Report for exercise 2 from group C

Tasks addressed:	5
Authors:	Alejandro Hernandez Artiles (03785345) Pavel Sindelar (03785154) Haoxiang Yang (03767758) Jianfeng Yue (03765255) Leonhard Chen (03711258)
Last compiled:	2024–03–02
Source code:	https://github.com/alejandrohdez00/Exercises-MLCMS-Group-C/tree/main/Exercise-2

The work on tasks was divided in the following way:

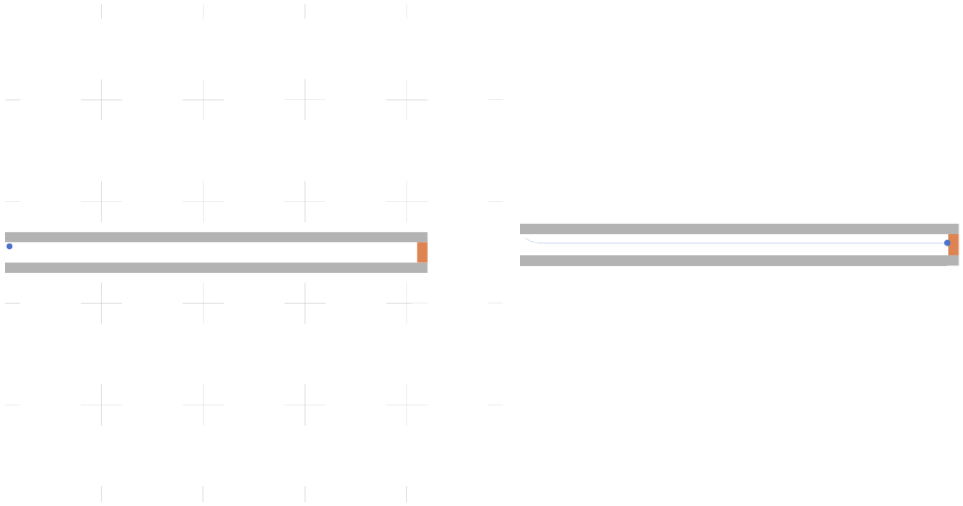
Alejandro Hernandez Artiles (03785345)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Pavel Sindelar (03785154)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Haoxiang Yang (03767758)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Jianfeng Yue (03765255)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Leonhard Chen (03711258) **(Project Lead)**	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

Report on task 1, Setting up the Vadere environment

In this task, we will explore the Vadere software, a valuable tool for simulating pedestrian movement. The objective is to use its graphical interface to recreate specific scenarios: RiMEA scenarios 1 (straight line) and RiMEA scenarios 6 (corner), along with the "chicken test" from the first exercise. Employing the Optimal Steps Model (OSM) with its standard template, our aim is to observe and analyze model trajectories, visualizations, user interface interactions, and test results. Additionally, we will compare these outcomes with the cellular automaton developed in the first exercise to identify any similarities and differences. This exploration allows us to better understand Vadere's capabilities in simulating pedestrian movement. *Note that for task 1,2,3 the vadere project with its scenarios are placed in the folder* `Exercise-2\vadere-project-task123`

RiMEA scenario 1 In this scenario, the goal is to demonstrate that an individual in a 2 m wide and 40 m long corridor, with a predefined walking speed, will cover the distance within the corresponding time period.

To simulate this scenario in Vadere, as can be seen in Figure 1(a), the `Obstacle` tool was utilized to create two gray rectangles of size 1×42 in the center of a 50×50 map, representing the corridor. The rectangles are spaced 2 units apart. Additionally, the `Target` tool was employed to create an 2×1 orange rectangle on the right side to represent the exit. On the left side, the `Pedestrian` tool was used to introduce an individual with a walking speed of 1.33 m/s, represented by a blue color.



(a) RiMEA 1 setup

(b) RiMEA 1 finshed

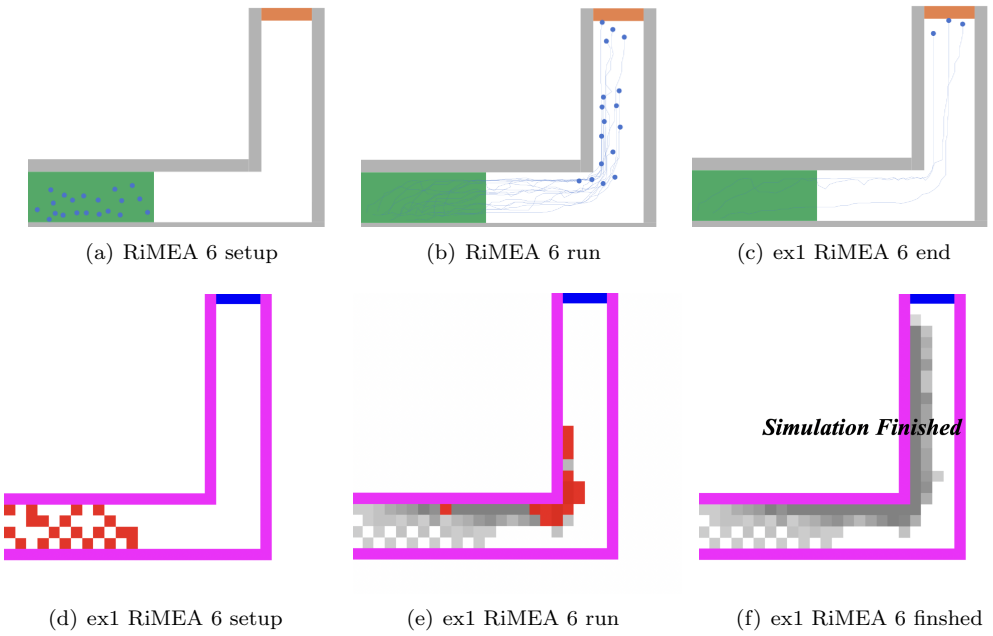
(c) ex1 RiMEA 1 setup

(d) ex1 RiMEA 1 finshed

Figure 1: Comparison of RiMEA 1 simulations

The Figure 1 illustrates a comparison between RiMEA Scenario 1 in exercise 1 and this simulation. Similarities can be observed, as pedestrians take approximately the same amount of time to traverse the entire path. However, a notable distinction lies in the paths traversed by pedestrians in each experiment. In the simulation in exercise 1, the pedestrian moves in a straight line throughout the journey. However, in the this simulation with the OSM, the pedestrian moves toward the midpoint of the corridor during the forward movement, and then walks in a straight line to complete the route.

RiMEA scenario 6 The purpose of this scenario is to demonstrate that a group of twenty individuals, moving towards a corner that turns left, will successfully navigate around it without passing through walls.



(a) RiMEA 6 setup (b) RiMEA 6 run (c) ex1 RiMEA 6 end

(d) ex1 RiMEA 6 setup (e) ex1 RiMEA 6 run (f) ex1 RiMEA 6 finshed

Figure 2: Comparison of RiMEA 6 simulations

As shown in the Figure2(a), the simulation setup utilized Vadere. The `Obstacle` tool was employed to create a corridor with a width of 4 units and a turning corner a 30x30 map. On the right side, the `Target` tool was used to establish a 4x1 rectangle, representing the exit. On the left side, the `Source` tool was employed to create a green area that generates 20 pedestrians. These pedestrians move at a speed of 0.5 m/s.

In Figure 2, a comparison can be made between two simulations. In both experiments, pedestrians successfully navigate around the corner without passing through the walls. In the Vadere simulation using the Optimal Steps Model (OSM), pedestrians exhibit mutual repulsion, and not all of them cluster tightly while passing the corner. Instead, some pedestrians intentionally choose to walk on the outer side of the corner, maintaining a sufficient distance. However, in the simulation in exercise 1, pedestrians tend to take the shortest path, brushing the walls as they pass through the corner.

RiMEA Chicken test The purpose of the "Chicken Test" scenario is to assess obstacle avoidance. In the absence of obstacle avoidance, pedestrians would become trapped and unable to reach the target.

we used the `Obstacle` tool to create a fence with a width of 1 in a 50x50 map. Then, the `Target` tool was employed to establish a 1x1 exit directly below the obstacle. One Pedestrian are placed within the fenced area.

(a) chicken test (b) ex1 chicken test

Figure 3: Comparison of chicken test

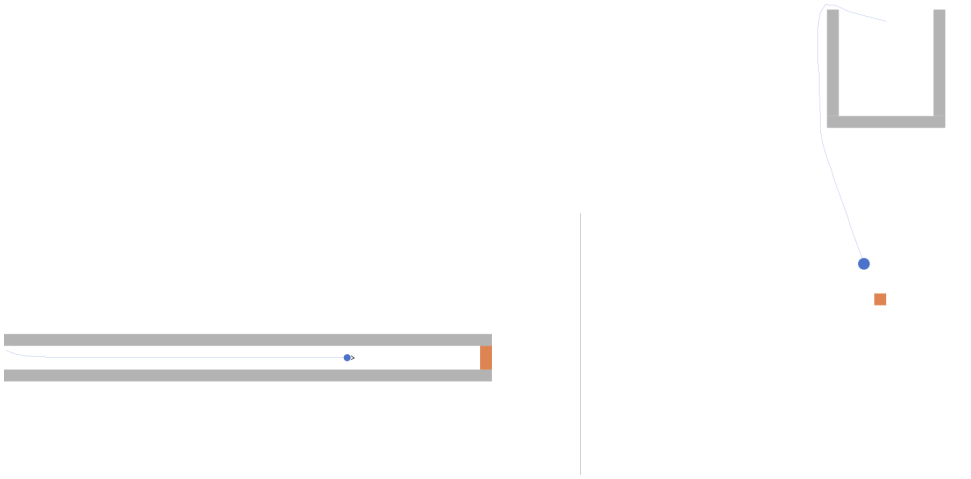
As shown in the Figure 3, although both simulations successfully navigate around the fence, their trajectories exhibit distinct differences. The most significant distinction is that, unlike in Exercise 1 where pedestrians tend to first move diagonally downward before transitioning to a straight downward movement, in the Vadere simulation, pedestrian, after circumventing the fence, proceed directly towards the exit.

Software Comparison In terms of visualization, Vadere uses different colors to represent elements. It offers an array of additional features for showcasing the movement process, including options to show/hide the walking direction of all pedestrians, show/hide the grid, and show/hide trajectories, etc. However, Vadere lacks the capability, present in our own cellular automaton from the first exercise, to retain the path lines of removed pedestrians on the map. As far as the user interface is concerned, Vadere excels in scene creation compared to our own cellular automaton. It not only enables scene creation through mouse interaction but also allows users to input code for scene elements directly within the interface window.

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

Once we have tested Vadere's default model, Optimal Steps Model, we can try to experiment with other models, like the Social Force Model.

After testing this new model in RiMEA 1 and the chicken test, we can observe there is almost no changes in the pedestrian behaviour, the pedestrians travel to the goal as before. Figure 4a and Figure 1a seem to be exactly equal. In the chicken test (Figure 4b), the only difference, barely perceptible, is the path each model takes in the first steps out of the obstacle structure. The OSM model faces the obstacles until it dodges them, while the SFM feels the disturbance of the obstacles from the start and takes a slightly more diagonal initial path. A priori, this may indicate that a difference between the two models lie in the repulsive potential function of the obstacles.

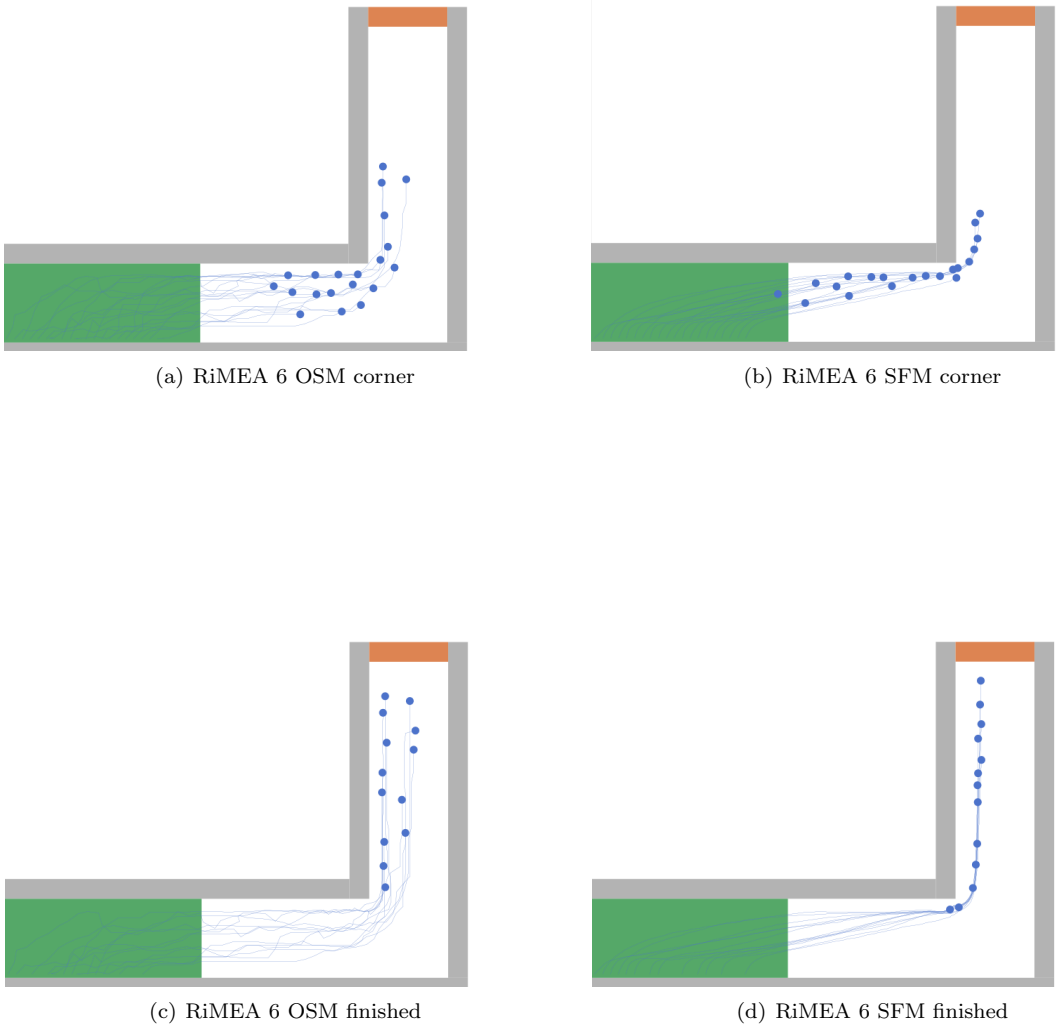


(a) RiMEA 1 (b) Chickent test

Figure 4: RiMEA 1 and chicken test with SFM

By definition, RiMEA 6 requires obstacle avoidance as well as pedestrian avoidance in both models. Therefore, now we can observe more differences than before testing on this scenario.

As we hypothesised with the chicken test, it now seems clear that both models have different functions for modelling pedestrian repulsion to obstacles and to other pedestrians. Looking at the critical point which is the corner of the scenario, we can see how the pedestrians are positioned in the two models. In OSM they have more distance between them and there is less density of pedestrians in the corner, being able to move away from the wall. However, in SFM the pedestrians stick close together, going almost completely in a line during the final corridor. That is, in SFM every pedestrian seems to follow exactly the same path as we can observe in Figure 5d.



(a) RiMEA 6 OSM corner

(b) RiMEA 6 SFM corner

(c) RiMEA 6 OSM finished

(d) RiMEA 6 SFM finished

Figure 5: Comparison between SFM and OSM in RiMEA 6

As it is said in Ref [3], the SFM's pedestrian movement is mainly computed taking into account the motivation of a pedestrian to accelerate or decelerate, depending on the external forces that surround it. These external forces are mainly: The shortest way to reach the goal given a desired speed, the pedestrians near itself, the obstacles near it and the possibility that a pedestrian is attracted by other persons (friends, celebrity, etc.). Also, it is important to note that this model gives weaker influence to situations located behind a pedestrian, coherent when observing the organization of pedestrians in Figure 5d where they follow each other. SFM uses differential equations to model all these interactions.

On the other hand, the OSM model's pedestrian use discrete movement in a continuous space, taking into account a floor field where pedestrians and obstacles have repulsive potential [4].

Therefore, the main differences between the behaviour of the two models, apart from their own way of defining the function that models the repulsion of obstacles and other pedestrians, is that in the case of OSM, the pedestrians have a discrete motion, which makes them move more like real people and not like particles. This can be realized in Figure 5a, where the OSM model got a more realistic approach of how a crowd would distribute itself around a corner.

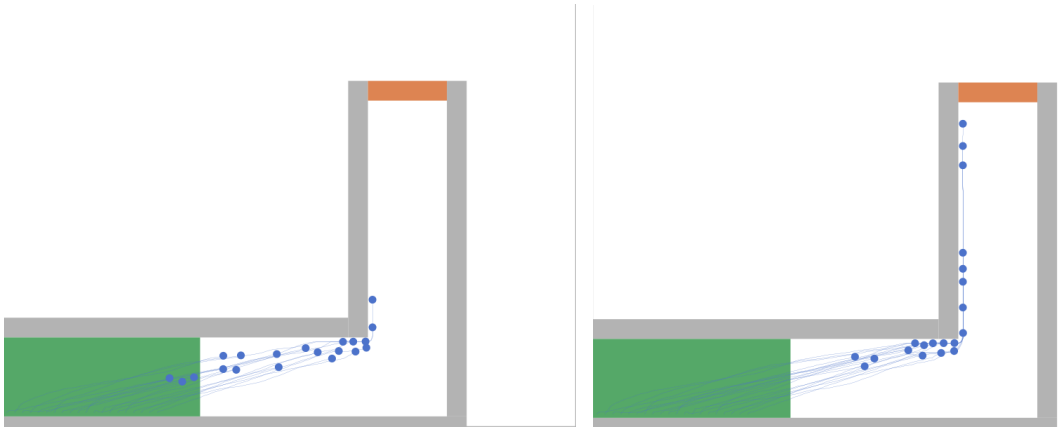
In addition, OSM's pedestrians, as stated in [4], can decide whether to move closer to their desired speed, slow down or maintain a group structure, whereas SFM's pedestrians move accelerated by external forces, one of them being arriving at the goal using the shortest path. This difference can be noted in Figure 5c and Figure 5d.

After testing and comparing OSM and SFM models, we tried the Gradient Navigation Model. Without reading its related publication, we can already observe that this model tries to take the most optimal way to the goal, being very tolerant of being close to obstacles. We can observe in Figure 6a the pedestrian does not initially position itself in the centre, like the previous models did, in order to be separated from obstacles, but takes the fastest way, going straightforward directly. Also, we can see how in Figure 6b the pedestrian takes the path closest to the obstacles, being this the shortest one.

(a) RiMEA 1 (b) Chickent test

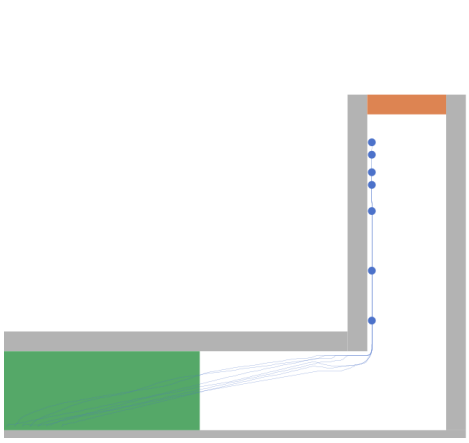
Figure 6: RiMEA 1 and chicken test with GNM

In RiMEA 6, we can observe how this behavioural pattern remains. Pedestrians wait their turn to turn the corner, keeping as close to the wall as possible to ensure the most optimal path to the goal.



(a) RiMEA 6 start

(b) RiMEA 6 middle



(c) RiMEA 6 finish

Figure 7: RiMEA 6 test with GNM

GNM is based on certain assumptions about people's behaviour, like: "Crowd behavior in normal situations is governed by the decisions of the individuals rather than by physical, interpersonal contact forces" that contrast with the functioning of SFM, where pedestrian acts as if it would be subject to external forces [2]. However, if we analyse the publication further, we can better understand the nature of the different functioning of this model. It states that pedestrians want to reach the target in as little time as possible, as SFM also does, and that pedestrians are able to change their desired velocity as a reaction to the presence of pedestrians and obstacles, like OSM.

Nevertheless, we consider that what causes this distinct behaviour is the navigation function or field, since it is in the paths that the pedestrians of this model follow to the goal that differentiates them most from the other two models. The navigation uses the solution σ, that represents the first arrival times or walking costs in the specified domain, and $G(x)$, the speed function. [2] states that if $G(x) = 1$ for all x, the solution σ represents the length of the shortest path to the closest target region. However, to account for the fact that pedestrians cannot get arbitrarily close to obstacles, the wave is slowed down in the immediate vicinity of walls by reducing the speed function G.

Hence, we believe that GNM excels in path optimization due to its utilization of a constant value for G in this case. As a result, the solution σ yields the most efficient path. Also, running simulations in RiMEA 6 reveals that pedestrians situated further from the wall move at a higher speed until they gradually adjust their course to adhere to the optimal path, showcasing the reduction of G in the presence of nearby obstacles. Despite the decrease in pedestrian speed, it seems that approaching the wall allows pedestrians to cover less distance in these test scenarios, establishing the proximity to the wall as the optimal path identified by σ.

Moreover, observing the results of our cellular automaton in Exercise 1, we can observe that they look very similar to GNM's behaviour. As commented in Task 1, our cellular automaton goes straightforward in RiMEA

1, being GNM the only model that does that too. Looking at the chicken test, our cellular automaton and GNM are the only ones that adhere to the wall to ensure the optimal path. Then, comparing RiMEA 6's results, we can realize that the behaviour is very similar, where pedestrians follow the shortest path waiting for the other pedestrians to turn the corner.

Our cellular automaton was designed to search for the optimal path using fast marching method. Therefore, this similarity gives us a new clue as to how GNM works. As it is reported in [2], GNM also uses fast marching to solve σ [5]. Thus, we think that is why both models do the same routes and ensure the optimal path.

The .scenario files for this tasks can be found in the scenarios folder and they are named according to the algorithm in use:

```
<scenario_name>_SFM.scenario
<scenario_name>_GNM.scenario
<scenario_name>.scenario #(default one, using OSM)
```

Report on task 3, Using the console interface from Vadere

This task aims to explore the utilization of the Vadere simulation software's console interface, allowing for the modification of scenario files and the addition of pedestrians through programming. Leveraging the console version enables Vadere's use as a "black box" within alternative software environments, such as Python.

The primary objective of this task is to programmatically add pedestrian to scenario files and compare the results between the console version and the graphical user interface. The "corner scenario" will be employed as the initial scenario, and the code will be developed to add pedestrians to specific locations. By comparing runtimes, we aim to discuss the similarities and differences in reaching the target between the added pedestrians and those originating from the source area.

add_pedestrian.py To modify the Vadere simulation scenario file programmatically in order to add pedestrian, we created a Python script (`Exercises-MLCMS-Group-C/Exercise-2/add_pedastrian/add_pedestrian.py`). Here's a concise overview of the implementation details:

- **Predefined pedestrian template**
  - We formulated a JSON template, representing the attributes of the pedestrian intended for addition to the scenario.
- **Adding Pedestrian function**
  - The heart of the implementation lies in a function:
    - It reads the existing scenario file.
    - Picked a unique ID for the newly introduced pedestrian.
    - Updates the predefined pedestrian template with the ID and user-provided inputs.
    - Appends the modified pedestrian template to the 'dynamicElements' list within the scenario file.
- **Setting Pedestrian Parameters**
  - During script execution through the Command Line, users specify parameters such as the scenario path to be modified, the initial position of the pedestrian to be added, their speed, and the target ID.

Workflow The following is the workflow for the task. For the sake of convenience, these commands has been recorded in `Exercises-MLCMS-Group-C/Exercise-2/add_pedastrian/task3.ipynb`

1. add pedestrian to RiMEA scenario 6

```
python add_pedestrian.py
--scenario "..\\vadere-project-task\\scenarios\\RiMEA scenario 6.scenario"
--x 22 --y 3 --targetID 3 --speed 0.5
```

2. call vadere-console.jar with the newly modified scenario file.

```
$java -jar vadere-console.jar scenario-run
--scenario-file "..\\vadere-project-task\\scenarios\\RiMEA scenario 6_modified.scenario"
--output-dir "..\\scenarios\\output"
```

Command vs GUI After executing the command, as compared to the GUI, the Command prints out log records detailing the initialization of the simulation scenario, analysis of the JSON tree, utilization of the configuration file, solving of the floor field, initialization of pedestrians, and the various stages of the simulation run. Additionally, the logs cover the generation of output files, including trajectory data, overlap information, and other relevant details.

After completing the simulation run, four output files were generated in both GUI and Command Line executions. These files include:

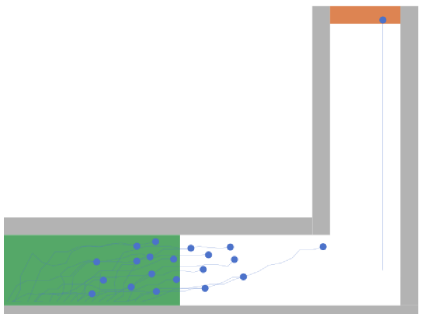
- `overlapCount.txt`
- `overlaps.csv`
- `postvis.traj`
- `RiMEA scenario 6_modified.scenario`

A thorough comparison of the content within these files reveals that the files generated through both interfaces are found to be identical in every aspect. It signifies the reliability and coherence of the Vadere. This consistency is valuable for users who may opt for either interface based on their preferences or specific workflow requirements, assuring them that the simulation results will remain consistent irrespective of the chosen mode of execution.

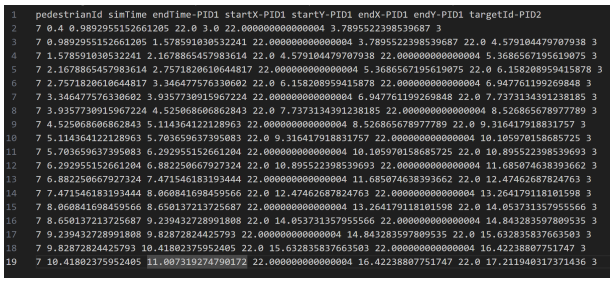
Comparing Pedestrian Travel Times In the graphical user interface scenario, Figure 8(a) illustrates that the pedestrian we added is the fastest to reach the destination.

Upon utilizing the `Command` to add a pedestrian, the system outputs: "Finished adding pedestrian. ID of the new pedestrian is 7." Consequently, we can use ID 7 to precisely locate the trajectory information of this newly added pedestrian in the `./vadere-project-task123/output/RiMEA scenario 6_modified/postvis.traj` file. To isolate the trajectory details for the newly added pedestrian, we apply the regular expression `^(?!7).*\n`, replacing all matching entries with empty strings. After restoring the first line of the original text, the resulting content, as shown in the figure 8(b), encapsulates the entire trajectory information for the new pedestrian. Notably, this pedestrian completes its journey in 11 simulation unit time.

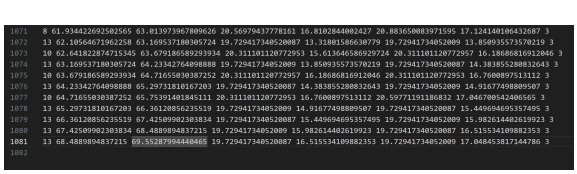
In contrast, pedestrians generated in the source area exhibit a broader range of travel times. The slowest pedestrian from the source area takes 69.55 simulation unit time to reach the destination, as indicated by the last line of the `postvis.traj` file 8(c). The fastest pedestrian from the source area completes the journey in 25.28 simulation unit time 8(d). To extract this information, we employ regular expression repeatedly to remove entries containing IDs, those appearing in the last line. Ultimately, the remaining two entries consist of the newly added pedestrian and the one who from source area reached the fastest.



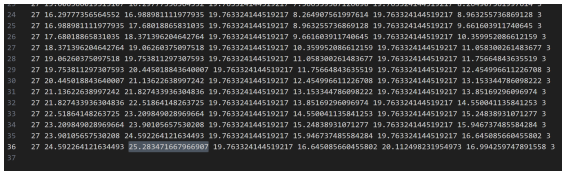
(a) The Added pedestrian reaches exit in GUI



(b) The Added pedestrian, ID 7



(c) The slowest source pedestrian, ID 13



(d) The fastest source pedestrian, ID 27

Figure 8: Pedestrian Travel Time

Report on task 4, Integrating a new model

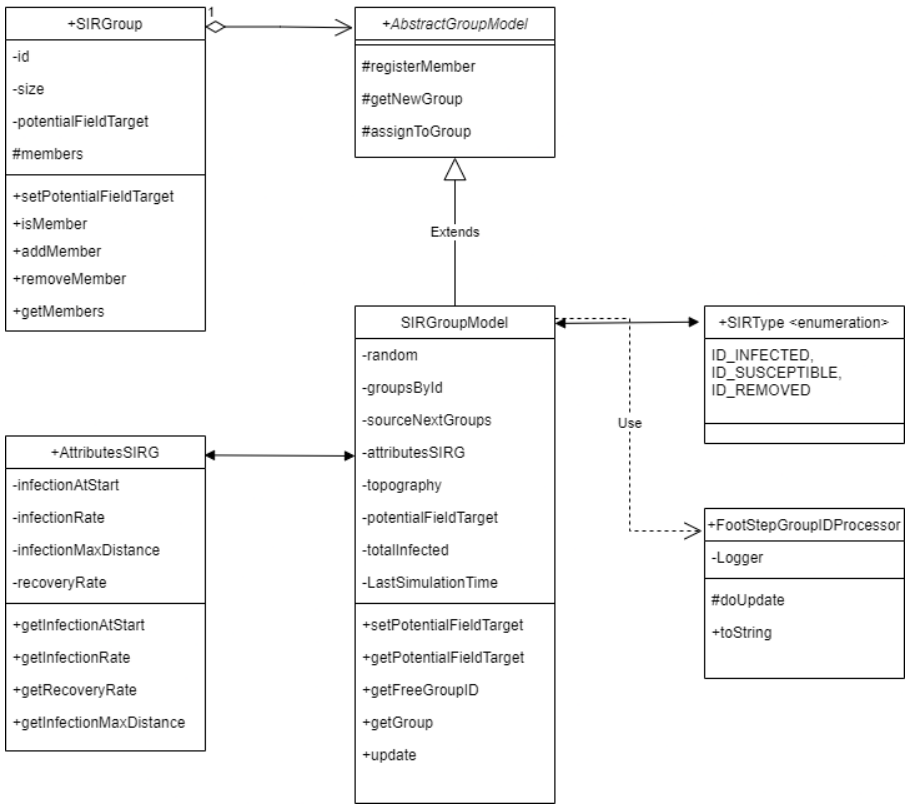


Figure 9: UML diagram of SIR Model

1. Implementation of the SIR Model into Vadere The UML diagram above 9 describes the relationships between the classes `SIRGroupModel`, `SIRGroup`, `AbstractGroupModel`, `AttributeSIRG`, `SIRType` and `FootStepGroupIDProcessor`. For some classes only important functions were listed.

`SIRGroupModel` is the main class for handling the behaviour of groups in the SIR model. It keeps track of the group ids and assigns pedestrian to each of the groups in the SIR model. The update function does most of this work. It implements calculation for infecting and recovering pedestrians and also contains the efficient neighbourhood lookup using LinkedCellsGrid to check for susceptible pedestrians.

`SIRGroup` contains essential information about a SIR group as well as adding and removing members. Furthermore it defines the type of the generic class `AbstractGroupModel`, which `SIRGroupModel` inherits from. This leads `SIRGroupModel` returning groups of type `SIRGroup` for the inherited functions.

`AttributesSIRG` contains the parameters of our SIR model and is saved as attribute in the `SIRGroupModel`. Infection rate and recovery rate can be changed here for example. Additionally the values in this class the initial parameters when loading the SIR model into Vadere and can be changed in the Vadere editor before running a simulation.

`SIRType` is simple enumeration class which contains 3 different states in the SIR model. This class is used in `SIRGroupModel` to determine the pedestrian state in the SIR model.

`FootStepProcessor` uses `SIRGroupModel` to log information about SIR simulation. This class is an additional output processor that can be linked in the Vadere. When running the SIR model this processor output additional information about the SIR model.

2. Description of the output processor In this part we use our own output processor `SecondGroupIDProcessor` instead of `FootStepGroupIDProcessor`, which can be found in the same folder as the other output processors. This class simply writes down at each time-step the pedestrian id and group id of the pedestrian. This way even pedestrians that did not move are tracked unlike in the footstep processor and this allows for a more direct output. Of course we could have set up a program to read the footstep processor output and memorize the group ID from the previous time-step if the next time-step does not include a pedestrian but this did not occur to us at the time.

In the visualization we use a python notebook that loads the file and extracts the group IDs in each time-step and then plots percentages of the group IDs. This is handled in the python notebook `visualizeSIR.ipynb` which generates graphs tracking changes of the SIR groups. *Note that for task 4 and task 5 the vadere project with its scenarios are placed in the folder* `Exercise-2\vadere-project-tas45`.

3. SIR group visualization To color the pedestrians based on their group we need to put the group id and its corresponding color into the `colorMap` in the class `SimulationModel.java`. We assigned the following colors to each group id:

- 0 = Red
- 1 = Black
- 2 = Blue

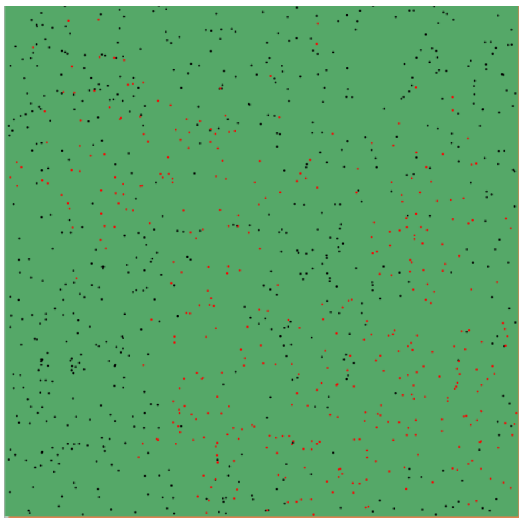
We also didn't use green or white for susceptible individuals, because the pedestrian spawning area is visualized with green and the background color is white. To visualize the pedestrians according to group color during simulation we need to set the `DefaultSimulationConfig.agentColoring = AgentColoring.GROUP`. It is noteworthy that due to our own implementation of the output processor in `SecondGroupIDProcessor` we have only implemented the visualization of groups during simulation. Visualization of groups in post-visualization is not supported in this iteration of our code.

4. Using LinkedCellsGrid to improve efficiency To increase the efficiency of finding nearby pedestrians we modified the `update(...)` function in `SIRGroupModel.java`. To get the neighbourhood of given radius more efficiently the LinkedCellsGrid slices the 2 dimensional space into uniform squares of given side length. It then only checks the squares which are less than the radius of the neighbourhood away from a given position's square.

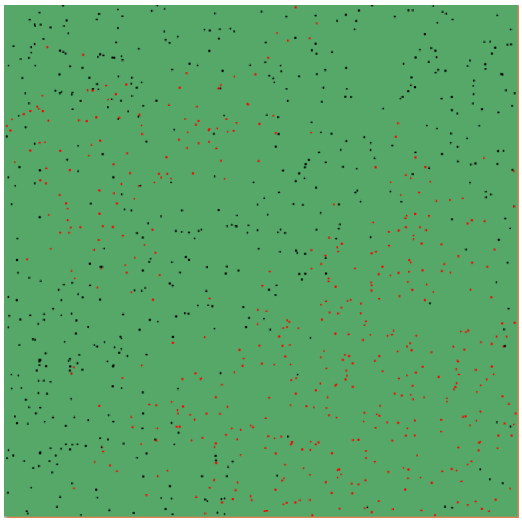
5.1 Test: comparing infection rates In this test we setup a large area in which we randomly placed a 1000 pedestrians uniformly distributed. In this test we want to compare how fast 50% of the pedestrians become infected depending on the infection rates. We have setup the same starting amount of 10 infected pedestrians in both scenarios. In this test we only changed `infectionRate`, but a full list will be provided at the end of this section. Running the tests yields the following results, where the `infectionRate` is the probability for and the time is how long it takes until 50% of the pedestrians are infected:

	Scenario (a)	Scenario (b)
`infectionRate`	0.01	0.05
time (in seconds)	36.2	4

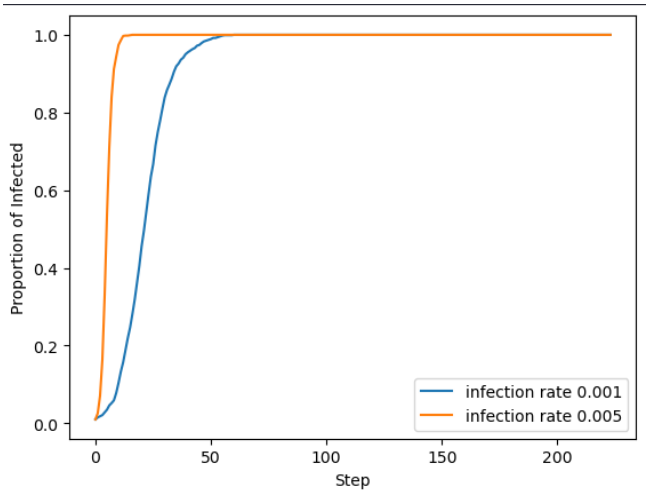
We see that with 5 times the infection rate the time until 50% of the pedestrians are infected decreases from 36.2 seconds to 4 seconds. We know from the mathematical nature of infection spreading that it has an exponential behaviour. Hence the immense decrease in time until 50% is infected is expected. In the figures 10 below we see the results after running both simulations. Further more in figure 10(c) we see a graph comparing the infection spread between both scenarios.



(a) infection rate 0.01 at second 36.2



(b) infection rate 0.05 at second 4



(c) Comparison between different spread rates

Figure 10: Comparison of the figure 6 simulations

Scenario 10(a) parameters

```
"infectionsAtStart" : 10,
"infectionRate" : 0.01,
"infectionMaxDistance" : 10.0,
"recoveryRate" : 0.0
"infectionRateWhenAdded" : 0.0
```

Scenario 10(b) parameters

```
"infectionsAtStart" : 10,
"infectionRate" : 0.05,
"infectionMaxDistance" : 10.0,
"recoveryRate" : 0.0
"infectionRateWhenAdded" : 0.0
```

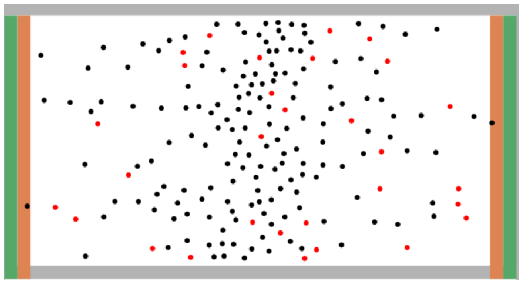
5.2 Test scenario corridor with counterflow In this test we have two groups of 100 pedestrians walking in opposing directions in a corridor of 40x20 meters size. The pedestrians are released over time. One of the

groups has 10 infected pedestrians. We now want to know how much infection spreads given that one group has no infected pedestrians.

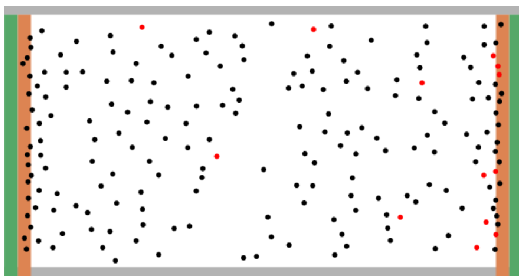
At first we ran the simulation with the infection rate set to 0.001 (1.0E-3) and the absorbers enabled, this did not allow for the pedestrian infection counts to be reliably counted and the infection spread even after the groups passed each other. We could have written some code to allow for getting the numbers of infected in each group from the resulting file and either extrapolating the number of pedestrians infected during inter-group contact or changed to code to allow for the groups to be set separate. Instead we first tried to achieve this with the tools at hand.

To minimize infection between group members and make the calculation easier we simply set the infection rate really low to 0.0001 (1.0E-4) and run the simulation with the target absorbers disabled. Observing the image we can easily count that two pedestrians got infected. From before in the simulation we can see that no infections between group members in the group originally without infected occurred before that point.

	Corridor (a)	Corridor (b)
`infectionRate`	$1.0E-3$	$1.0E-4$



(a) infection rate 0.001 at second 20



(b) infection rate 0.0001 after meet

(c) infection rate 0.0001 at the end

Figure 11: The corridor scenario

Scenario 11(a) parameters

```
"infectionsAtStart" : 10,
"infectionRate" : 1.0E-3,
"infectionMaxDistance" : 10.0,
"recoveryRate" : 0.0
"infectionRateWhenAdded" : 0.0
```

Scenario 11(b)(c) parameters

```
"infectionsAtStart" : 10,
"infectionRate" : 1.0E-4,
"infectionMaxDistance" : 10.0,
"recoveryRate" : 0.0
"infectionRateWhenAdded" : 0.0
```

6. Decoupling infection rate from simulation time-step To do this we initially assume that an infected pedestrian has a chance of $(1-p)$ to not spread the disease. Let the probability of an infected pedestrian not spreading the disease at a *single* time-step be $(1-q)$. Let n be the amount of time-steps in a second, we then have the following equation:

$$(1-q)^n = (1-p) \tag{1}$$

$$(1-q) = (1-p)^{1/n} \tag{2}$$

Note that the amount of time-steps n in a second is equivalent to $n = 1/t$, where t is the delta time or time since the last time-step has happened. Inserting this gives us the following:

$$(1-q) = (1-p)^{1/(1/t)} \tag{3}$$

$$(1-q) = (1-p)^t \tag{4}$$

We have shown that that if a pedestrian does not spread the disease with probability $1-p$ every second, the chance to not spread the disease in delta time t is $(1-q) = (1-p)^t$. Hence the chance that a pedestrian does spread the infection at a time-step is simply $1-(1-p)^t$.

To implement this we added an attribute `LastSimulationTime` to `SIRGroupModel.java`, which is used to calculate the delta time t. In the `update(...)` function of the same class we then use the expression $1-(1-p)^t$ to correctly calculate the probability of infecting a pedestrian. Note that the same formula is used for recovery as well (See task 5.2).

The simulation rate still technically affects the infection. The infection radius and infection ticks caused by the new infected could be taken into account. For example if we instead simulated the infection retroactively. This however would necessitate ignoring simulation rate to an extent and therefore seems like a sub-optimal solution. If we truly wanted to allow the user to remove this inaccuracy at the cost of processing power it could be possible to add a separate simulation rate setting for the infection spread.

7. Possible extensions So far the integrated SIR model only contains basic features to model the spread of infections. Including the implementation from Task 5 the model would additionally consider the recovery of people. Still the SIR model as presented in [1] considers removed people instead of recovered people and therefore we only model infection in a simplified manner. In the following we will discuss possible extensions to improve the model.

The first possible extension is allowing recovered people to infect susceptible people. Recovered people realistically can still carry the infection around for a certain time. Therefore recovered people would still be able to infect susceptible people.

The second possible extensions is allowing recovered people to become reinfected. Simply speaking an infection could mutate within a group of people after a certain time frame. This mutated infection would then also be able to spread to recovered people and reinfect them.

Further more on completing the SIR model, under removed individuals we can consider subgroups other than recovered people. Here for example we could add deceased, permanently immune and/or isolated individuals. Disregarding the realism for a moment in the Vadere simulation this would mean deceased pedestrians could be modeled as static pedestrians. Permanently immune individuals would have 0 chance of being reinfected, where as isolated individuals are modeled as pedestrians performing social distancing.

At this point it becomes clear that the SIR model considers infection from a very general standpoint with the least amount of assumptions about the diverse factors that play a role in reality. The SIR model implemented in Vadere assumes that infections are spread by being in the vicinity of an infected pedestrian. This assumes that infections are spread through human interaction only. In Vadere we can extend this implementation to contain other groups that are carriers of infections. For example certain pedestrians could be representing animals or insects. Similarly an infection that spreads through the air would could also be modeled as a group of infected pedestrians.

Beyond the assumption of modeling infections spreading in large crowds, we can also imitate the spread within social structures in Vadere. As shown in [6] an agent-based network provides an abstraction to model social connections. A majority pf people don't spend their time in crowds, but in certain environments with a set of other people e.g. workplace, school, university. This could be imitated in Vadere to a decently realistic degree. For example we can have the pedestrians form multiple smaller crowds at a sequence of targets and remain there for a certain time span. Here the targets can represent a home, workplace, school, supermarket,

etc. A potential issue in this model would lie within the movement of the pedestrians between the targets. While infections do occur when traveling between different locations, this proposed model does not explicitly capture infections during traversal.

Overall we conclude this discussion with the fact that spread of infections in reality are very difficult to model due to a variety of factors. Every model is only an approximation of reality and depending on the assumptions made in the model, they provide a decently accurate depiction on certain aspects of it.

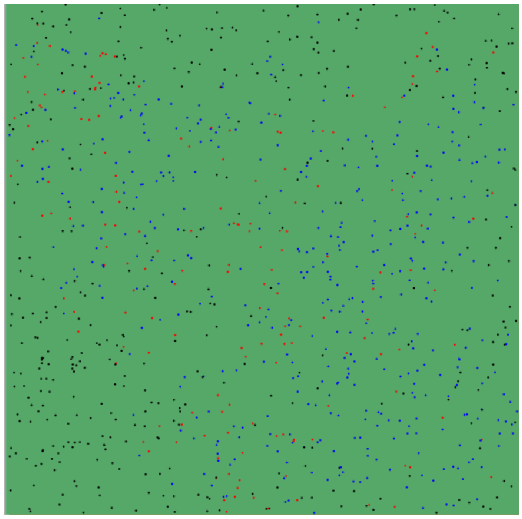
Report on task 5, Analysis and visualization of results

Adding recovery To add a recovered state we assign recovered pedestrians to the group id for recovered individuals. To make recovery work we first added the attribute `recoveryRate` to `AttributesSIRG.java`. Similar to decoupling the infection rate probabilities from the time-steps we used the same expression $(1-p)^t$ as in Task 4.6, where p is the probability to recover and t is the delta time since the last time-step. At the end of the `update(...)` function in `SIRGroupModel.java` we check if pedestrians have recovered and then assign them to the recovered group id accordingly. Due to the simulation visualizing pedestrians according to groups, the simulation will change the colors of pedestrians for recovered individuals to blue.

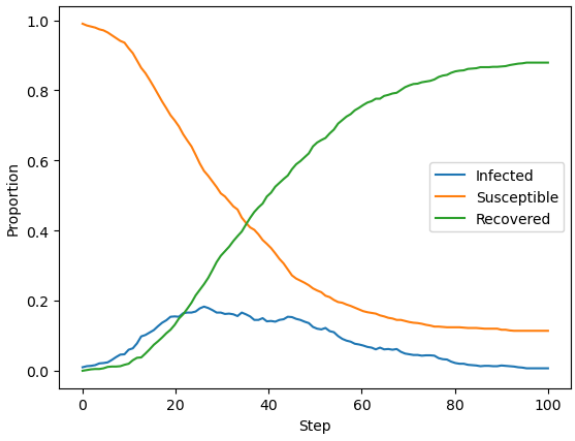
Test 1 In this test we repeat the test from task 4.5.1, where we had 1000 static pedestrians in an large area. Out of those 1000 pedestrians 990 are susceptible and 10 are infected. Unlike in task 4.5 pedestrians are now able to recover. We used the following parameters for the experiment:

```
"infectionsAtStart" : 10,
"infectionRate" : 0.05,
"infectionMaxDistance" : 10.0,
"recoveryRate" : 0.1
"infectionRateWhenAdded" : 0.0
```

Once 50% of the pedestrians had been infected we will also observe recovered pedestrians as shown below in Figure 12(a). We can observe that a majority of the pedestrian have already become recovered in this case. Plotting the overall amount of susceptible, infected and recovered pedestrians over time we receive graphs seen in Figure 12(b), which furthermore confirms our initial observations. Here we see the rapid decline in susceptible pedestrians as they become infected. As for the infected pedestrians we see an increase at the start, but due to the higher recovery rate the amount of infected pedestrians quickly declines to zero. Consequently we see a steady rise in recovered pedestrians.



(a) Infection rate 0.01 with recovery rate 0.1 at second 37



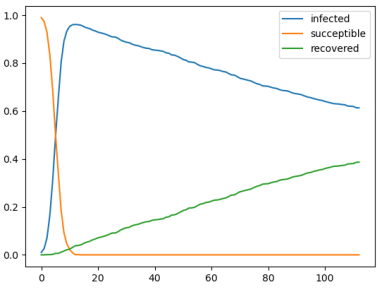
(b) Infection rate 0.01 with recovery rate 0.1 SIR curves

Figure 12: Test 1 observations

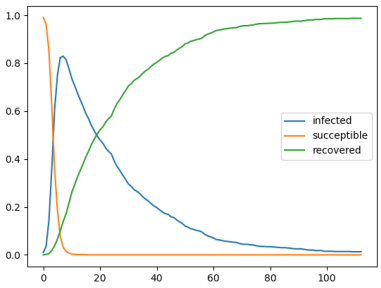
Test 2 In this section we perform further tests based on the scenario given in the previous section "Test 1". In this test we experiment with different infection rates and recovery rates, which are captioned below the figures 13(a)-(h) as $(infectionRate, recoveryRate)$. The graphs show on the x-axis the time-step and on the y-axis the percentage of pedestrians in each group. Additionally we added a ratio $r = \frac{infectionRate}{recoveryRate}$. The graphs are sorted in ascending order of the ratio r.

We can generally observer that with increasing infection rate, the amount of infected increases exponentially faster, which also shows in the faster decrease in susceptible pedestrians. Similarly we observe that faster

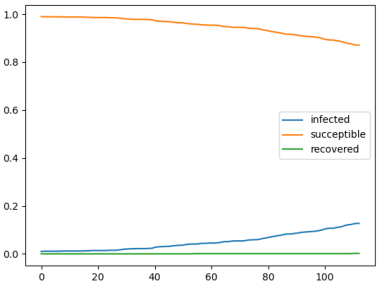
recovery rates lead to an exponential increase in recovered pedestrian which saturates eventually. Notably in Figure 14(c) we observe that with significantly higher recovery rate the infection is unable to spread to all susceptible individuals. Hence the recovery curve is much steeper and saturates much earlier on. In 14(a) we can see that having the recovery rate be only slightly higher does not produce the same effect. In 14(b) we can see that having the recovery rate be 6 times the infection rate with our infection radius leaves only few pedestrians uninfected and is therefore close to the threshold.



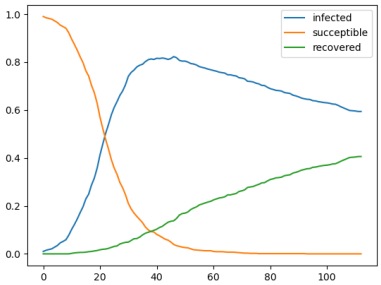
(a) (0.01, 0.01), r=1



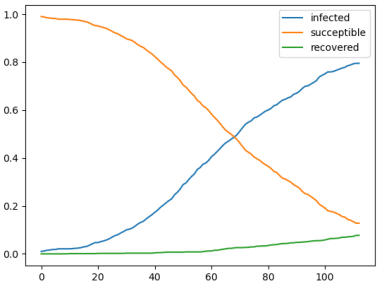
(b) (0.07, 0.05), r=1.4



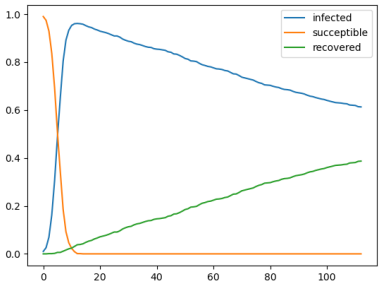
(c) (0.001, 0.0007), r=1.429



(d) (0.01, 0.007), r=1.429

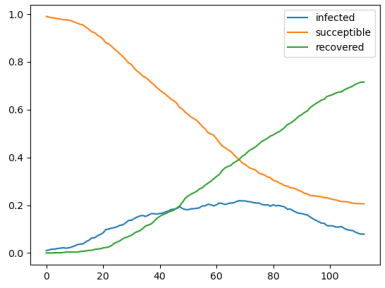


(e) (0.003, 0.002), r=1.5

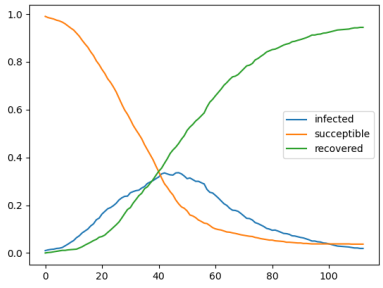


(f) (0.05, 0.005), r=10

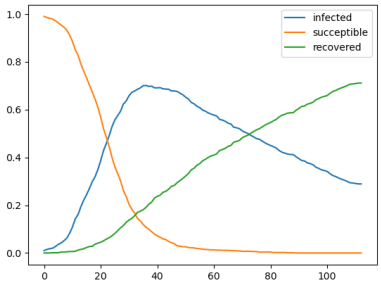
Figure 13: Test 2 observations



(a) (0.005, 0.05), r=0.1



(b) (0.01, 0.06), r=0.166



(c) (0.01, 0.015), r=0.66

Figure 14: Test 2 preventing infection of all pedestrians

Table of parameters: Here again an overview of the parameters and their ratios used for the graphs in 13 and 14.

Figure	Infection Rate (IR)	Recovery Rate (RR)	IR/RR-Ratio
13a	0.01	0.01	1
13b	0.07	0.05	1.4
13c	0.001	0.0007	1.429
13d	0.01	0.007	1.429
13e	0.003	0.002	1.5
13f	0.05	0.005	10
14a	0.005	0.05	0.1
14b	0.01	0.06	0.167
14c	0.01	0.015	0.667

Supermarket test In order to test the infection model in a real-case scenario, we have designed a supermarket inspired on the ALDI SÜD at Wotanstraße 9c (Figure 15a). This supermarket is not very big so it is one where infections are more likely to occur. The supermarket has a main door where the checkouts are placed in front. To enter the actual supermarket, you enter through the left zone of the door, where the fruit and vegetable area is located with the meat shelves in front, a place that as we will see is quite crowded. Then, one of the most prominent areas is the corridor at the end on the left, where the dairy products are, as well as pasta, eggs and other foodstuffs that are quite necessary in people's diets. This is also where a lot of people tend to accumulate (We have personally verified this). Then, we have different aisles with less required foods such as biscuits, soft drinks and alcoholic beverages, as well as cleaning products. Behind the checkouts, the supermarket has its freezer and clothing sections.

We have modelled the scenario in such a way that we try to faithfully imitate the crowds of people we have seen during our personal experience in the supermarket. So each aisle has targets, which is where people tend to go and stand. For example, the last aisle on the left is one of the most populated ones, so it has more areas where people go interested in what's there. However, the corridor next to it contains refreshments and water,

so apart from being less crowded, people don't tend to walk the whole aisle (Figure 15b).

Finally, we also created a variety of sources, where each of them spawn different types of customers. For example, one of them models a type of customer who makes a complete purchase (meat, vegetables, pasta, drinks, cleaning products, goes to the checkout and leaves). Another models a type of customer who simply wants snacks so his path is shorter and more direct (he goes to the biscuit aisle, the snack aisle, goes to the checkout and leaves). Then we basically added variations of these two main customer types (long path and short path). One source generates pedestrians who buy food to host a party (snacks, alcoholic beverages and soft drinks) or another source generates customers who do a complete shopping, but instead of cleaning products they buy frozen food. In addition, we observed that most people do complete purchases, so this is also modelled. There will be more people of the first type of customer than of the second type.

We achieve this behaviour of the pedestrians adding a list of targets to the source and setting the targets' absorber property to false.

(a) A photo of the supermarket from the checkout

(b) The supermarket scenario

Figure 15: Modelling of the ALDI SÜD

After designing the scenario, we can start to test the model in it.

We did a dual test on the scenario. First, we observed how the model behaves without social distancing, and then we test the simulation with social distancing.

The attributes for the SIR model are same for both test:

`"org.vadere.state.attributes.models.AttributesSIRG"`

are these

```
"infectionsAtStart" : 0,
"infectionRate" : 0.05,
"infectionMaxDistance" : 1.0,
"recoveryRate" : 0.0,
"infectionRateWhenAdded" : 0.2
```

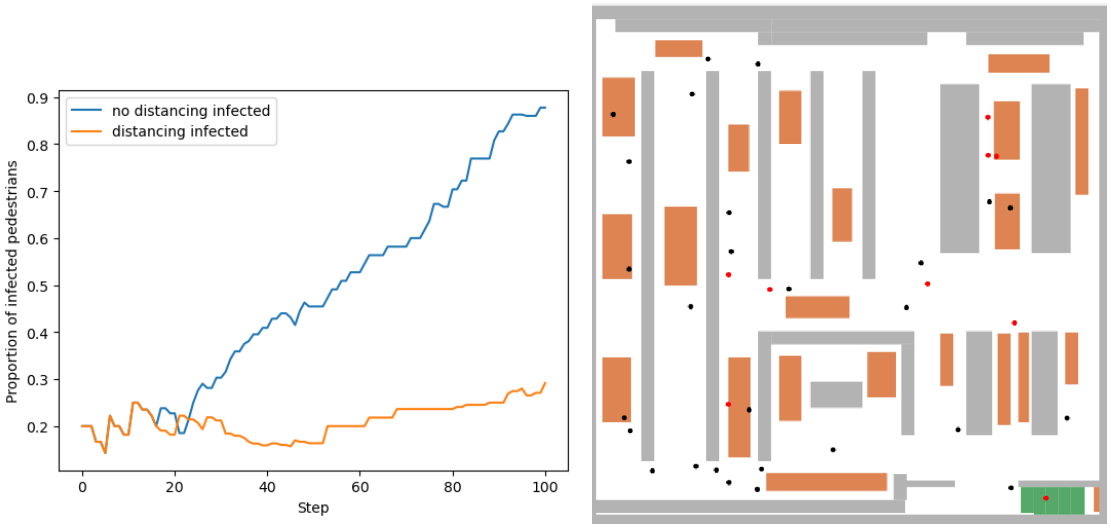
However, the attributes for `org.vadere.state.attributes.models.AttributesPotentialCompactSoftshell` are different in both cases.

In the test without social distancing we have the following attributes:

```
"pedPotentialIntimateSpaceWidth" : 0.2,
"pedPotentialPersonalSpaceWidth" : 0.5,
"pedPotentialHeight" : 50.0,
"obstPotentialWidth" : 0.8,
"obstPotentialHeight" : 6.0,
"intimateSpaceFactor" : 1.2,
"personalSpacePower" : 1,
"intimateSpacePower" : 1
```

We decrease the `pedPotentialIntimateSpaceWidth` space in order to be able decrease the **pedPotentialPersonalSpaceWidth** because our scenario is quite small and we need the personal space to be smaller than the infection radius (precisely half).

In the test with social distancing `pedPotentialPersonalSpaceWidth` is increased to 2.0 to double the infection radius so the difference is substantial enough to distinguish from noise.



(a) Difference between distancing test

(b) Simulation with social distancing

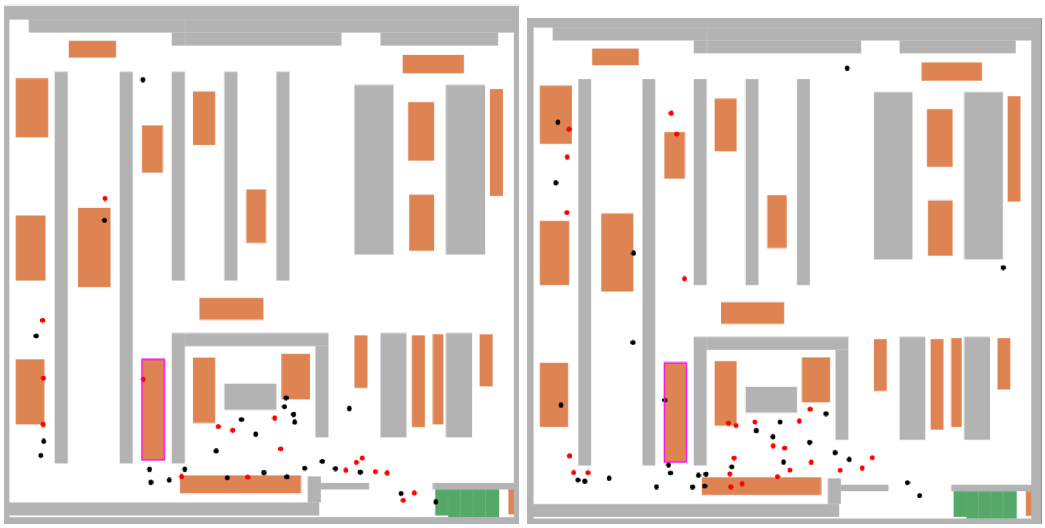
Figure 16: Supermarket test with and without social distancing

As we can observe in Figure 16a, it is very clear that the distancing is helping to prevent the infection from spreading. In the test without social distancing, the 90% of the pedestrians get infected in the end of the simulation, whereas with social distancing, the proportion of infected pedestrians remains almost constant, not surpassing the 30% of infected people.

In addition, we have made a study about the most dangerous zones. Given the intersections of the paths that pedestrians follow, i.e. the areas that will have the highest density as more paths pass through them; we can detect which are the zones with more probability to spread the infection. According to our model's paths based on our real experience in the supermarket, the main ones are these 2 zones:

- **Vegetable, fruit and meat zone**: Many paths coincide in the vegetable, fruit and meat zone, mainly because is the entrance and the food in that zone is very demanded (that's why there are 3 targets very close).
- **Supermarket check-outs**: Obviously the other zone where all the paths converge are the supermarket check-outs, being a zone that can reach high densities of people. However, in the model is less dangerous that in the real world, as it is not implemented in the scenario that pedestrians have to wait their turn. So they pass by, being less populated because each pedestrian arrives in different steps at the check-out depending on their path.

Also, the last corridor on the left is a very crowded place where many paths converge, as we commented in the designing of the scenario, but being a little less dangerous as pedestrians go in the same direction.



(a) People entering the danger zone at time t (b) Simulation at $t + 2$

Figure 17: Vegetable, fruit and meat zone

An example of a danger zone can be viewed in Figure 17, where we can observe one of the highest density of people, and thus, a potential infection point.

Moreover, in Figure 16, the increment of infected people is almost linear without social distancing. On the other hand, with social distancing, in spite of being mainly constant, we can observe that the highest infection rates are achieved in the beginning and in the end, coinciding with the entrance to the vegetable, fruit and meat area, and the supermarket checkouts respectively.

In short, although social distancing keeps infection under control at less than 30%, this is still a high number in the case of a real scenario. We believe that the best way to reduce the number of infected without restricting too much the number of customers that can enter the supermarket, would be to change mainly the organization of the danger zones. Therefore, if you know that meat, fruit and vegetables are very demanded, increasing the distance between each other would be helpful. Also, it would be good that the most demanded food is not in front of the door of the supermarket, as this ALDI horribly does, forcing the paths of all possible types of customers to coincide in this already crowded area. On the other hand, check-outs are more difficult to manage as every customer's path need to pass through there. It would be a suitable option, if possible, to provide two or more check-out areas with their respective doors to the outside, so as to reduce the density in each of them.

Bonus tests: TUM Main Campus exit For the bonus tests we have created a new scenario imitating another real case that we have experienced: Leaving TUM's main campus when classes are over.

Simplifying the actual architecture of the university exit, the scenario consists of two corridors, facing each other, with classes in each. The class exits are modelled as sources, which are placed opposite each other along the corridor. Looking at the number of people in a TUM class we have assumed that each class contains 30 people.

The aim of the simulation is to test how the infection spreads in a common and chaotic situation such as a school exit, where all pedestrians have a very similar path. We have modelled the two doors through which students must pass to reach the actual exit, which is more extensive, as two targets.

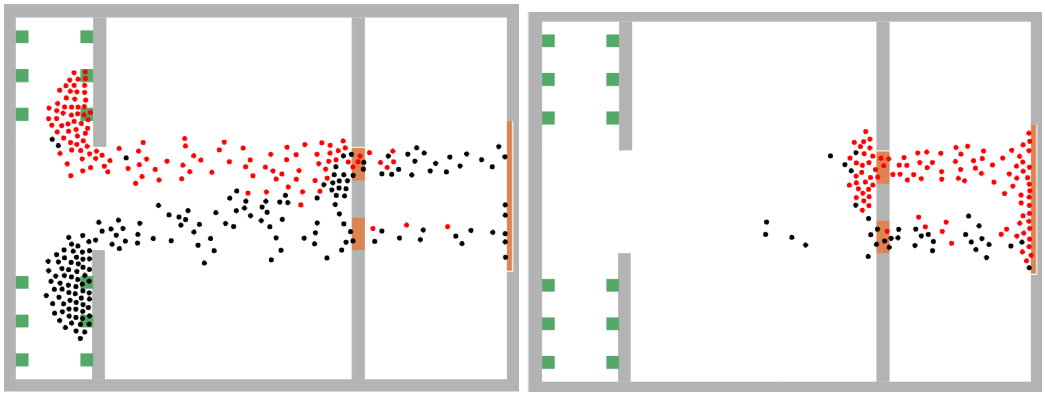
This way, pedestrians are forced to pass through but it also allows us to have control over which gate pedestrians will use. Therefore, to better simulate a real situation, we have forced two groups of pedestrians, each of them belonging to a classroom, to go through the opposite door to the one closest to them, as sometimes happens in reality, where there are people who go through the door that a priori is the least intuitive.

(a) Entrance/exit TUM Main campus (b) Scenario

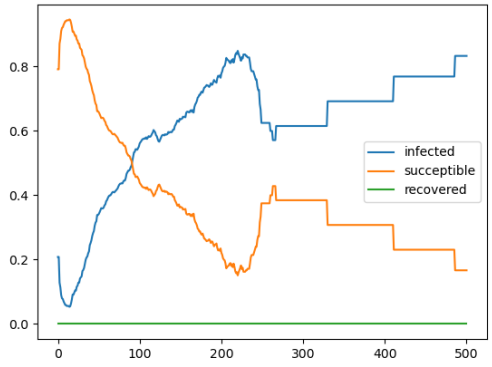
Figure 18: TUM Main campus exit modelling

We have added 10 infected pedestrians to the northern group. In this way, we will be able to observe how the infection spreads and if it reaches people coming out of the southern corridor.

First, we have done a test without social distancing. As we can observe that, cause of not having social distancing, almost all northern pedestrians get infected before leaving the corridor, as they accumulate on the wall, sticking closely together (Figure 19). Also, we can see how the pedestrians that belonged to the southern group that pass through the northern gate (the black dots that are transferred to the other group), become infected. Then, the infected group that goes through the southern gate infects most of the healthy ones, reaching 80%, as shown in Figure 19c.



(a) Simulation middle (b) Simulation finish

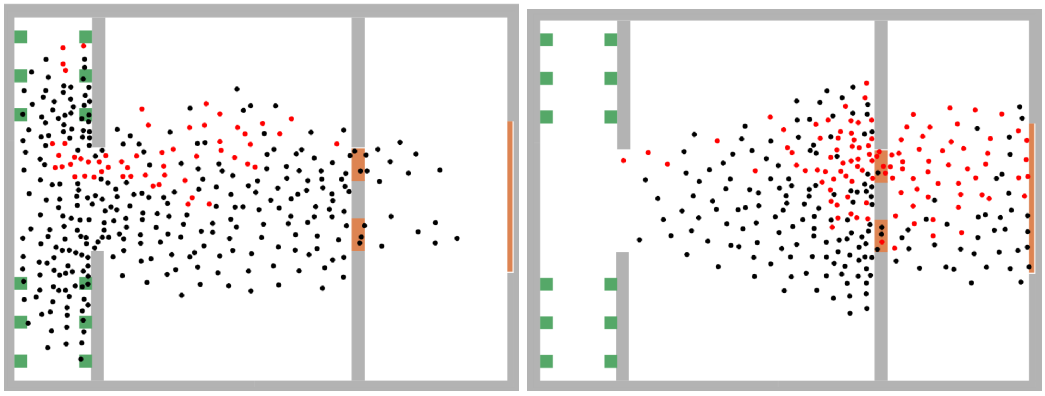


(c) Percentage of infected over time

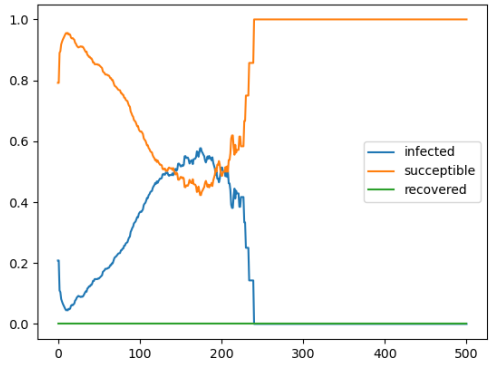
Figure 19: Simulation without social distancing

Moreover, in the plot we can observe how the infection grows very fast at the start, when all pedestrians start to accumulate on the wall (until the middle of the graph). Obviously, there a no recovered pedestrians, not making sense in this real case scenario. After the middle of the graph, we can observe how the amount of people is reduced in a kind of pattern, being this a sign that the pedestrians are exiting the door, i.e. reaching the final target and disappearing in a ordered pattern.

In the test with social distancing, we can see that this time the pedestrians do not all pile up together when leaving the classroom, allowing those already infected to move on without infecting the entire corridor population (Figure 20a). However, as soon as they reach the area of the two doors, it is more difficult to separate, and added to the fact that groups of infected people enter the south door, where most of the healthy people are, the proportion of infected people increases (Figure 20b). Finally we can observe this behaviour looking at Figure 20c, where the slope of the graph is a little gentler at the beginning, but steepens when most of the pedestrians reach the danger zone of the two gates, until it finally reduces the population when exiting. We can also observe that with social distancing pedestrians arrive earlier to the exit, due to a less bottleneck in the two corridors.



(a) Simulation start (b) Simulation middle



(c) Percentage of infected over time

Figure 20: Simulation with social distancing

Finally, comparing both experiments' plots, it can be seen that social distancing reduces the extent of infection to 60% in contrast with 80% obtained without social distancing. However, in both cases the infection reaches a high proportion of the population, showing us that this scenario in a real-life situation is dangerous and should be treated with caution. Assuming that the architecture of the building cannot be changed, we think that the best way to leave the Main campus of Tum in an infection situation is by individually organizing the classes to leave at a certain time each, avoiding contact with the rest of the classes at the exit and in the corridors.

References

[1] Nino Boccara. *Modeling Complex Systems*. Springer, 2nd edition, 2010.

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

[3] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[4] Michael J Seitz and Gerta Köster. Natural discretization of pedestrian movement in continuous space. *Physical Review E*, 86(4):046108, 2012.

[5] James A Sethian et al. *Level set methods and fast marching methods*, volume 98. Cambridge Cambridge UP, 1999.

[6] Christopher Wolfram. Agent-based network models for covid-19, Nov. 11, 2023 [Online].