Report for exercise 2 from group H

Tasks addressed:	5
Authors:	Ahmad Bin Qasim (03693345) Kaan Atukalp (03709123) Martin Meinel (03710370)
Last compiled:	2019–11–14
Source code:	https://gitlab.lrz.de/ga53rog/praktikum-ml-crowd

The work on tasks was divided in the following way:

Ahmad Bin Qasim (03693345)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Kaan Atukalp (03709123)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%
Martin Meinel (03710370)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
	Task 5	33%

Report on task 1, Setting up the Vadere environment

We successfully downloaded the vadere software and ran it. We simulated the RiMEA scenarios 1 (straight line) and 6 (corner) as well as the "chicken test". For all of these simulations we used the Optimal Steps Model (OSM). In the following paragraph we compare the vadere simulation of these scenarios with our own cellular automoton simulation of the first exercise.

- RiMEA scenario 1 (straight line):
  We set up the vadere simulation of the first RiMEA scenario with a mean speed of 1.33, a minimum speed of 1.25 and a maximum speed of 1.44. Moreover we set up the speed standard deviation to 0.1. The pedestrian took with 29.999 seconds almost 30 seconds. This is very close to our result we achieved with our own implemented cellular automoton in the first exercise (29.67s). One difference of vadere to our simulation is that the RiMEA scenarios are already set up so we only configured the speed, but the setup of the topography was already done. Moreover the pedestrian in our cellular automoton walks straight because we use cells with a shape of a rectangular. In the vadere simulation the pedestrian walks a little bit up and down. We think that the reason for this behaviour is that the grid consists of hexagonal cells. So the resolution of the grid is very high. In Figure 1 you can see the simulated scenario and recognize how the pedestrian walked up and down and not totally straight to the target.

Figure 1: RiMEA Scenario 1 with OSM

- RiMEA Scenario 6 (corner):
  We set up the RiMEA scenario 6 with a mean speed of 1.34. Moreover we set the minimum speed to 0.5 and the maximal speed ot 2.2. The speed standard deviation was set to 0.26. We choose the RiMEA scenario from a list of pre-defined scenarios which vadere offers. So we didn't need to set up the topography by ourselves. The following table shows for every pedestrian the evacuation time.

pedestrianID	evacuationTime
1	16.8
2	19.2
3	21.2
4	17.2
5	20.0
6	10.4
7	24.4
8	20.2
9	20.8
10	24.0
11	17.6
12	33.2
13	16.8
14	14.0
15	12.8
16	27.2
17	20.4
18	25.6
19	11.2
20	15.6

  In comparison to our simulation from the first exercise we did not record the time every pedestrian needs to reach the target. We recorded only the time that all pedestrians took to reach the target around the corner. Furthermore vadere shows the single paths of the pedestrians to the target, while in our cellular automoton it was only possible to see which cells were not part of any path to the target. Figure 2 shows

the RiMEA 6 sceanrio simulated with the Optimal Steps Model. The picture shows which path the single pedestrians take to reach the target.

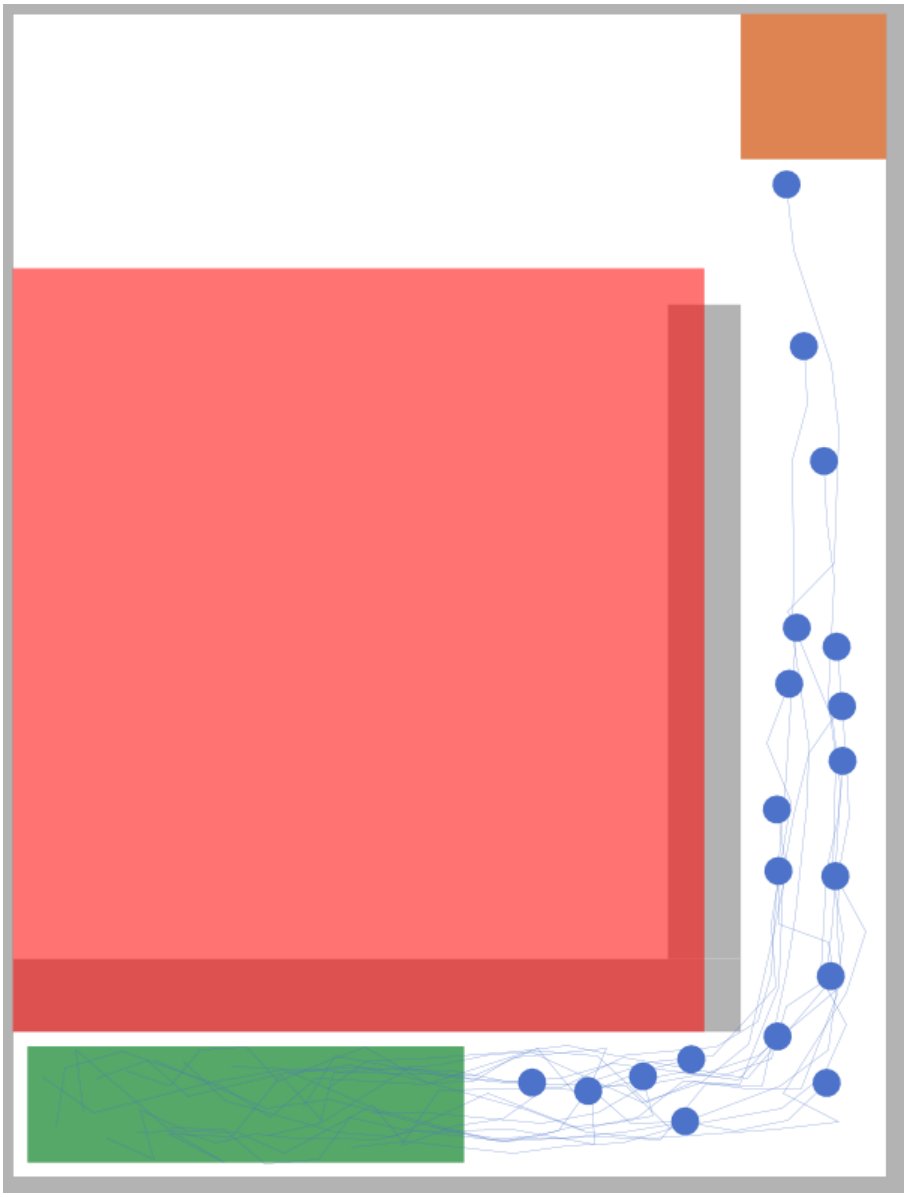

Figure 2: RiMEA scenario 6 with OSM

- "Chicken Test" scenario:
  At first some of the pedestrians do not pass by the obstacle left or right. They walk some steps in the wrong direction in direction of the obstacle and wait until the the most pedestrians have passed the obstacle. Afterwards they turn around and go back on the right path and reach the target as well. This behaviour is caused by the big crowd. In our simulation we did not have such a behaviour. The reason for this could be that we set less pedestrians in our chickentest scenario, so that every pedestrian can reach the target on its own path and is not blocked by any other pedestrian. Our simulation could probably not handle the big amount of pedestrians which vadere uses to simulate the scenario. This scenario was pre-configured as well. In Figure 3 the pedestrians which first take the wrong path and do not pass by the obstacle at the first try can be seen.

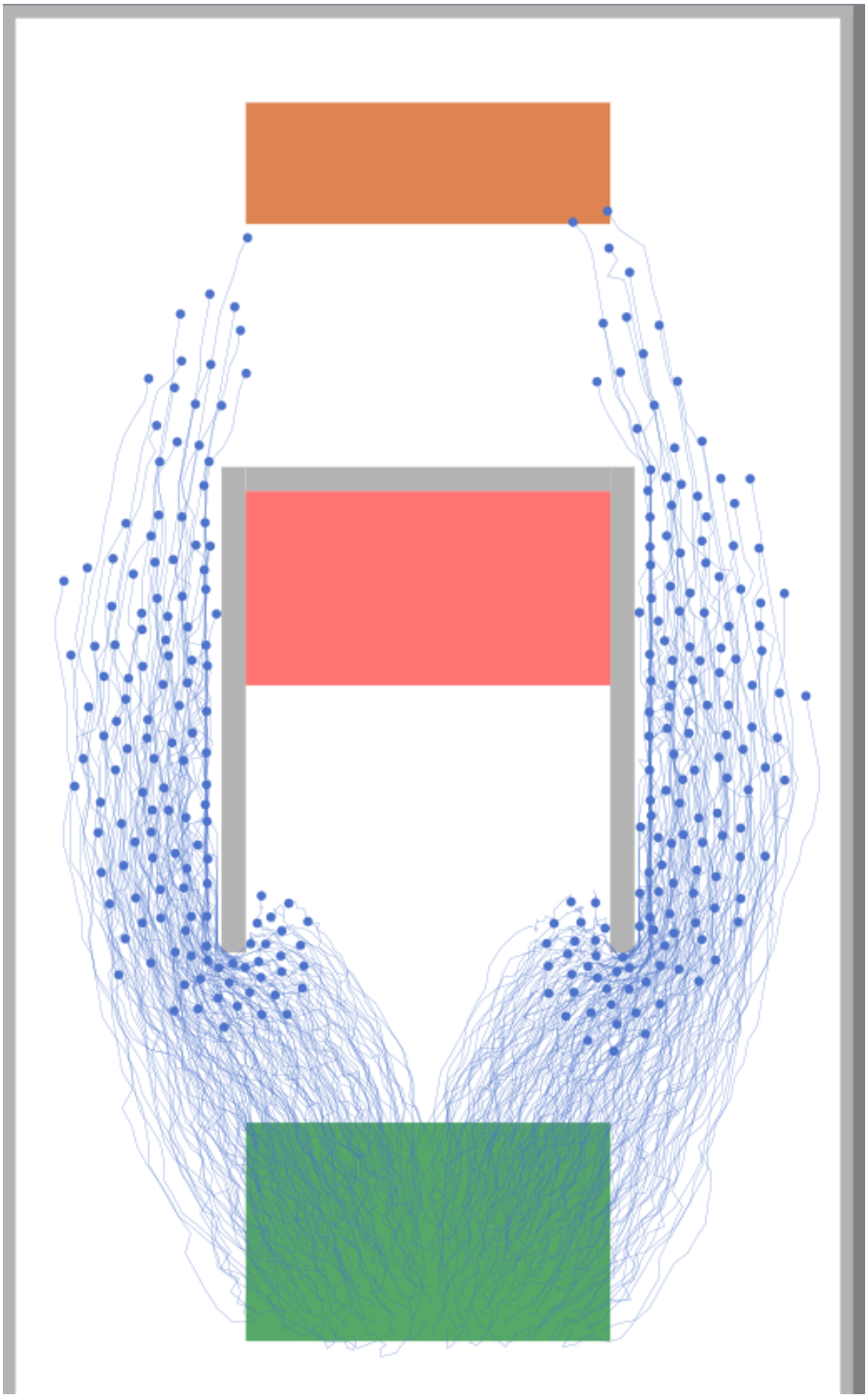

Figure 3: Chickentest with OSM

- General differences and observations:
  In general our cellular automoton passes the tests as well as the vadere simulation does. Furthermore, the GUI of vadere lets the user adjust much more things of the simulation. We use the command line to set up different scenarios and adjust them, but we do not offer as many options as vadere does. One example is that the user can set up a mean, minimum, maximum and standard deviation of the pedestrian speed. In our case the pedestrians have a constant speed.

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

For this task we used the same scenarios (RiMEA scenario 1, RiMEA scenario 6 and chickentest scenario) but with different models. Here we use the Social Force Model (SFM) and the Gradient Navigation Model (GNM) and compare them between eahch other.
At first the Social Force Model is used to simulate the three scenarios.

- RiMEA Scenario 1:
  We use the same configurations as for the OSM model. In comparison to the OSM model there cannot be seen any differences. The pedestrian takes 30 seconds to reach the target, which is exactly the same time as the pedestrian needs with the Optimal Steps Model. In Figure 4 can be seen how the pedestrian walked from the source to the target. Besides, it shows that the pedestrian walks slightly up and down on its path as it does with the OSM.

Figure 4: RiMEA scenario 1 with SFM

- RiMEA Scenario 6:
  The RiMEA scenario runs smoother with SFM than the OSM. The distance between the single pedestrians is smaller than for the OSM. The following table containing all pedestrians and the time they take to reach the target, shows that the pedestrians reach faster the target than in the OSM. The reason for that might be that the distance between the pedestrians is smaller.

pedestrianID	evacuationTime
1	12.0
2	23.4
3	14.8
4	18.4
5	20.8
6	14.0
7	12.0
8	19.6
9	15.6
10	18.0
11	12.4
12	18.8
13	18.8
14	13.6
15	16.0
16	14.0
17	22.0
18	14.8
19	23.6
20	13.2

  Figure 5 shows that the distance between the pedestrians is smaller than before. It can be seen from that picture that the radius with which the pedestrians walk around the corner is smaller for the SFM than for the OSM. Figure 6 shows the corner situation and makes it clearer that all the pedestrians have less distance towards each other.

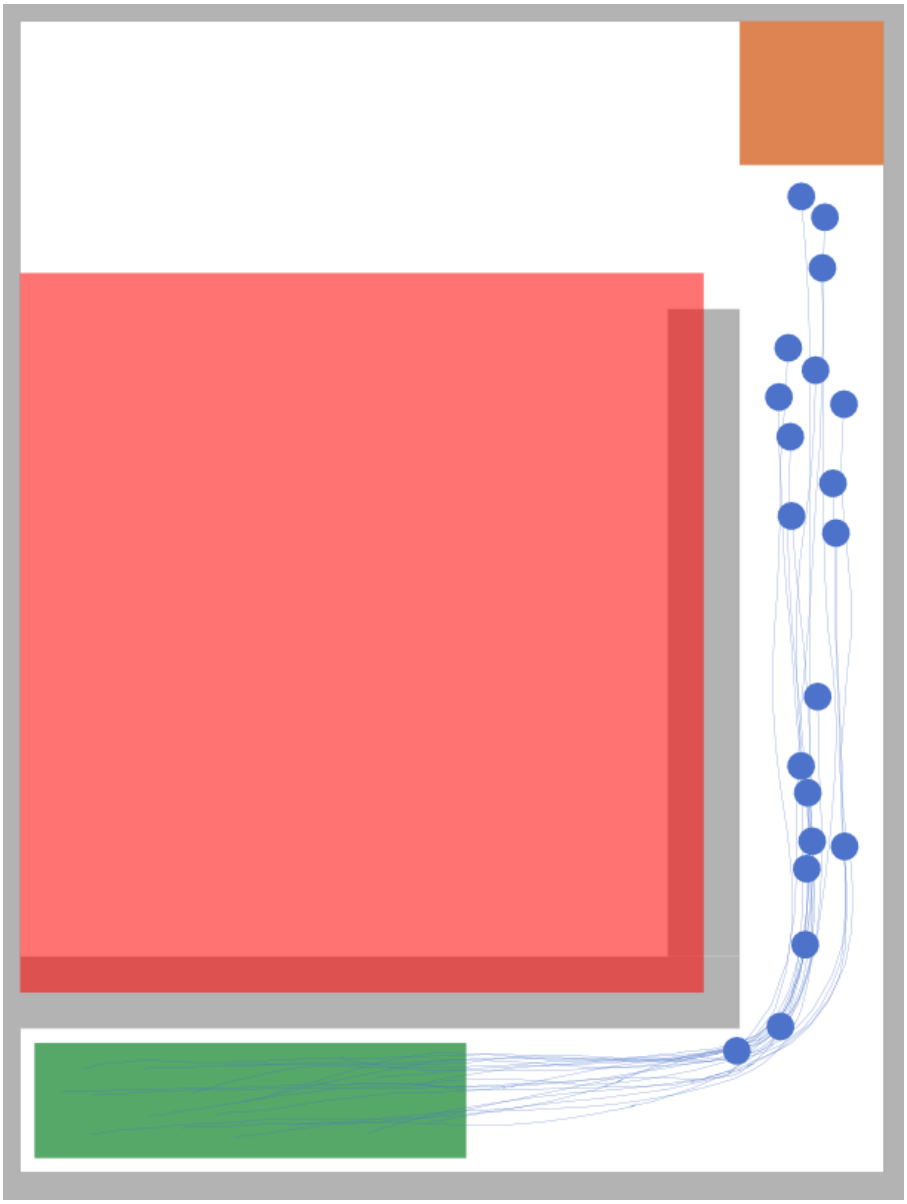

Figure 5: RiMEA 6 scenario with SFM

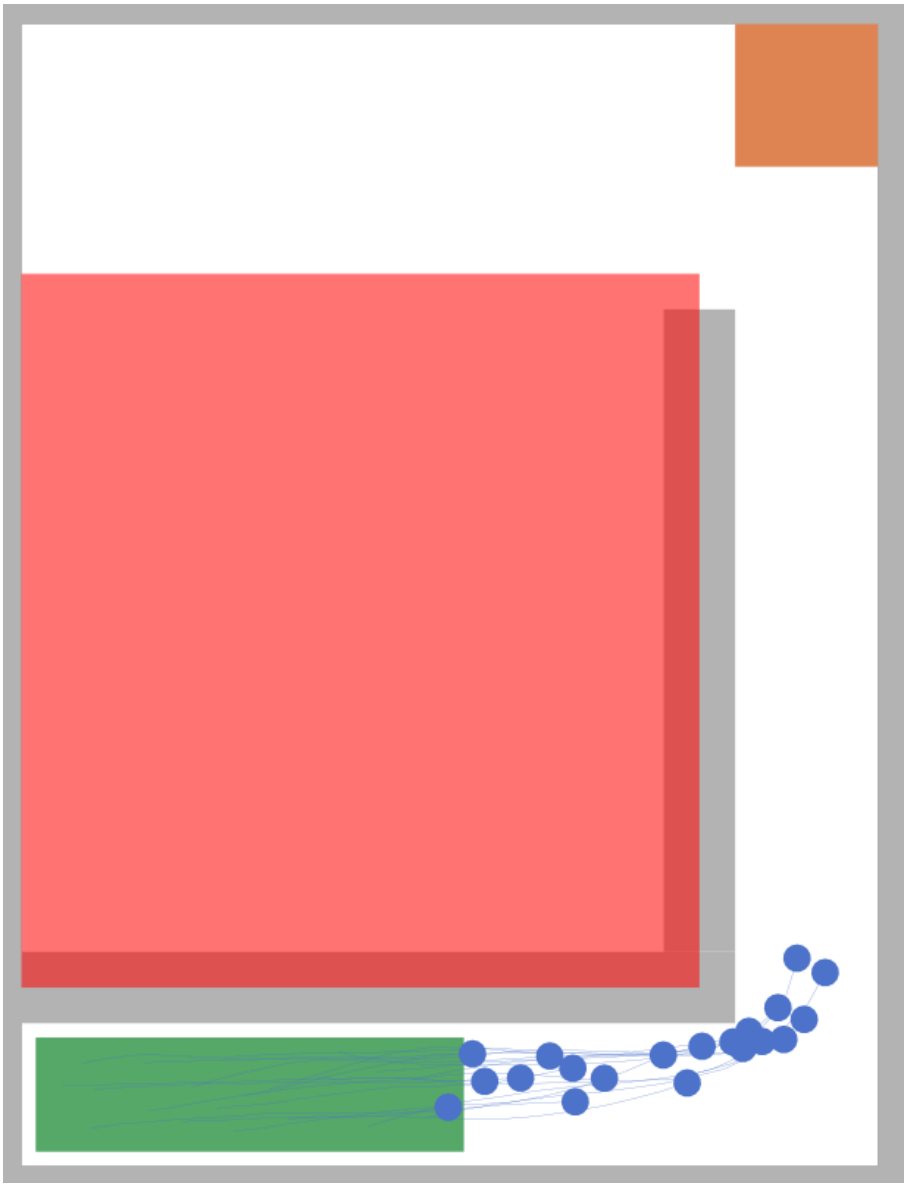

Figure 6: RiMEA scenario showing the corner with SFM

- Chickentest:
  The number of pedestrians is set to 300 to make it possible to compare the OSM to the SFM. If the Social Force Model is used the pedestrians try not not to avoid other pedestrians as much as for the OSM. Consequently, less people leave the right path and walk in the wrong direction towards the obstacle. So less people get stuck for a while in the region of the obstacle, even though we observed a problem that some pedestrians overlap with the obstacle and jump over it. Figure 7 shows that much less pedestrians get stuck at the inner side of the obstacle than before. Furthermore it shows the encountered problem that pedestrians jump over the obstacle. This can be seen from the paths leading over the obstacle.

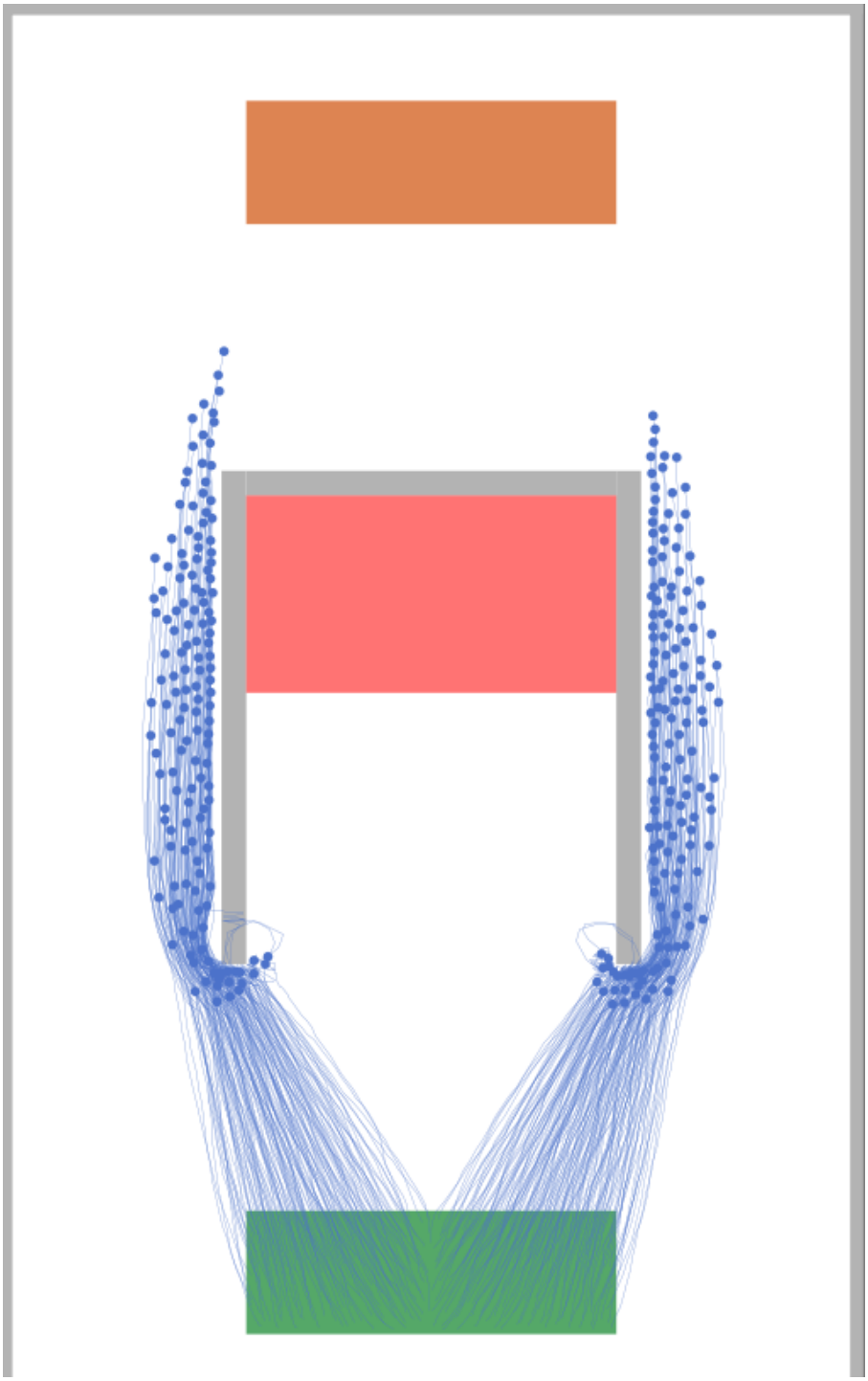

Figure 7: Chickentest with SFM

After using the Social Force Model for simulating the three scenarios and comparing the results to the results from OSM the Gradient Navigation Model is used in the following paragraph.

- RiMEA 1 Scenario:
  There are no differences to the other models. The pedestrian takes 30 seconds to reach the target as for the other two models as well. Figure 8 shows that the pedestrian walks as for the previous models slightly not straight which supports our claim that the shape of the cells of the grid is responsible for that, since the model cannot be responsible for this.

Figure 8: RiMEA scenario 1 with GNM

- RiMEA Scenario 6:
  The difference to the previous models is that the pedestrians try to walk along the wall around the corner. So they are stick at the wall. The following table shows that the pedestrians using this model take the most time to reach the target in comparison to the other models. The reason for this is that the pedestrians are stick to the wall and do not want to go around the corner in a bigger radius. This behaviour can be seen in Figure 9. In comparison to the previous models it seems unnaturally.

pedestrianID	evacuationTime
1	22.0
2	33.6
3	34.0
4	21.2
5	34.8
6	31.6
7	18.4
8	33.2
9	16.0
10	20.0
11	22.4
12	32.4
13	35.6
14	20.8
15	18.0
16	20.4
17	35.2
18	17.2
19	31.2
20	21.6

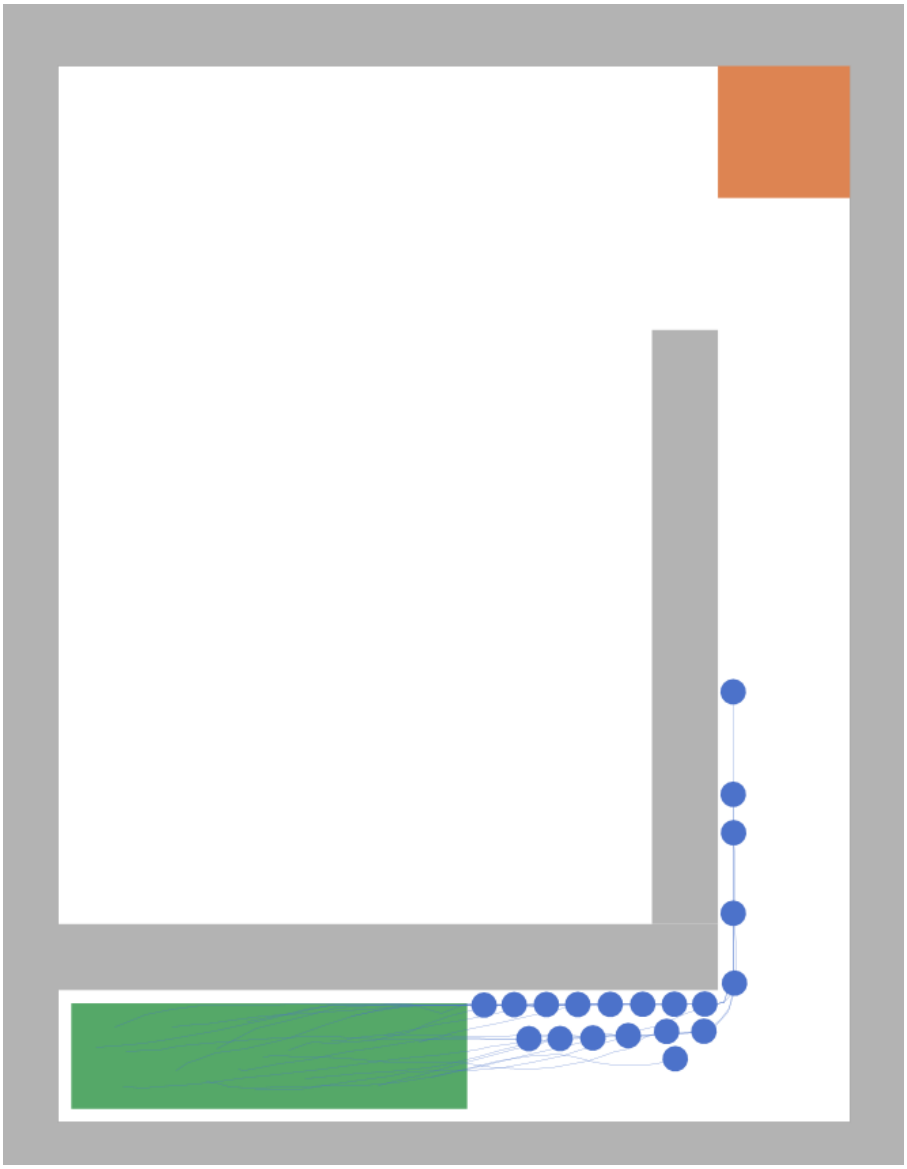

Figure 9: RiMEA scenario 6 with GNM

- Chickentest:
  For the GNM model simulated with the chickentest scenario the pedestrians gather a lot at the corners of the obstacle, but do not enter the area surrounded by the obstacle. This is different to the other used models. Besides, the problem of the SFM does not occur anymore here. There are no pedestrians which jump over the obstacle from the inner side of the obstacle to the outer side of the obstacle. Furthermore the pedestrians take some more time to reach the target because they do not walk with a big radius around the obstacle. They try to pass by slightly the obstacle and walk very close to it. Another remarkable difference is that the computational work for the GNM is heavy. It takes more compuational work than for the OSM and SFM. The phenomen that the pedestrians gather at the corners of the obstacle and slightly walk around the corner can be seen in Figure 10.

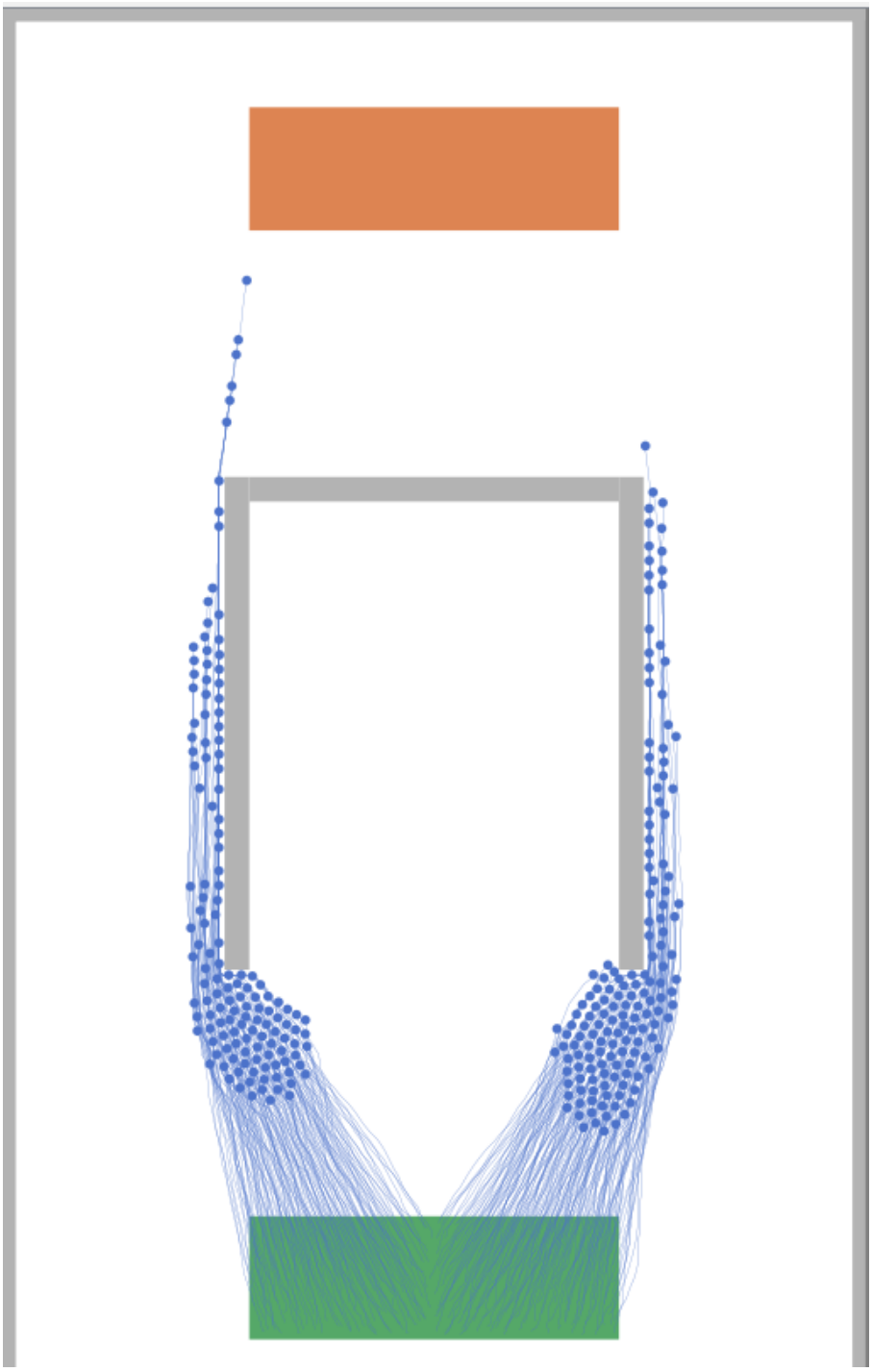

Figure 10: Chickentest with GMN

Report on task 3, Using console interface from Vadere

The new pedestrian needs only 9.6 seconds to reach the target since it starts walking at the corner. We set up the pedestrian at the position (12.3, 1.8) with the targetId of 1. We used the json library of Python to read the scenario file of the "Corner Scenario" and adjusted it so that the additional pedestrian is added to the corner at position (12.3,1.8). We gave the newly generated JSON file's path as an argument on the command line and ran Vadere from the console as well. The output can be seen below where the first column is the pedestrian ID and the second column is the arrival time at the target. The pedestrian of ID 1 is the one that

was added to the corner.

pedestrianID	evacuationTime
1	9.6
2	22.8
3	15.6
4	14
5	12.
6	19.6
7	22
8	16.4
9	14.4
10	21.6
11	17.2
12	14.8
13	18.4
14	13.2
15	11.2
16	19.6
17	26
18	14.8
19	20.4

Report on task 4, Analyzing output of models

1. In the given example, the true data is generated by using the formula:

$$(1 + \frac{sin(5\theta)}{3}) * \exp(1_j * (angle + dt))$$

   where dt is a constant which drives the change in angle at each timestep. This expression results in a complex expression. One dimension of the true data is the real part of this expression and the second dimension is the imaginary part. The modeled data is simply the true data with a normally distributed noise added to it with a standard deviation equal to **model_error** value. A normally distributed noise with a standard deviation equal to **true_error** is added to the true data as well.

2. The few simplifications are given here by:

   - The algorithm is implemented for only a single agent
   - The inverse of Z_k is calculated using the pseudo inverse which is not mentioned in the paper.
   - The implementation of maximum likelihood is not consistent with the algorithm provided in the paper

3. The entropy increases logarithmically as the `model_error` parameter increases (see Figure 11). Since $\hat{M}$ has a linear relationship with `model_error` and $\hat{M}$ is fed in into the logarithm function to calculate the entropy, the entropy shows a logarithmic correlation with `model_error`. Thus, the bigger the model error, the more the model differs from the truth; but the rate of difference diminishes.

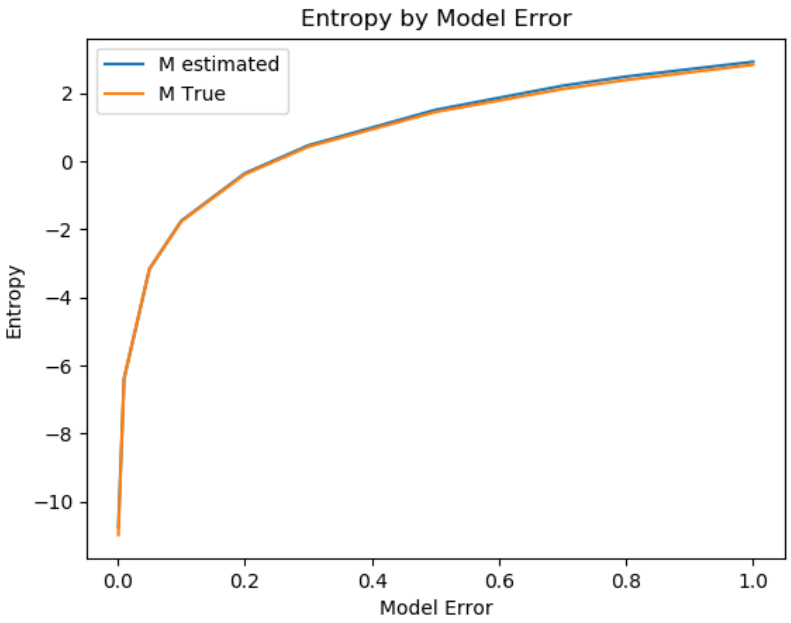

Figure 11: Change of entropy by model error

Report on task 5, Tests

We setup the bottleneck scenario, as it is recommended in the exercise sheet. The dimensions of the source, the target and the obstacles as well as the distances between them are calibrated accordingly.

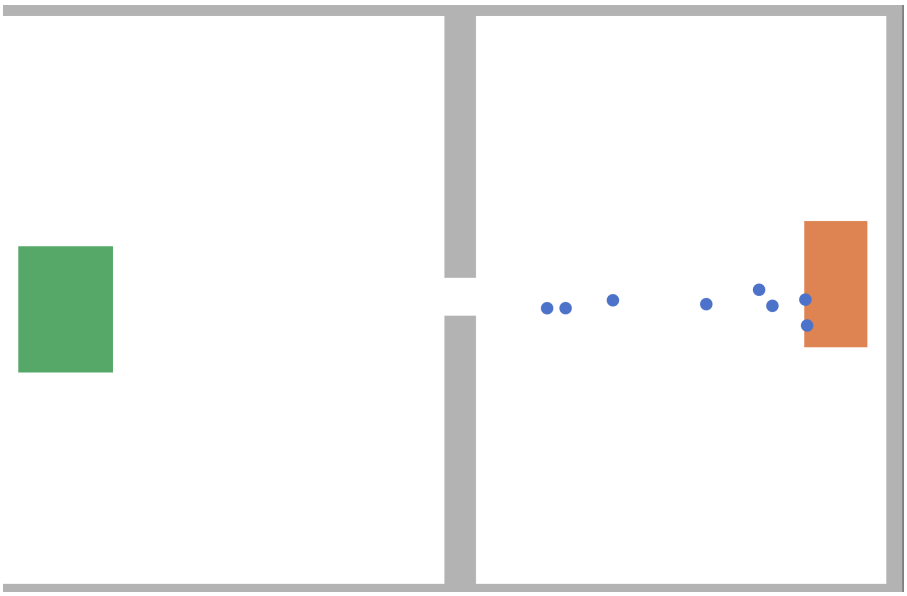

Figure 12: The bottleneck scenario being simulated using the OSM model and 25 pedestrians

1. We simulate the bottleneck that we setup using the OSM model for a set of pedestrians i.e. {15, 20, 25, 30}. We save the output files (postvis.traj and velocities.txt) as the true data in the following folders bottleneck_OSM_15, bottleneck_OSM_20, bottleneck_OSM_25 and bottleneck_OSM_30 respectively. We also save the simulation scenario file into a separate folder so that it can later be edited programmatically.

We repeated this process for the GNM model as well so that during the later phase of this task, GNM can be used as the source

2. As a second step, we parse the postvis.traj and velocities.txt files of the "true" model (OSM or GNM in case of the reverse case) using the **parse_trajectory** function and the obtain the "predicted" coordinates and velocities. The following approach is used to achieve this:

   - The input to this function is the path to the output files (postvis.traj and velocities.txt) of true model and it outputs coordinates and velocities of each pedestrian over a series of timesteps, which are considered as the true data that is stored in the **zk** variable in the implementation.
   - In order to obtain the "simulated" model output to run the Guy et. al. algorithm, we create a function titled **f_model**. The input to this function are the true model's pedestrian coordinates and the velocities at a given timestep.
   - This true data is used to initialize dynamic elements within the bottleneck scnenario file created in the previous step. The bottleneck scenario file is parsed using the json python library and then the dynamic elements are initialized within a loop for each pedestrian in the true data. The modified scenario file is saved at a temporary location. The modified scenario file uses the "prediction" model (GNM or OSM in the reverse case).
   - Then we use the python os library to execute the vadere-console.jar to run the modified scenario file for 5 timesteps.
   - The output generated from the prediction model is parsed using the **parse_trajectory** function and the predicted coordinates and velocities are returned as the output of **f_model** function.

   Lastly, we have to mention the approach that we took for calculating the velocity. We used a built-in vadere processor for obtaining the true data velocities.txt file. This file contains the simTime and endTime-PID1 attributes for each timestep for every pedestrian which we use to determine the time duration of each timestep. This file, also contains the startX-PID1, startY-PID1, endX-PID1 and endY-PID1 attributes, which we use to calculate the distance covered in the corresponding time duration. Using the time duration of each timestep and the corresponding distance covered by each pedestrian in that time duration, we calculate the velocity's X and Y component.

3. To estimate the entropy of GNM with respect to OSM, we made the following changes to the implementation which was provided already:

   - Changing the maximum likelihood implementation, which was implemented for only a single agent in the original implementation. In order to achieve that we followed the maximum likelihood algorithm provided in the Guy et. al. paper.
   - The observation function H is set as the identity function, since in our case, model state x and the true data z are the same.
   - We modify the implementation to use the true data obtained from the true model output and the predicted data obtained from the prediction model output.
   - A normally distributed noise is added to the true and the predicted model data.
   - The whole code is made modular and dynamic so that it can be run with different models set as true/prediction.

   The parameters that we use for calculating the entropy between the true and prediction models are:

Parameter	Value	Description
true_error	1e-4	The standard dev of the normally distributed error added to the true data
model_error	1e-3	The standard dev of the normally distributed error added to the predicted data
NT	30	The total number of timesteps from the true data
m	10	The number of ensembles
d	4	The dimension of the true/predicted data.
N_ITER	5	The number of iterations for which the algorithm is executed

4. As mentioned during the first step, we generate the true and the predicted data for different number of pedestrians in the bottleneck scenario. We plot the entropy results obtained using OSM model as the source of true data and comparing it to GNM as prediction model.

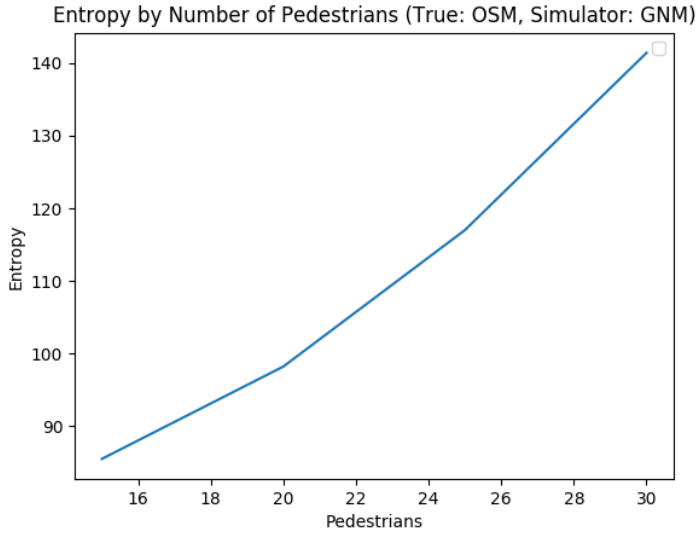

Figure 13: The entropy increases linearly with the increase in the of pedestrians

The linear increase in the entropy is due to the fact that, as the number of pedestrians increase, so does the interactions between the pedestrians and also the interactions of the pedestrians with obstacles. As the OSM and GNM model direct the pedestrian behavior differently, this results in less similarity in the pedestrian trajectories and the velocities throughout over a series of timesteps.

5. We implemented two simple command line arguments which can be adjusted to interchange between using OSM as the true data source and the GNM as prediction model and vice versa. All the paths are adjusted accordingly e.g. paths for the true data and the scenario files.

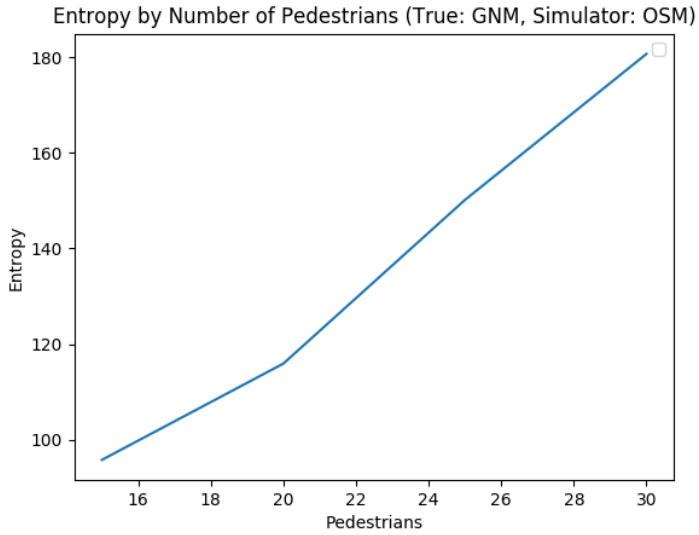

Figure 14: The entropy increases linearly with the increase in the of pedestrians