Report for exercise 2 from group G

Tasks addressed:	5
Authors:	Angelos Kafounis
	Caner Karaoglu
	Joaquin Gomez Sanchez
Last compiled:	2022–08–31
Source code:	https://github.com/joaquimgomez/ml-crowds-group-g

The work on tasks was divided in the following way:

Angelos Kafounis	Task 1	30%
	Task 2	30%
	Task 3	30%
	Task 4	35%
	Task 5	35%
Caner Karaoglu	Task 1	30%
	Task 2	30%
	Task 3	30%
	Task 4	35%
	Task 5	35%
Joaquin Gomez Sanchez	Task 1	40%
	Task 2	40%
	Task 3	40%
	Task 4	30%
	Task 5	30%

Report on task 1, Setting up the Vadere environment

For this first tasks we must recreate two RiMEA scenarios, i.e., the Scenario 1 or Straight Line scenario and the Scenario 6 or Corner scenario, as well as the chicken test, as we did for the first exercise.

In the following sections can be seen the description of the implementations of the three scenarios together with the simulation results and differences between Vadere results and our results from the first exercise. The simulations will be done with the Optima Steps Model (OSM).

Apart from the simulation and automata differences that will be latterly stated, the main difference between our implementation and Vadere is the application itself. While Vadere is an entire program, our implementation only consists of an Automata class and some visualization helpers that need one support, in our case Jupyter Notebooks. What is the same in both is the description of the scenarios in JSON files. This proves our reasoning while developing the Exercise 1 that a JSON file is the best option to save information in a human and computer understandable way.

RiMEA Scenario 1 - Straight line

As can be observed in the scenario file `Task1-RiMEA-Scenario-1` available in the GitHub repository, we have recreated the scenario keeping accurately the proportions of 2x40 (m) specified by the RiMEA description. We have set a velocity of 1.33 m/s in both directions, since the description does not specify otherwise, and a body size of 40 cm.

Simulating we can observe that the pedestrian reaches the target in ∼32s and this is a time between the expected boundaries of RiMEA description, i.e., between 26 and 34 seconds.

Comparing the simulation with our one from the first exercise we can conclude that both are the same, because we obtain kind of the same traversal time and the movement is constant. The main differences come from the visualization. While our in our implementation the cells of the grids are clearly visible, in Vadere we don't have a visible grid or at least the grid resolution is sufficiently big to avoid the visualization of the space discretization.

In the following Figure 1 we can observe a frame at second 24 from the simulation.

Figure 1: Frame of the simulation of the Straight line scenario at second 24

RiMEA Scenario 6 - Corner

(a) Scenario 6 after simualting

(b) Frame from the simulation at second 9

Figure 2: Initial state and state and second 9 for the RiMEA Scenario 6

For this scenario (available in the file `Task1-RiMEA-Scenario-6`) we have a corner and a set of 20 pedestrians uniformly distributed at the beginning of the corridor. For the dimensions of the scenario we have followed the description of RiMEA and for the pedestrians characteristics we have set for each one a velocity of 1.33 m/s for both directions an a radius of 0.2 meters. The initial state can be observed in the Figure 2a.

As we can observe in the Figure 2b, which consists in a frame at second 9 of the simulation, we pedestrians behave as expect, turning the corner those that are close to it. The simulation until all the pedestrian reach the target takes 30 seconds.

The differences between the Vadere simulation and our one in the Exercise 1 are the same, but for the Vadere simulation we can appreciate that the distribution of the pedestrians in the spaces seems more normal and equal. This we suppose is due to the resolution of the grid, as we argue for the previous scenario.

Chicken test

For the chicken test or rotated U test we have followed the figure of the Exercise 1 Sheet. As we do not have a specific description, we have decided to create a grid of 9x9 meters and a U of 4x6 meters. The pedestrian and the target are placed in the same horizontal plane, separated by the U.

Broadly speaking, the pedestrian behaves as expected and in the same way as our simulation in the Exercise 1. The difference with our one is that while walking above the U the pedestrian does not follow a straight path but it seems to follow a sinusoidal/wave path. This can be appreciated in Figure 3

Figure 3: Evolution of the simulation of the pedestrian above the U

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

For this task we have to simulate the three previously described scenarios (which will not be described again here), but with the Social Force Model (SFM) and with the Gradient Navigation Model (GNM).

Social Force Model (SFM) Simulations

RiMEA Scenario 1 - Straight line

For this first scenario the pedestrian behaves as expected. It follows a straight line from the origin to the target, but going up a bit at the begging to get straight again. It takes ∼33s.

Broadly speaking, OSM and SFM behave in the same way for this scenario. The unique difference is the strange movement a the beginning with SFM and that this last takes 1 more second compare to the 32s of OSM. The strange movement could be justified with the tendency to oscillatory movement caused by SFM, that is unnatural to pedestrian movement.

RiMEA Scenario 6 - Corner

The pedestrians in this scenario with SFM behave correctly going from the origin to the target. In this simulation, the uniformly distributed pedestrians collapse in a straight line at the beginning. Then, they follow a more or less equal path really close to the corner. After turning the corner, the pedestrians follow a straight line but now more scattered, specially after and before other pedestrians. The pedestrians in the second half of the distribution in the space, when more pedestrians in the first half arrive to the target, seem to have less velocity and the space between pedestrians before and after each one is bigger. We can say that the uniformity seems to break.

This last point is the main difference between the simulation with OSM and SFM. For the OSM, the uniform distribution does not seem to break and the pedestrians either collapse to a single straight line. This collapsing in SFM causes a total number of 153 overlaps between pedestrians, something that does not happen with OSM.

The simulation time from the origin to the target is quite similar: for OSM it takes ∼30 seconds and for SFM ∼28.8 seconds. The described behave can be observed in Figure 4.

The overlapping problem is something common in SFM, as stated by Seitz and Köster [1] . The different environments after and before turning the corner could be explained by the equilibrium between repulsive and attractive forces in the model.

(a) Collapsing at the beginning of the simulation

(b) Scatter after turning the corner

Figure 4: Frames from the simulation of the RiMEA Scenario 6 using SFM

Chicken test

For the chicken test, the pedestrian stars trying to follow a straight line from the origin to the target, but once it faces the first vertical plane with obstacle elements, it seems to recompute the route and borders the U really close to it.

The simulation with SFM differs from the one with OSM in this last point. For OSM the pedestrian always avoid to be close to the U and it even has a small oscillation when turning the last corner of the U, as explained before. This is completely different from the behavior with SFM.

Gradient Navigation Model (GNM) Simulations

RiMEA Scenario 1 - Straight line

For this Straight line scenario the pedestrian behaves as expected. It follows a straight line from the origin to the target in ∼32 seconds.

The GNM simulation for this scenario is the same as with OSM, with no visible difference. Compared to SFM the difference is the same as for OSM, the strange movement at the beginning that does not occur for GNM.

RiMEA Scenario 6 - Corner

For this scenario the pedestrians, while maintaining a separation with other ones, go directly to the corner, causing a fake bottleneck, because all them are trying to turn the corner at the same time and very close, without using all the available space. After turning the corner, the pedestrians follow a straight line glued to the wall, instead of dispersing through space.

For this simulation Vadere returns that 4 overlaps occurred, but we have visually detected no overlap.

This simulation with GNM differs from other two in the aforementioned fake bottleneck in the corner and the close-to-the-obstacle bordering movement after turning the corner (see Figure 5). As analyzed by Dietrich et al. [2], seems to be normal for GNN, a model that tends to remain close to the obstacles, while others "prefer" to put distance.

(a) Bottleneck in the corner (b) Movement close to the obstacle after turning the corner

Figure 5: Frames from the simulation of the RiMEA Scenario 6 using GNM

Chicken test

As happens in the previous scenario, for this one the pedestrian turns very close the U. This is similar to the explained before behavior with SFM, but for GNM the pedestrian goes even closer to the obstacle (see Figure 6 for a comparison of the three models). Apart from this, the GNM simulation is the same as using SFM or OSM.

(a) OSM (b) SFM (c) GNM

Figure 6: Chicken Test simulation with OSM, SFM and GNM

Report on task 3, Using the console interface from Vadere

Firstly, for this task 3, we have to compare the outputs for a simulation using the Vadere GUI and the Vadere Terminal Simulator. At least with the default configuration of Vadere, for both options the outputed files are `overlapCount.txt`, `overlaps.csv`, `postvis.traj` and the scenario file. Comparing both outputs using the command `diff`, we can conclude that there is no difference between both types of simulation.

For the second part of the task, on which we have to modify an scenario using code, we have created the script `ModifyScenario.py`. This is a Python scenario that receives as input (using `argparse`) the arguments:

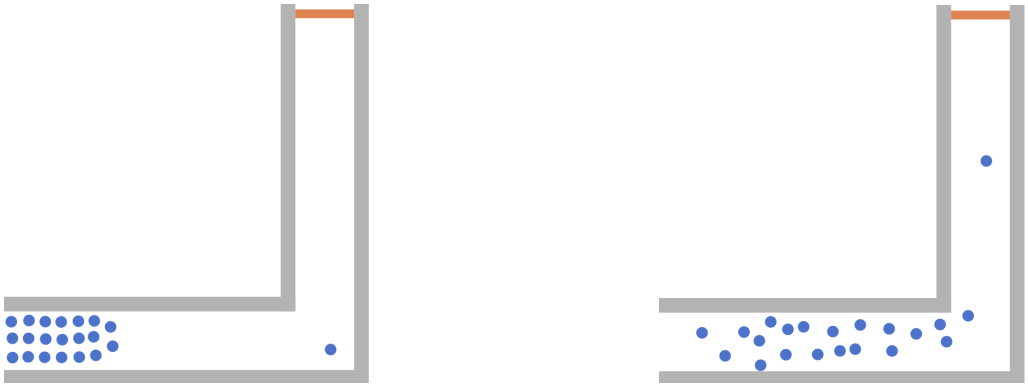
- `file`: Argument used to indicate the scenario file path.
- `addpedestrians`: Argument used to indicate the pedestrians to be added. It expects a string with the format `"[ (targetId, positionX, positionY, velocityX, velocityY), ... ]"`.
- `store`: If set, the script saves the modified scenario in a new file. If not set, the script works with a temporary file.
- `execute`: If set, the script calls `vadere-console.jar` as expected and using the modified scenario.

We have think this script in some way that allows for posteriors modifications, for example in order to add obstacles, just by creating a new function and adding the required arguments and calls in the `main()` function.

The scenario is saved as a Python dictionary, since it is the easier way to map a JSON to a Python data structure, and is modified accessing the different elements.

The method `addPedestrians(scenario, pedestriansStr)` receives an scenatio to be modified and after converting the string `pedestriansStr` (in the format previously explained), it adds to the scenario the new pedestrians. To add them, the function uses an schema (saved in `pedestrianSchema`), which is the default pedestrian added when using Vadere, but with Nones in `targetId`, `positionX`, `positionY`, `velocityX` and `velocityY`.

Using the explained script, we add a pedestrian in the position (13, 3) of the Scenario 6 (Corner) of the Task1, i.e., in the corner and away form any obstacle. The used command to execute the the script in order to obtain the explained scenario is: `python3 ModifyScenario.py scenarios/Task1-RiMEA-Scenario-6.scenario --addpedestrians "[ (100, 13, 3, 1.33, 1.33) ]" --execute`. In this case we do not need the option –**store** since gthe exectuion of the simulation already save a copy of the scenario. The result can be seen in Figure 7.



(a) Initial state (b) Simulation after few seconds

Figure 7: Frames from the simulation of the RiMEA Scenario 6 using modified with our script

Simulating with this new scenario using the Vadere console version and looking at the result with the Post-Visualization option, we can see that there is no differences between the original and the new scenario, apart obviously from the new pedestrian. The inserted pedestrian takes 10 seconds to reach the target, while the last original pedestrian that reach the target takes 30 seconds.

Report on task 4, Integrating a new model

For this fourth task we have to integrate the SIR model, a model to study the spread of an infection within a population, taking into account Susceptible, Infective and Removed persons.

To integrate the model, we have add the following given files to its corresponding packages.

- `AbstractGroupModel.java`. Added to the path `vadere/VadereSimulator/src/org.vadere.simulator/models/groups/`, it contains the abstract class `AbstractGroupModel` required for implementing the class `SIRGroupModel`. No changes have been made to this file.
- `SIRGroup.java`. Added to the path `vadere/VadereSimualtor/src/org.vadere.simulator/models/groups/sir/`, it contains the base class for the SIR model. No changes have been made to this file.
- `SIRGroupModel.java`. Added to the path `vadere/VadereSimulator/src/org.vadere.simulator/models/groups/sir/`, it implements (extending the class `AbstractGroupModel`) the SIR model, considering initially the groups susceptible and infected. Changes have been made to this file which will be explained later.
- `SIRType.java`. Added to the path `vadere/VadereSimulator/src/org.vadere.simulator/models/groups/sir/`, it only contains an enumeration for the IDs of infected, susceptible and removed groups. No changes have been made to this file.
- `AttributesSIRG.java`. Added to the path `vadere/VadereState/src/org.vadere.state/attributes/models/`, it defines the attributes such as infections at start, required by the SIR model. Changes have been made to this file which will be explained later.

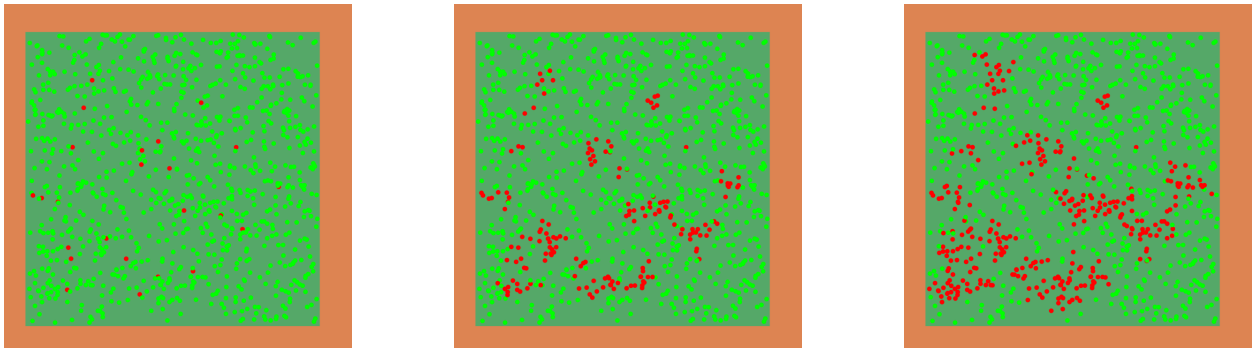
- `FootStepGroupIDProcessor.java`. Added to the path `vadere/VadereSimulator/src/org.vadere.simualtor/projects/dataprocessing/processor`, it defines a processor to output the results from the groups in the SIR model.

We have that this SIR model, essentially defined in the file `SIRGroupModel.java`, works as a submodel for, e.g., the Optimal Step Model. It submodels the interaction between pedestrians depending on the group of each one. For example, if one infected pedestrian is close to another that is not infected, this last one will be randomly infected, depending on the infection rate. The definition of the groups, i.e. the IDs, are in the `SIRType.java` file and the different model properties are specified using the available attributes included in the `AttributesSIRG.java`. While simulating, if we want to obtain an output, we have to use the `FootStepGroupIDProcessor`, which allows us to obtain an output file with the state of the pedestrians at every step.

In order to allow for the visualization of groups, we have modified the method `getGroupColor` in the file `VadereGui/src/org/vadere/gui/components/model/SimulationModel.java`. We have assigned, depending on the group ID the colors red to the group 0 (infected), green to the group 1 (susceptible) and blue to the group 2 (recovered). Our solution allows for visualizing the color while simulating.

As suggested in the exercise sheet, we have improved the efficiency of computing the distance for neighbors. We have modified the method `update` from the `SIRGroupModel`, which is the one that updates group of the pedestrians depending on the distance to the neighbors and their status. For this purpose, we instantiate a `LikedCellsGrid` from `org.vadere.util.geometry` to obtain a linked cells grid for every pedestrian and a list of neighbors. This grid representation will allow us to get the neighbors of every pedestrian more efficiently. Then, for every pedestrian (if we have some pedestrian), we obtain its grid and its neighbors. If the pedestrian is removed we have nothing to do and if the pedestrian is infected then it will be randomly removed. This last aspect is not required for this task, so in order to enable/disable this option we have added to the SIR attributes a boolean attribute `recoveryEnabled`. We also have already incorporated the required attribute for the recovery, i.e., the recovery rate. If the current pedestrian is not removed nor infected, then it will be randomly infected if some of its neighbors is already infected.

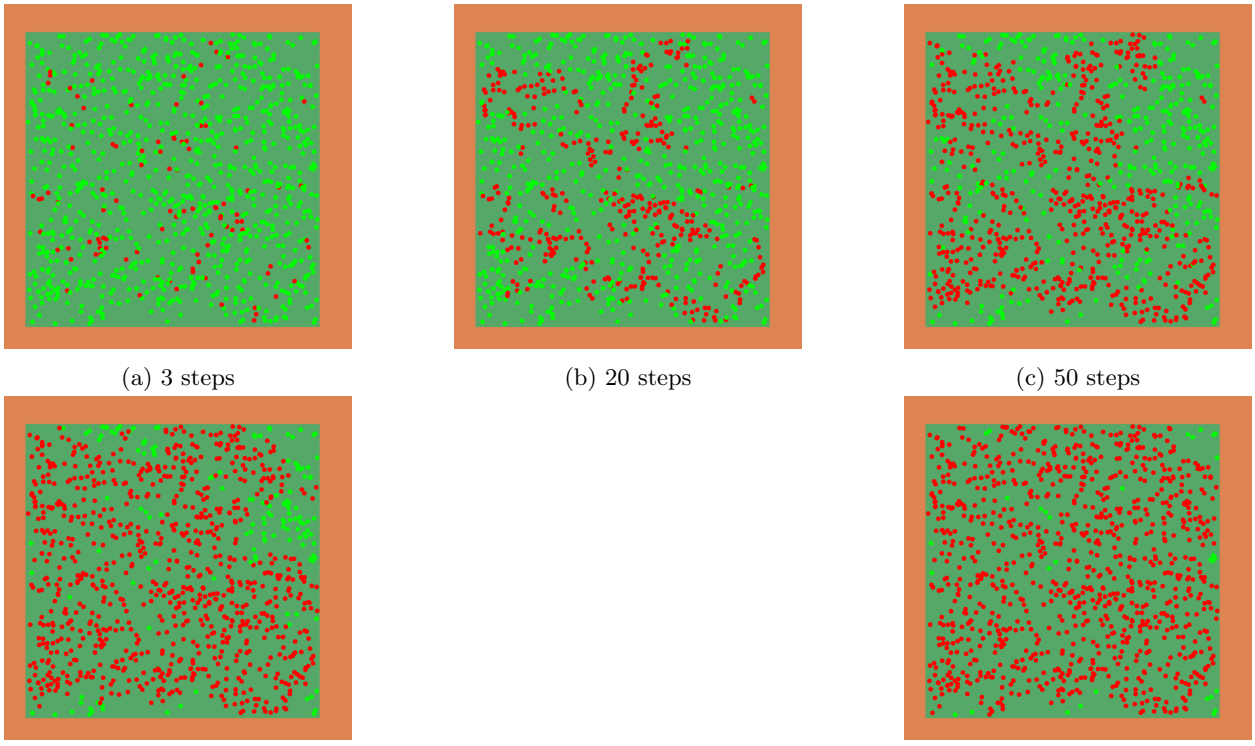
Now, let us test the explained implementation of the SIR model. For this purpose we have created, as suggested in the exercise sheet, a grid of 30x30 meters (effective or source area of 25x25 meters) with 1000 static pedestrians overlapping a target and a source that produce the pedestrians randomly. The file containing the scenario is `Task4-1-Static1000Crowd.scenario`. Using an infection rate of 0.01, 10 infected pedestrians at the beginning and an infection max distance of 1.0, we obtain for the first 200 time steps the evolution of Figure 8. We can clearly observe the evolution of the infected clusters, as expected in an infection transmission.



(a) 3 steps (b) 100 steps (c) 200 steps

Figure 8: Evolution of the simulation for the static scenario with an infection rate of 0.01 (grid)

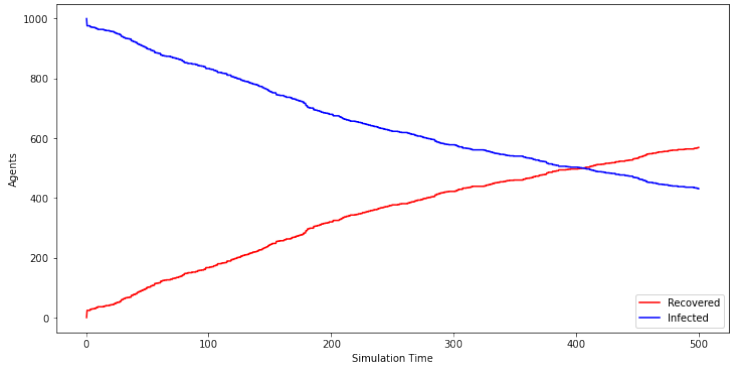
If we set an infection rate of 0.1 we obtain a rapidly evolution of the infections, as expected. It can be appreciated in Figure 9. We have that around ∼250 steps the infections stop because the uninfected pedestrians that remain are not close to infected people.



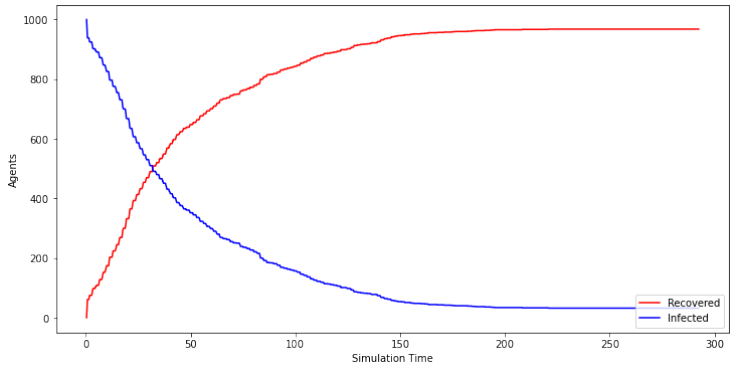
(a) 3 steps (b) 20 steps (c) 50 steps

(d) 100 steps (e) 230 steps

Figure 9: Evolution of the simulation for the static scenario with an infection rate of 0.1 (grid)



(a) Infection ratio of 0.01

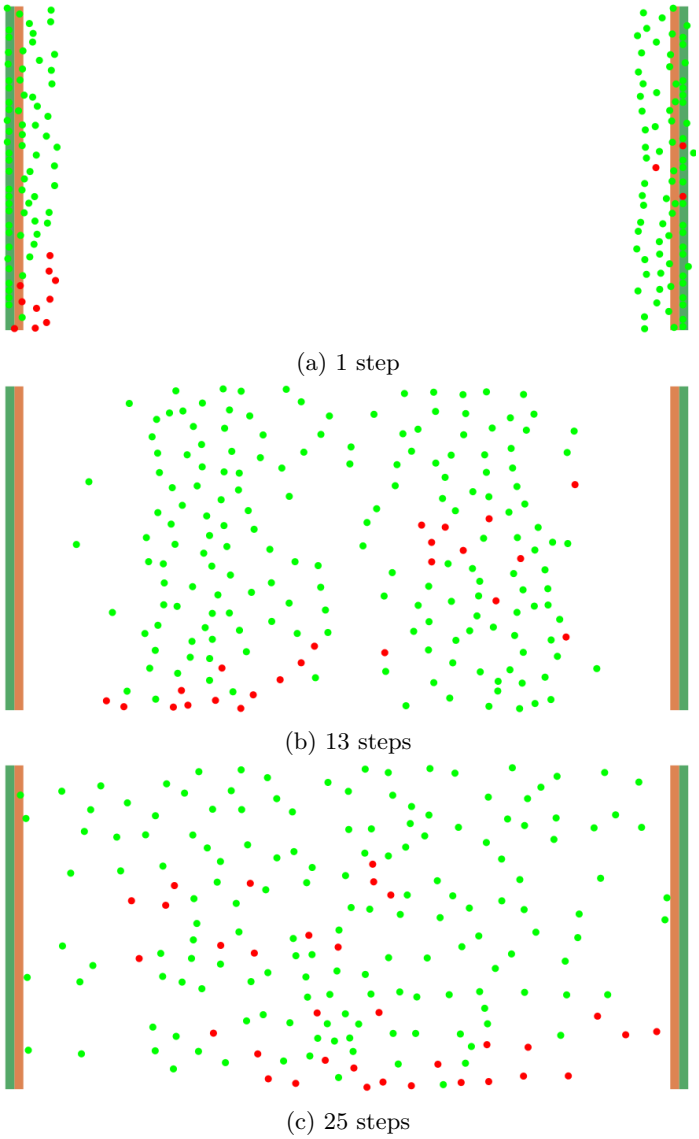


(b) Infection ratio of 0.1

Figure 10: Evolution of the simulation for the static scenario for different infection ratios (graphs)

In Figure 10 we can observe two plots with the evolution of the infections and susceptible persons in the time. We can see that, for an infection rate of 0.01, it takes ∼400 steps to have half of the population infected; and for an infection rate of 0.1, it takes ∼40 steps.

The third test that we have implemented, as the exercise sheet indicates, is the corridor with 200 people in a counter-flow. With 10 infections at start, an infection rate of 0.05 and an infection maximum distance of 1.0, we obtain that 36 pedestrians get infected and the evolution of Figure 11.



(a) 1 step

(b) 13 steps

(c) 25 steps

Figure 11: Evolution of the simulation for the corridor scenario

The given GUI for the visualization does not work for us. We have used the `utils.py` implementations in the notebook `Visualization` to obtain the plots shown.

In order to decouple the infection rate and the time step as far as possible we have modified the original `update` method from the `SIRGroupModel` class. Instead of executing the update of the pedestrians every time that the method is called, in our implementation we have to wait until the accumulation of differences between the current time and previous time (with modification) reach one. This allows us to work over periods of time instead of working directly with time.

As the last objective of this task, we are going to point possible extensions. Our proposals are the following ones:

- Include the concept of ventilation in some way, as a complementary to the infection rate or as a substitute.

It could be interesting because, while infection rate is something more related to a general behavior of a disease/virus/..., the ventilation is an specific aspect of a concrete scenario. It can be implemented adding the option of windows and using an specific algorithm in order to scan the state and update the infection condition/rate.

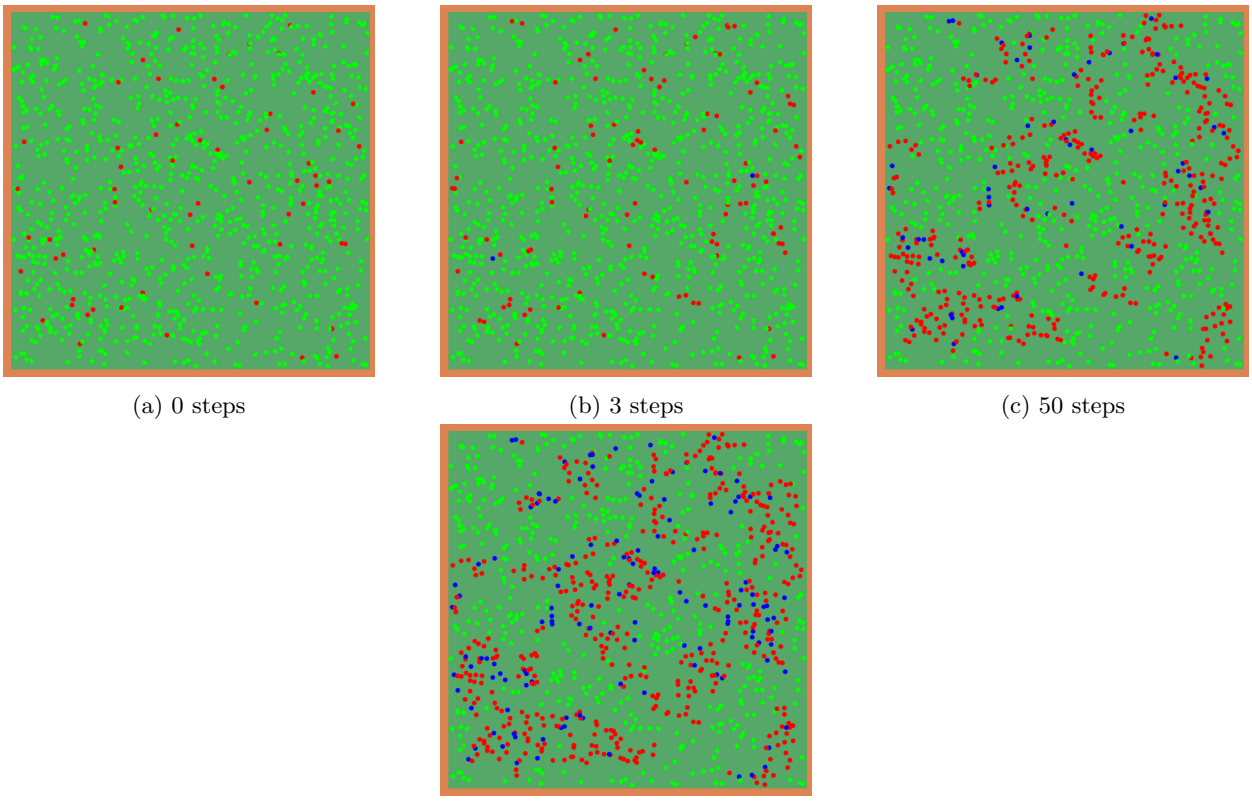
- Include the concepts of age or health condition for the pedestrians, in order to approximate in a better way the recovery/fatality rated and the spread speed of the disease. This could be used to accelerate or decelerate the recovery rate depending on the condition of each pedestrian.
- Add the ability to model a distribution of masks among pedestrians. This option would allow us to reduce the infection rate of specific pedestrians and see new dynamics caused by the use of masks.

Report on task 5, Analysis and visualization of results

For this final task 5 we have to add new features to Vadere, concretely, the "recovered" state to represent recovered persons (they cannot get re-infected nor infect susceptible persons) and a probability for an "infective" person to become "recovered" at every step. We have introduced a bit of this in the previous task, because we already implemented the features in the task 4, but making it optional.

In order to add the first feature, i.e. the "recovered" state, we have used the "revomed" named suggested by the exercise sheet and already included in Vadere. To add the probability to become recovered we have added to the `update` method from the `SIRGroupModel` class a condition that, given an infected pedestrian, it will become "removed" (in terms of the model) randomly, depending on a recovery rate specified in `AttributesSIRG`. This recovery features has been made optional using the `recoveryEnable` boolean attribute, also in `AttributesSIRG`. We have also include a first condition that checks if a pedestrian if removed. If it is, the method continues with the next one.

For visualizing the plots required for this report and shown following, we have properly modified the visualization given script. We have used, as mentioned before, our own Jupyter notebook.



(a) 0 steps

(b) 3 steps

(c) 50 steps

(d) 100 steps

Figure 12: Evolution of the simulation for the static scenario with recorevy

Test 1 - 1000 Static Pedestrians

For the first test we have to use the explained in the previous tasks scenario with 1000 pedestrians, starting with 10 infective and 990 susceptible. But now we have to incorporate the option recovery. For this test we have set the infection rate to 0.05 and the recovery rate to 0.005.

In Figure 12 we can see the evolution of the scenario for the first 100 steps. We can appreciate how the first recovered persons appear after 3 steps, being those people from the initial infection. The recovered people appears in blue color. Like without recovery, we can appreciate how the infection evolve in clusters, but now with "spontaneous" recoveries.

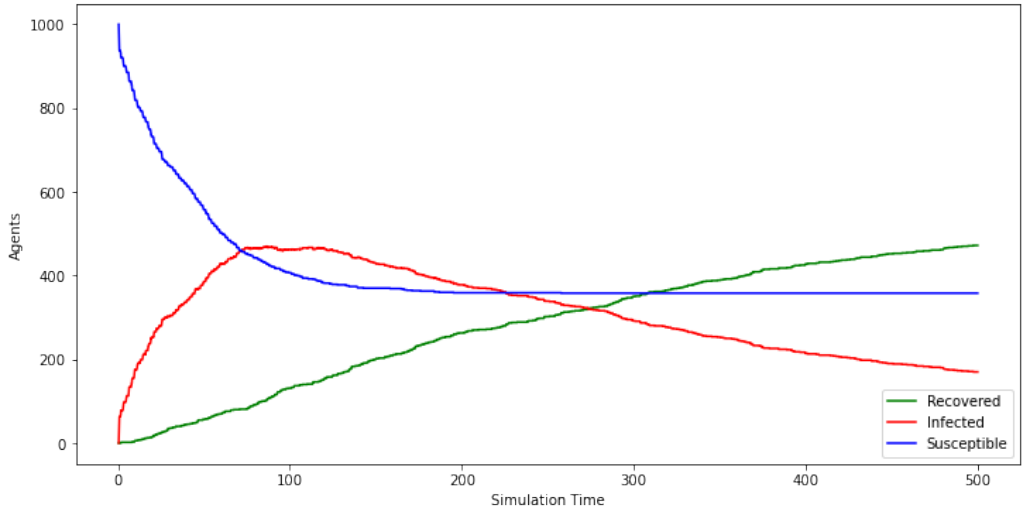


Figure 13: Evolution of Test 1 from Task 5

In Figure 13 we can see the evolution of the infections and recoveries in a plot. It can be appreciated how half of the population becomes infected at time ∼85 and how at some point around time 100 the infections start to decrease because the recovered people is not infecting any more. Due to this recovered people not infecting the susceptible people does not become infected, i.e., the propagation collapses.

Test 2 - 1000 Static Pedestrians and different infection rate and recovery rate values

In Test 1 we simulated with infection rate of 0.05 and recovery rate of 0.005. Now let us see what happens when both rates have the same value (0.05) and when the infection rate is lower (0.005) than the recovery rate (0.05).

When we have that both rates have the same 0.05 value (see Figure 14), recovered and susceptible persons collapse around time 100. This is occurs because the infected rapidly increase but around time 40-50 the number of recovered persons cut the infection transmission. If we have that both values are the same, at some time point the recovered persons stop the transmission and the number of infected persons in time will only depend on the scenario characteristics, i.e., the space between persons and the transmission distance.

For the case of a infection rate smaller (0.005) than the recovery rate 0.05 we have the results from Figure 15. We can see that the evolution is one that we can expect. If we have that infection rate is smaller that the recovery rate the initially infected persons will become recovered before infecting someone.

In order to strengthen our last argument, we have tested the same condition but with 100 infected persons at the beginning. As it can be seen in Figure 16, the recovered and susceptible rapidly collapse because the number of the infections also decrease fast.

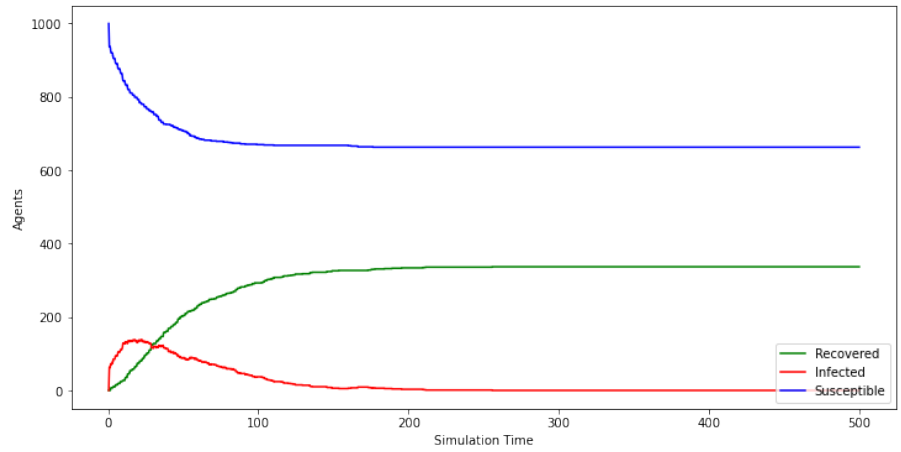


Figure 14: Evolution of Test 2 with equal infection rate and recovery rate (0.05)

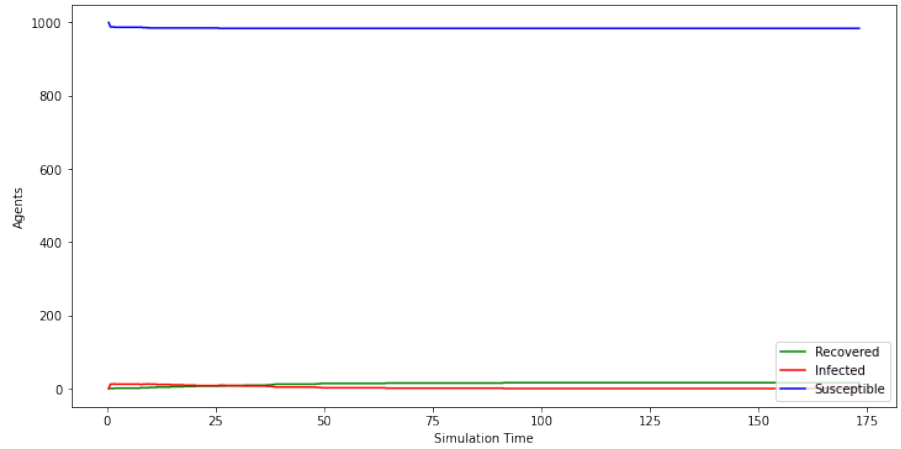


Figure 15: Evolution of Test 2 with 0.005 infection rate and 0.05 recovery rate (10 infected persons at the beginning)

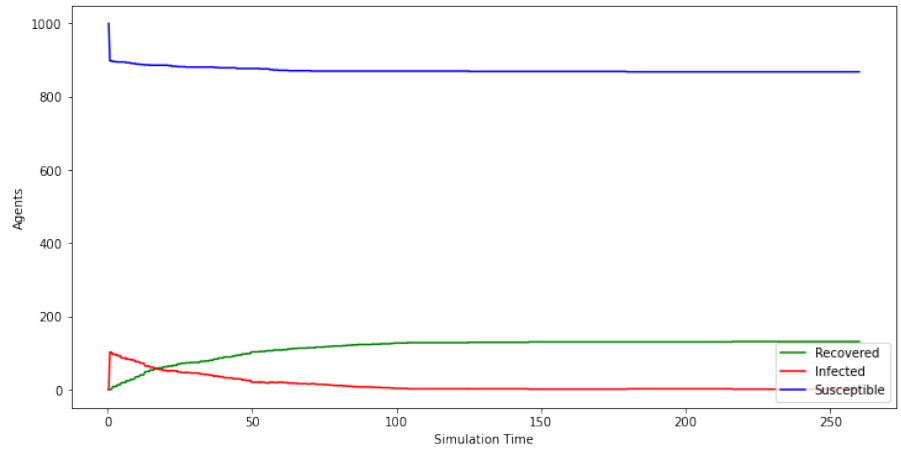


Figure 16: Evolution of Test 2 with 0.005 infection rate and 0.05 recovery rate (100 infected persons at the beginning)

Test 3 - Supermarket

In this final test we constructed an artificial Supermarket like the one given in the exercise sheet. The visualization and the structure of the artificial store can be seen in the Figure 17. We have placed 5 different sources on the entrance of the supermarket in order to make sure that the pedestrians are going to take different paths inside the supermarket.

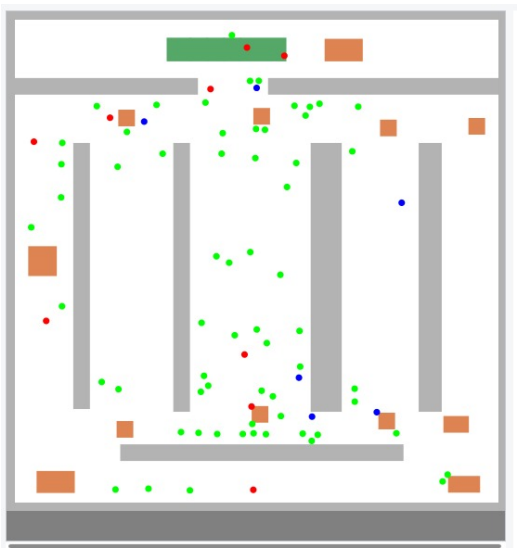


Figure 17: Artificial Supermarket Scenario

Following we are going to analyze the impact of `pedPotentialPersonalSpaceWidth` for different values. We ran the first simulation (see Figure 17) with an infection and recovery rate of 0.005. The final results can be seen in Figure 18. However, we are interested to see if we can bring the number of infected pedestrians lower, therefore in our next simulation we added some social distancing of 1.5 between the pedestrians.

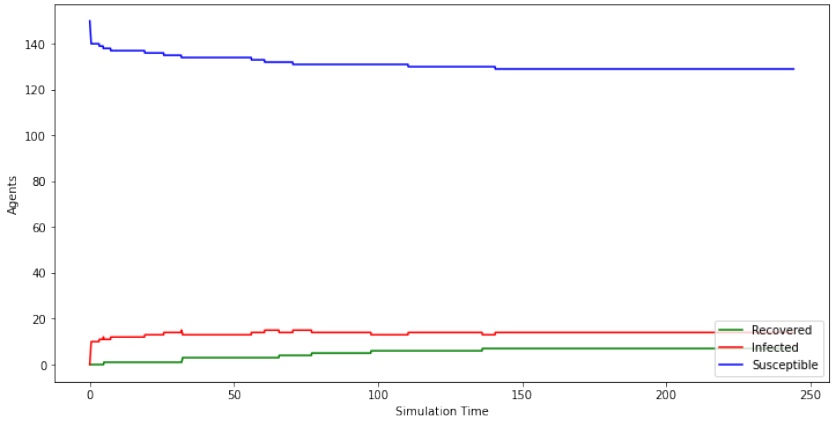


Figure 18: First supermarket simulation

Looking at the results of the simulation with the extra social distancing in Figure 19 we notice that not much really changed regarding the number of infected pedestrians. Like in the previous scenario it remained steady around roughly 18 pedestrians. We performed a couple more tests using different infection rates and social distancing in order to see how the number of infected pedestrians changes over time.

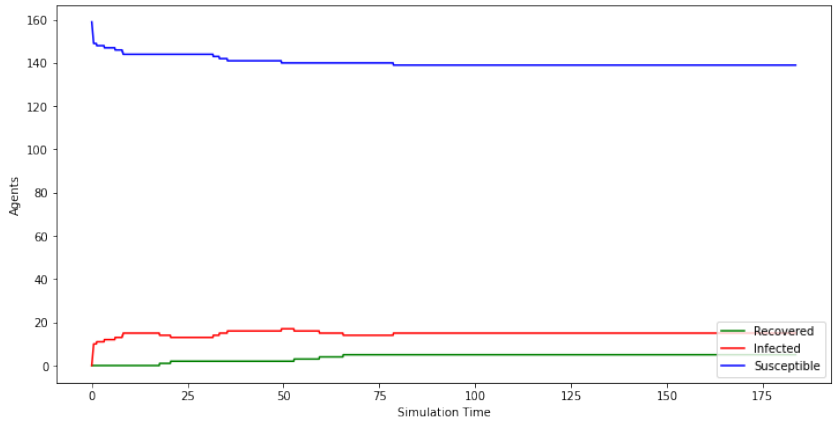


Figure 19: Simulation with increased social distance between pedestrians

For instance, in Figure 20 we increased the infection rate and the social distance. In this scenario we notice that interestingly the number of infected pedestrians in the supermarket is slightly decreasing over time in contrast to the previous simulations.

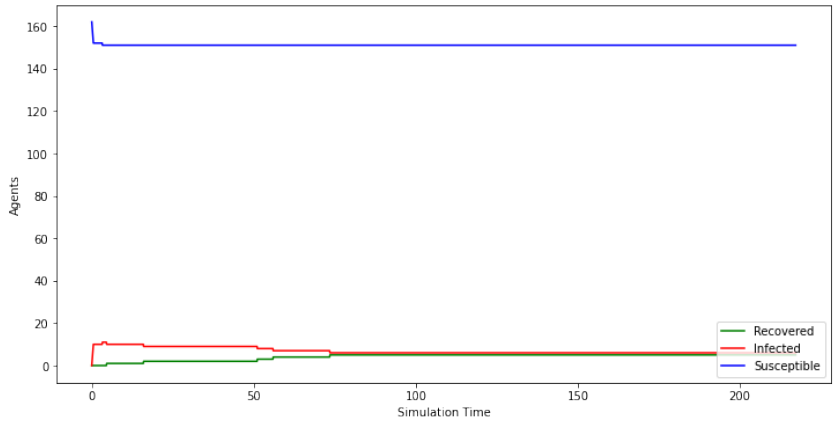


Figure 20: Simulation with higher infection rate and extra social distance between pedestrians

Furthermore, in Figure 21 we increased the infection rate but decreased the social distance between the pedestrians back to normal. The results show as expected that the number of infected pedestrians increasing drastically.

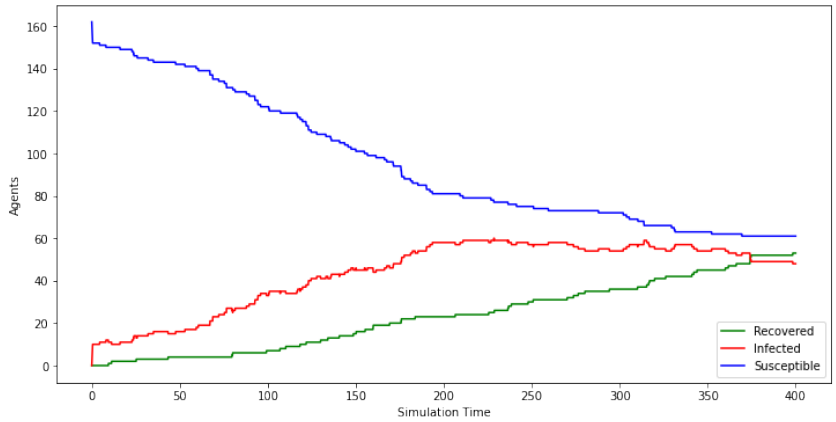


Figure 21: Simulation with higher infection rate and no extra social distance between pedestrians

Lastly, we tested if the number of infected people in the supermarket can get decreased if we reduce the number of pedestrians inside the supermarket at the same time. Therefore, we performed another test, this time with much lesser pedestrians and without extra social distance between them.

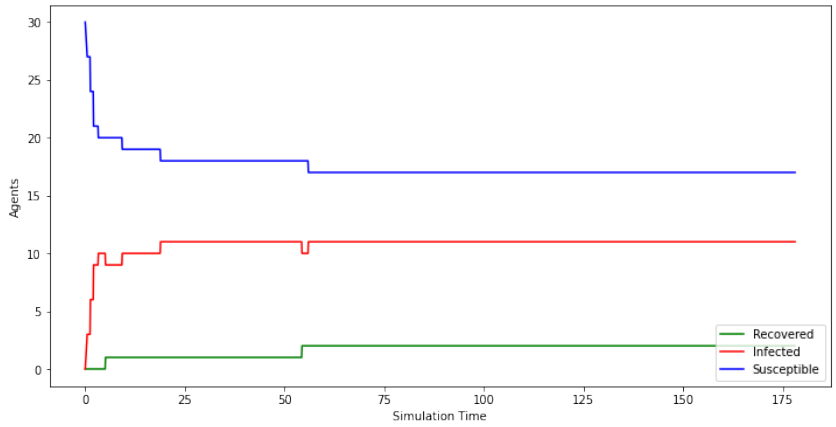


Figure 22: Final simulation with decreased number of pedestrian in the supermarket at the same time

Finally, we notice in the final simulation (see Figure 22) that the number of infected people in this simulation dropped but yet stayed relatively steady after a couple of time steps.

As a conclusion, we can state that increasing the social distance between pedestrians did not really have a big impact on the number of the infected pedestrians. However, restricting the number of pedestrians that can get into the supermarket at the same time managed to bring down the number of infected pedestrians but not so much as we were expecting.

References

[1] Michael J Seitz and Gerta Köster. “Natural discretization of pedestrian movement in continuous space”. In: *Physical Review E* 86.4 (2012), p. 046108.

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