Draft Report Exercise 2

Task 1:

Corridor straight line test: speed (mean speed) = 1.33, std.dev = 0.1, min. speed=1.25

Max. speed =1.4166 Total time = 29,999 = 30

Differences to our simulation. We use rectangular cells while valdere use cells shaped hexagonal. The pedestrian walks up and down, in our simulation it walks more straight

The resolution of the grid is very high.

Corner Test: mean speed = 1.34; min. speed =0.5; max. speed = 2.2; speed std.dev = 0.26

pedestrianId evacuationTime-PID2

1 16.800000000000004

2 19.199999999999996

3 21.19999999999999

4 17.200000000000003

5 19.999999999999993

6 10.400000000000004

7 24.399999999999977

8 19.199999999999996

9 20.79999999999999

10 23.99999999999998

11 17.6

12 33.199999999999946

13 16.800000000000004

14 14.000000000000007

15 12.800000000000006

16 27.199999999999967

17 20.39999999999999

18 25.599999999999973

19 11.200000000000005

20 15.600000000000009

We didn’t record the times for each pedestrian reaching the target. We recorded only the time in which everyone reached the target. Besides, in vadere you can track the single path of every pedestrian to the target, while in our simulation you can see which cells have not been used to reach the target.

Chicken Test:

Some of the pedestrians first walk some steps in the direction of the obstacle and wait until the majority of the crowd has left. Afterwards they go back on the right path and reach the target as well. This behavior is caused by the big crowd. In our simulation we did not have such a behavior. One reason for that could be that we set less pedestrians in our simulation, 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 this big number of pedestrians.

In general our simulation passes the test as well as the vadere simulation does. Furthermore, the GUI of vadere lets the user to adjust much more things of the simulation. We use the command line to set up the different scenarios and adjust them. But we don’T offer as many options as vadere does. One example is that you can set up a minimum, maximum and a standard deviation of the pedestrian speed. In our case the pedestrians have a constant speed.

Task 2:

For this task we use the SFM (Social Force model) instead of the Optimal Steps Model.

Corner Test: The pedestrians walk smoother than before. The distance between the pedestrians is smaller than before.

1 12.000000000000005

2 23.99999999999998

3 14.800000000000008

4 18.4

5 20.79999999999999

6 14.000000000000007

7 12.000000000000005

8 19.599999999999994

9 15.600000000000009

10 18.0

11 12.400000000000006

12 18.799999999999997

13 18.799999999999997

14 13.600000000000007

15 16.000000000000007

16 14.000000000000007

17 21.999999999999986

18 14.800000000000008

19 23.59999999999998

20 13.200000000000006

Pathway Test:

The pedestrian takes the same time (30 seconds) as for the OSM.

Chicken Test:

The number of pedestrians for OSM and SFM is 300 to be able to compare both models.

The pedestrians do not try to avoid other pedestrians as much as in the OSM. Consequently, less people walk in the wrong direction and get stuck in the corner for a while.

One problem of this SFM model is that some pedestrians overlap with the obstacles and jump over the obstacle.

GNM:

Chicken Test:

They do not jump over obstacles. The pedestrians gather at the corners of the obstacle a lot, but they did not get into the area of the obstacle. This was different for the other used models.

They take some time to reach the target because they do not walk far around the corner, every pedestrian wants to walk slightly around the corner. The computational work for the GNM test is heavy. It is more computational work than for the other models.

Corner Test:

The pedestrians try to walk along the wall. So they stick at the corner walls.

Pathway Test:

The pedestrians take the same time as before (30 seconds). There are no differences to the other Models.

Task 3:

The new pedestrian needs only 10 seconds to reach the target because it starts walking in the corner. We set up the pedestrian at the position (12.3, 1.8) with the targetId of one. We used the json library of python to read in the scenario file of the corner scenario and adjust so that the additional pedestrian is added in the corner at position (12.3,1.8). This new generated file we gave as an argument on the command line and run vadere from console as well. The output file can be seen here.

9.600000000000003

2 22.799999999999983

3 15.600000000000009

4 14.000000000000007

5 12.400000000000006

6 19.599999999999994

7 21.999999999999986

8 16.400000000000006

9 14.400000000000007

10 21.599999999999987

11 17.200000000000003

12 14.800000000000008

13 18.4

14 13.200000000000006

15 11.200000000000005

16 19.599999999999994

17 25.99999999999997

18 14.800000000000008

19 20.39999999999999

Task 4

Set model\_error to 1:

entropy(M estimated) 2.926796276169226

entropy(M true) 2.8378770664093453

Set model error to 1e-2

entropy(M estimated) -6.3852847596088305

entropy(M true) -6.3724633055668365

If the model error decreases the entropy decreases as well. The other way round if the model error increases the entropy increases too.

Set model error to 1e-3:

entropy(M estimated) -10.734192758886685

entropy(M true) -10.977633491554927

The entropy increases/decreases with log2.

1. The true model is generated by using the formula xc1 = (1 + np.sin(5 \* angle) / 3) \* np.exp(1j \* (angle + data\_t)), while the model data is generated by using the true model data and adding some random noise to it.
2. The algorithm uses only one agent and the maximum likelihood is not calculated per agent also the inverse of the sigmoid matrix is calculated using the pseudo inverse which is not mentioned in the paper.
3. The entropy increases logarithmically even for linear input. (see picture). The bigger

the error is the more the model differs from the truth. The entropy formula is the reason for a logarithmic change

Task 5