, infection distance=1m.

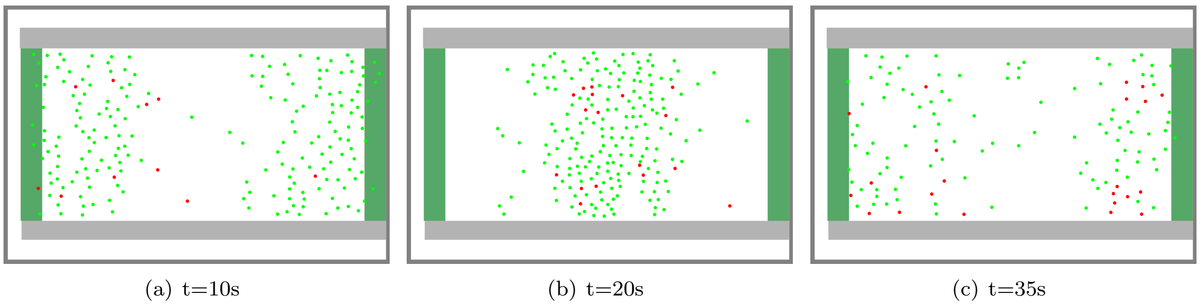
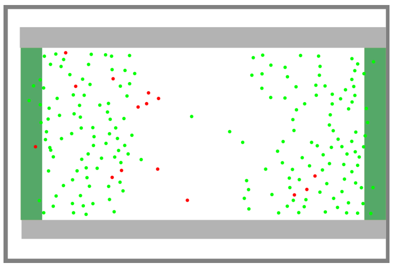
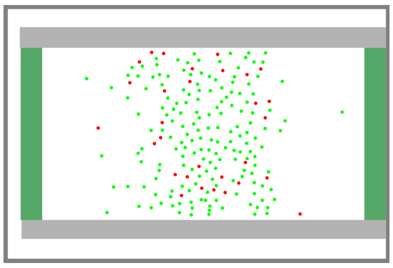

(a) t=10s (b) t=20s (c) t=35s

Figure 19: Counter flow scenario. Infection rate=0.01, infection distance=2m.


(a) t=10s


(b) t=20s

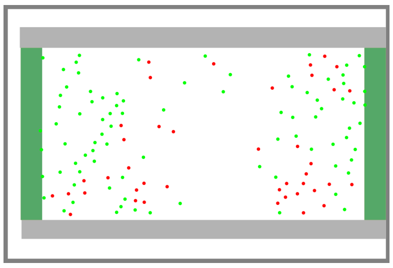
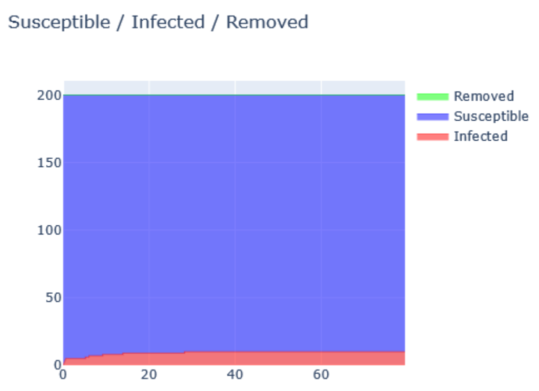
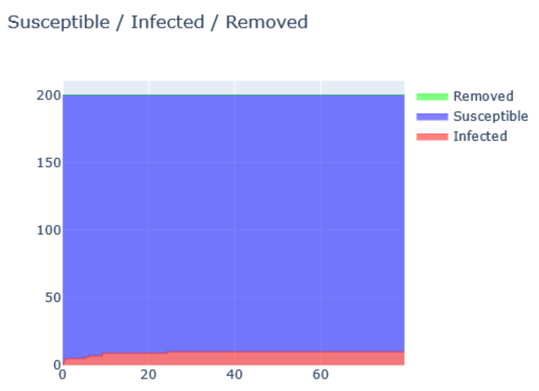

(c) t=35s

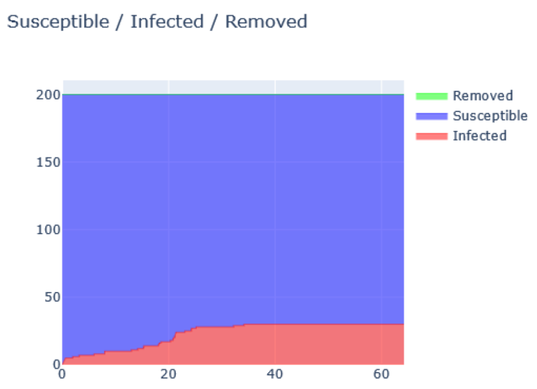
Figure 20: Counter flow scenario. Infection rate=0.02, infection distance=2m.



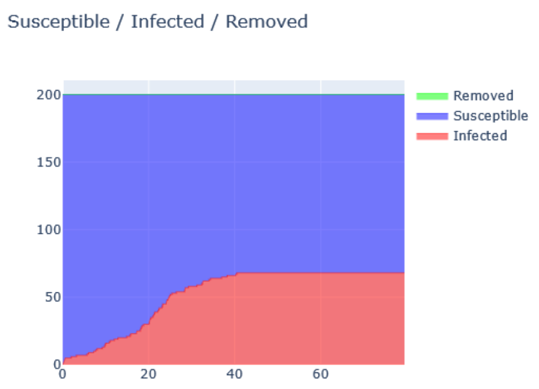
(a) rate=0.01, dist=1m, 10 infected



(b) rate=0.02, dist=1m, 10 infected



(c) rate=0.01, dist=2m, 30 infected



(d) rate=0.02, dist=2m, 68 infected

Figure 21: Counter flow scenario Stacked Area Chart.

6 Up to part 4 of this section, the setting of the infection in Vadere is: for each time step, each infected pedestrian can infect its neighbors with a given probability; each infected pedestrian can recover with a given probability (this is required by task 5). Let's say the default time step in Vadere is 0.4s. If the time step is changed to be smaller than 0.4s, then such infection spreading or recovery occurrence will be examined more frequently, which means the infection speed increases and recovery is accelerated in terms of real time. Therefore, changing the time step of simulation will influence the model development speed, which is not we expected. **Note**: The simulated results (figure 14) are under the update setting we introduced here (i.e. update per second).

To make the simulation more realistic, the infection spread and recovery occurrence should be examined for each unit time (e.g. per second) instead of for each time step in Vadere. Therefore, we modified the codes of "SIRGroupModel.java" and added a counter variable that counts the actual time. According to such counting, the "update" method will be executed per second instead of per simulation time step. Under such design, no matter how the users of Vadere change the simulation time step in Vadere, the development of decease using SIR model should be invariant for a defined scenario.

7 Brief ideas for extensions.

Mutation Infected people have small probability to mutate every per second and therefore re-infect the healed people with a small probability.

More realistic and not absolute healing Vaccine and already being infected once might not secure 100 percent but only 90 percent, so recovered people can have a small chance to get infected again.

Different types of susceptible people E.g. wearing masks, age, etc. and therefore we can assign different infection rates based on the type of people.

Report on task 5, Analysis and visualization of results

Task description

In the previous task, you had to integrate a simple SIR model into Vadere and test its core features. Now, you have to add features to the model, test them thoroughly and implement the following features:
1. Add a "recovered" state that represents recovered persons.
2. Add a probability for an "infective" person to become "recovered" at every time step.
3. If you are not using the Dash/Plotly visualization, implement your own version and describe it. If you are using the pre-implemented version, modify it so that the "recovered" state is correctly visualized.

Test at least the following things of the new model.
1. Construct a fairly large scenario with a source spawning 1000 pedestrians, and a target exactly on top of the source. Start with 10 infective and 990 susceptible. Visualize how the susceptible, infective, and recovered numbers change over time.
2. Experiment with the infection rate p and the recovery rate r of your model, illustrate how the SIR graphs change when these parameters are changed.
3. Now, decide on a fairly small infection and recovery rate. Then, construct an artificial "supermarket" scenario of at least 30 × 30m in Vadere, where people enter at one particular location, and wander around inside the scenario before they leave again after some time. Determine the impact of "social distancing" in the supermarket on reducing the number of infections

Solution and Result

1 Recovery State The recovered state was added into the "SIRType" class, and handled in the "SIRGroupModel" and coloured via the respecting function "getGroupColor()" in the file "SimulationModel" . Healthy Susceptible pedestrians are coloured in green, infected pedestrians are red and recovered pedestrians are visualised in pink, in respect to the group id, when colouring by groups in the simulation and in the output.

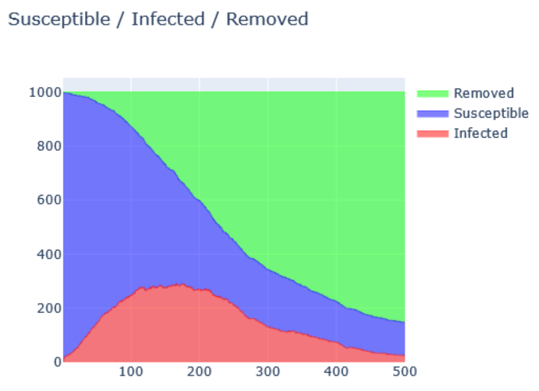
2 probability for recovery This was done very straight forward within the update function. Note that there always is only a probability that people are healed every time step so even if very unlikely in the real world, someone might be infected, but healthy very soon later and others stay ill a lot longer. This happens to a certain extend, but not that extreme, but obviously is fine for the granularity of the simulation.

Test: Visualize change over time when assigning different attributes The solutions of this task can be seen in the previous Task 4 where we used the Dash and Plotly application to visualize the SIR simulation results. Table 2 shows our different settings in this test.

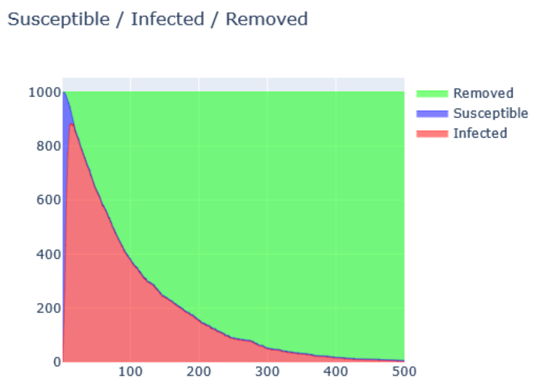
Test number	Infections at start	Infection rate	Recovery rate	Max. infection distance
Test 1	10	0.01	0.01	1
Test 2	10	0.01	0.01	3
Test 3	10	0.01	0.05	1
Test 4	10	0.01	0.05	3
Test 5	10	0.05	0.01	1
Test 6	10	0.05	0.01	3
Test 7	10	0.05	0.05	1
Test 8	10	0.05	0.05	3

Table 2: Distribution of the values in our variables for the different tests

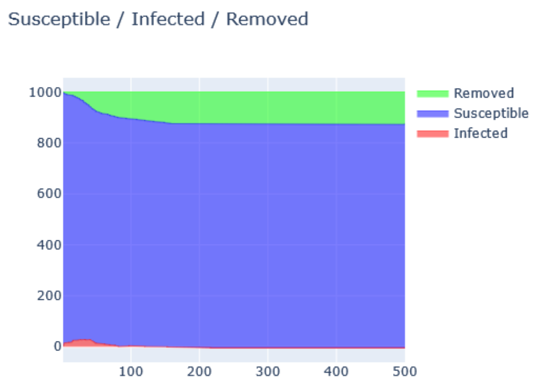
This is how we distributed the different values of the our test runs. as can be seen in the Dash/Plotly implementation some of this combinations lead to a infection of all the people in the scenario (e.g. Test 2,4,6,8). In other cases, few weakly connected people, that have "socially distanced" aren't infected at all (e.g. Test 4), and in some cases the Infection that started with 10 infected persons, briefly continued to thrive in a small outbreak before the recovery eventually dominated, since it also limits possibilities to spread (e.g. Test 3). Most infection scenarios have a exponential growth and then are limited by the total number of people in the simulations.



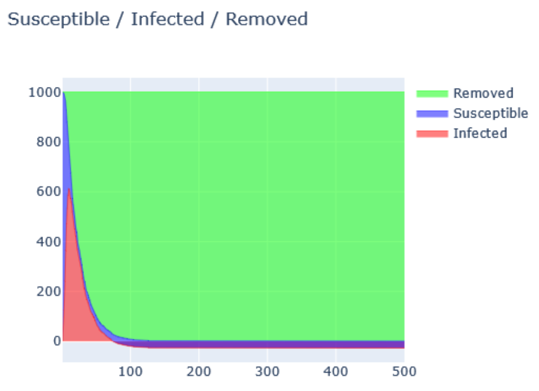
(a) Test 1



(b) Test 2

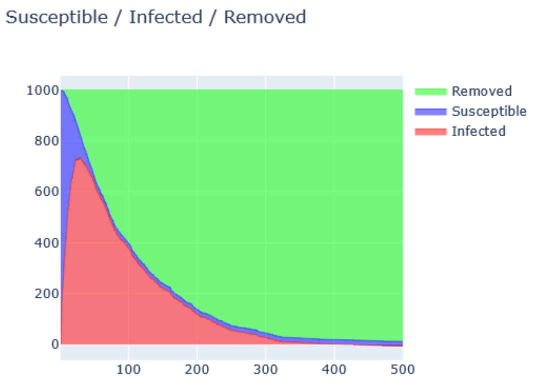


(c) Test 3

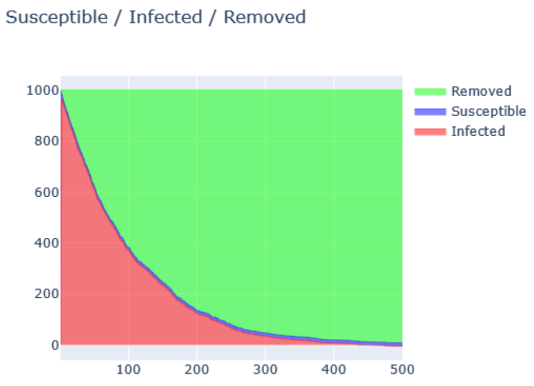


(d) Test 4

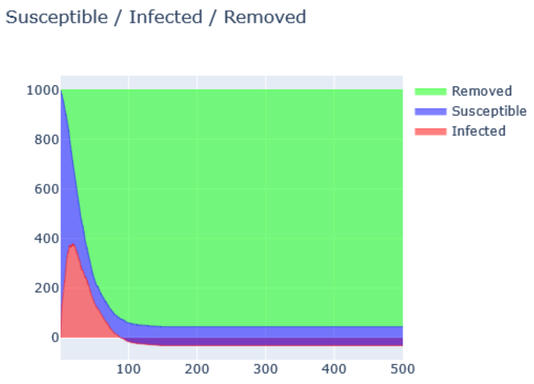
Figure 22: Test 1 to 4 on the infection progress of SIR model.



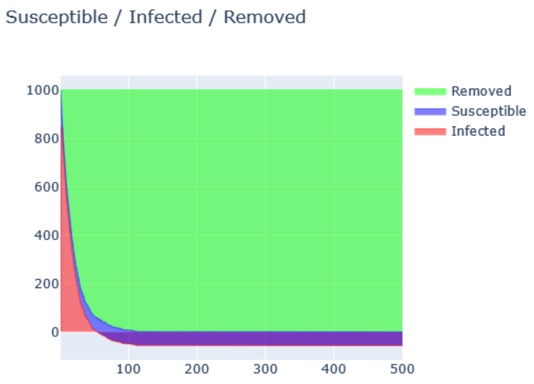
(a) Test 5



(b) Test 6

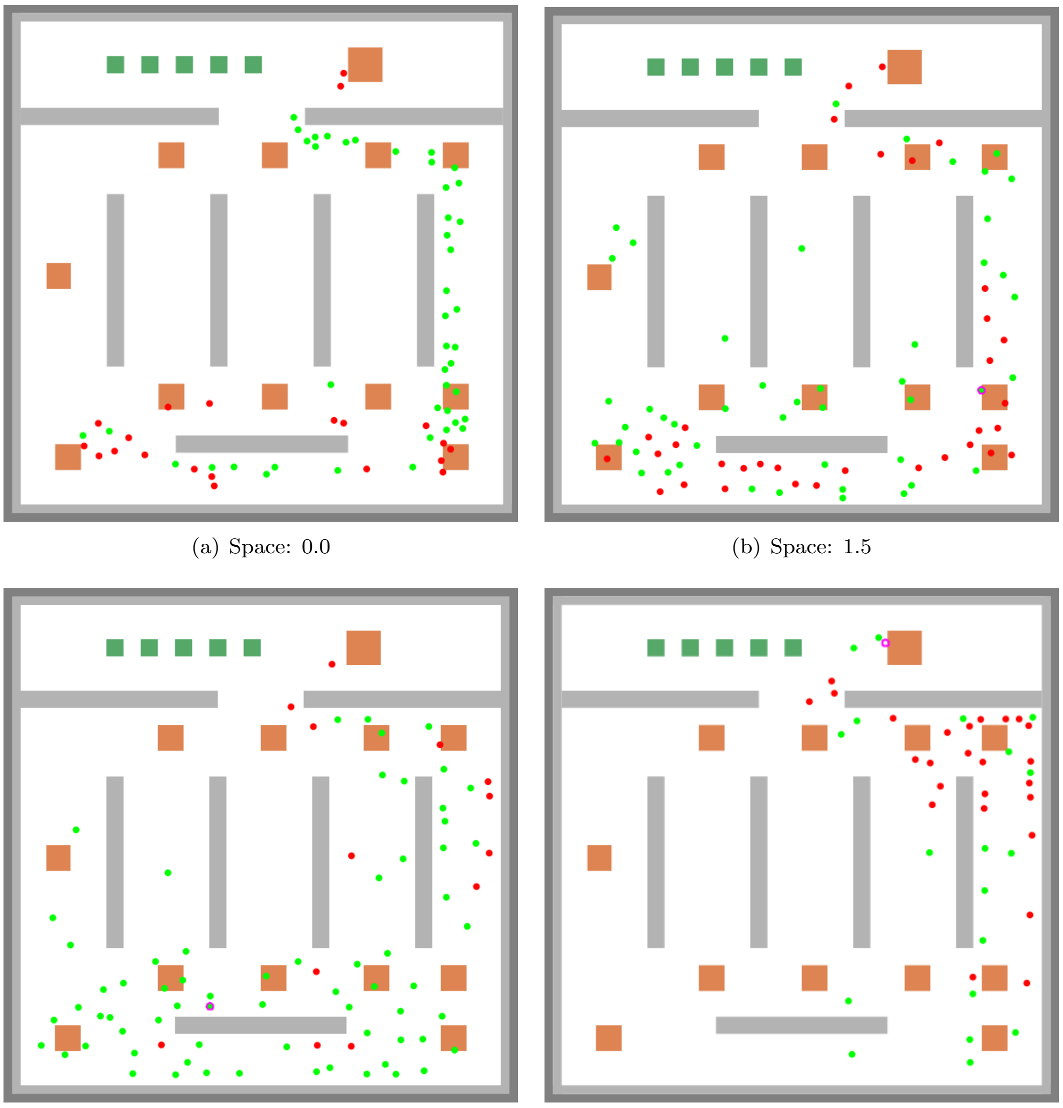


(c) Test 7



(d) Test 8

Figure 23: Test 5 to 8 on the infection progress of SIR model.


(a) Space: 0.0

(b) Space: 1.5

(c) Space: 3.0

(d) Space: 5.0

Figure 24: 4 different interesting timestamps for “pedPotentialPersonalSpaceWidth” in the supermarket scenario.

3. Test: Supermarket scenario The test can be found in the project folder :“vadere-project-supermarket”. They were calculated with the Optimal Steps Model from Vadere, only the “pedPotentialPersonalSpaceWidth” has been adapted for the different simulations.

About Test 1 (Figure 24(a)) Surprisingly, little infection occurs in the scenario where the parameter “pedPotentialPersonalSpaceWidth” is set to 0. One explanation might be that there is a random cluster of non-infected pedestrians that is fairly far apart from the infected pedestrians. A second reason may be that when disregarding personal space, pedestrians move much faster in and out of the supermarket. This means the total time exposed is fairly shorter. And finally, if the supermarket is very crowded , then there won’t have much space to avoid others.

About Test 2 (Figure 24(b)) In this test, the “pedPotentialPersonalSpaceWidth” is set to 1.5, and the Infection Distance is set to 2 (comparable to Covid19) which is enough to separate Pedestrians a little. But this might lead to “bridges” built e.g pedestrians form a constantly connected network in which they might even stay exposed to each other for longer. The image shows an early stage of this scenario. Unlike in real life, 1.5m distance and no distance at all result in the same risk for infection.

About Test 3 (Figure 24(c)) This is by far the best outcome with the least infections. Pedestrians preferably keep 3 meters distance, which is larger than the required 2 meters to be safe from an infection. They are evenly spread out, use more parts of the supermarket and overall are more evenly distributed than in the other scenarios.

About Test 4 (Figure 24(d)) The screenshot shows the most interesting part in the simulation for "pedPotentialPersonalSpaceWidth" being set to 5. For the most of the parts in this scenario, it behaves as ideal as figure c. However, when approaching to the end of the simulation, pedestrians get stuck in the top right corner and wait. This is when many of them get infected and further infect other pedestrians that pass by. Soon, pedestrians at that corner are all infected. A real disaster in which most pedestrians behave very ideally, but very few behave questionably, form a bottleneck and therefore infect many.