Report for exercise 1 from group H

Tasks addressed:	5
Authors:	Ahmad Bin Qasim (03693345) Kaan Atukalp (03709123) Martin Meinel(03710370)
Last compiled:	2019–11–02
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 modeling environment

The modeling environment was successfully implemented using Python. The targets, the pedestrians, the obstacles, and the path a pedestrian has taken are represented with the colors red, blue, black, and light grey respectively. Once a pedestrian reaches the target, it becomes light pink and other pedestrians become able to enter the position but this functionality can be turned off using a command line argument. For some tests in Task 5, inherited classes were implemented for ease of initialization. Configurations such as pedestrian, target and obstacle locations can be set through command-line arguments and the code need not be changed every time. The file gui.py implements a rudimentary GUI for executing different tasks 1. We used the tkinter python library for the GUI.

The command line arguments which can be tweaked in the implementation are as follow:

Arguments	Possible values	Description
grid_size_x	0-1000000	The size of grid in horizontal direction
grid_size_y	0-1000000	The size of grid in vertical direction
p_locs	array of tuples	The coordinates of the pedestrians on the grid
t_locs	array of tuples	The coordinates of the targets on the grid
o_locs	array of tuples	The coordinates of the obstacles on the grid
c_locs	array of tuples	The coordinates of the control points on the grid
p_density	array of tuples	The density of pedestrians which are to be initialized randomly
timesteps	0-infinity	The maximum number of timesteps to run the simulation
p_locs_mode	custom, circle, random or uniform	The type of initialization for pedestrians location
p_locs_radius	0-grid_size	If the pedestrian initialization type is circle then the radius of the circle
p_region_x	0-grid_size_x	The x length of the region in which pedestrians are initialized randomly/uniformly
p_region_y	0-grid_size_y	The y length of the region in which pedestrians are initialized randomly/uniformly
p_coord	tuple	The upper left coordinate of the pedestrian initialization region
p_num	0-infinity	The number of pedestrians to be initialized if the initialization type is not custom
mode	normal or dijkstra	To use normal Euclidean distance or Dijkstra for path finding
disable_pedestrians	0 or 1	If the pedestrians should be disabled after reaching the target
avoid_obstacles	0 or 1	If the obstacle avoidance should be turned on

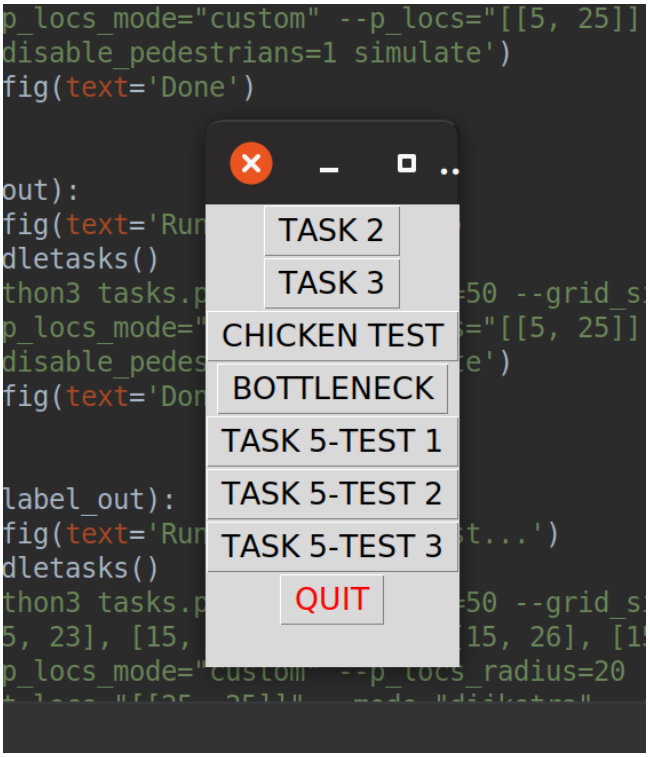

Figure 1: The simple GUI for running different tasks/scenarios

Report on task 2, First step of a single pedestrian

The task is to "define a scenario with 50 by 50 cells (2500 in total), a single pedestrian at position (5,25) and a target 20 cells away from them at (25,25)." The pedestrian is supposed to walk in 25 steps to the target and waits there. We set up the scenario successfully. (see Fig. 2)

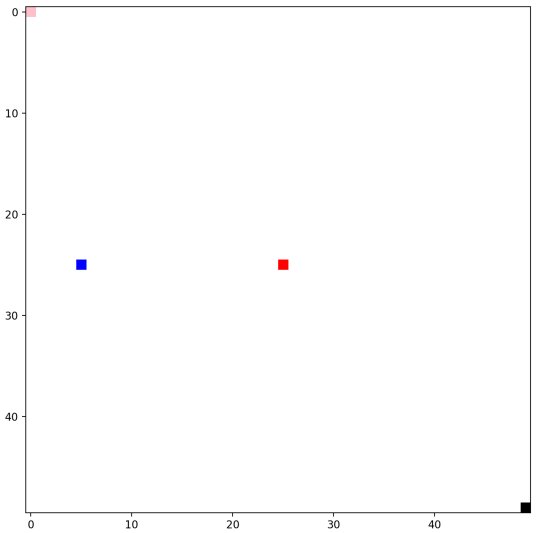

Figure 2: Start scenario of task 2

The pedestrian arrives in our scenario successfully at the target. The counter starts counting at zero, therefore the counter shows 24 steps at the end. The successful arrival after 25 steps can be seen in Figure 3. The pedestrian is marked in blue, while the target is colored red. We implemented the path the pedestrian takes in grey, so that it is possible to see which path the pedestrian took. The cellular automaton uses the Dijkstra algorithm to compute the best path from the pedestrian to the target.

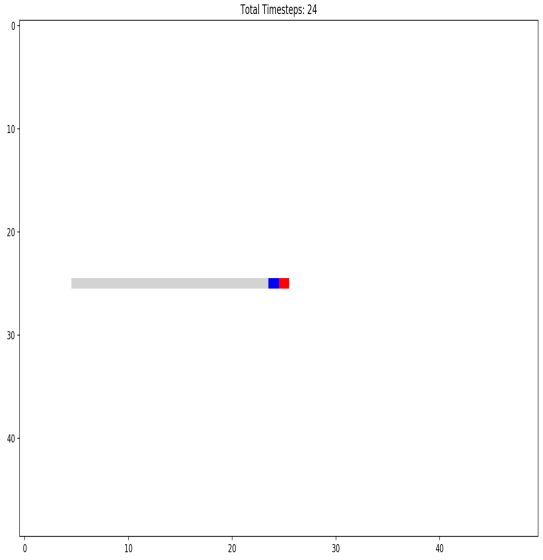

Figure 3: Pedestrian arrived at target successfully and waits

Report on task 3, Interaction of pedestrians

We use the previous defined scenario with 50 by 50 cells and a target at position (25,25). In this scenario five pedestrians are inserted in a circle with radius 20 around the target. The starting scenario can be seen in Figure 4, where the pedestrians are marked in blue, while the target is colored in red. In order to avoid pedestrian collision, for each pedestrian we add a cost for moving to a neighboring tile. For a pedestrian i, this cost is defined by the following formula:

$$P_{ik} = \sum_j^N \frac{1.5}{r_{kj}+\epsilon}$$

where P_{ik} is the total pedestrians avoidance cost for pedestrian i upon moving to the neighboring cell k, N is the total number of pedestrians, r_{kj} is the euclidean distance of the cell k to pedestrian j and ϵ is a very small constant to avoid division by zero.

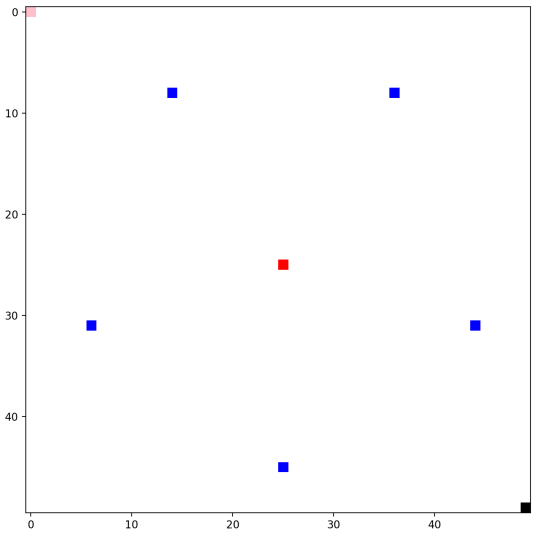

Figure 4: Setup of Task 3

We use the Dijkstra algorithm to find the shortest path from the pedestrians to the target. The paths of all pedestrians are colored in grey. The target has only four sides where pedestrians can reach it. Consequently the first four pedestrians reach the target directly, while the last pedestrian can only queue behind another pedestrian, since the way to the target is already blocked from every side. This can be seen in Figure 5.

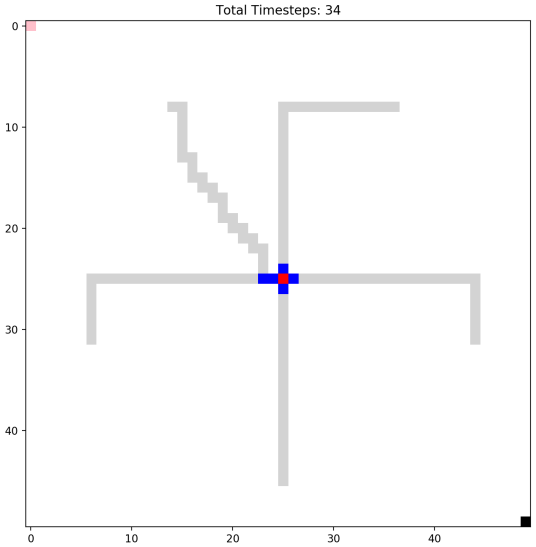

Figure 5: First pedestrian arrives after 9 steps

Report on task 4, Obstacle avoidance

As in the case of pedestrian avoidance, for obstacle avoidance a cost is added to the total cost of moving to each neighboring cell for a pedestrian. The cost can be defined by the following formula:

$$O_{ik} = \sum_j^M \frac{r_{max}}{r_{kj}+\epsilon}$$

where O_{ik} is the total obstacle avoidance cost for pedestrian i upon moving to the neighboring cell k, M is the total number of obstacles, r_{kj} is the euclidean distance of the cell k to obstacle j, $r_{max} = max(grid\_size\_x, grid\_size\_y)$ and ϵ is a very small constant to avoid division by zero.
The pedestrians managed to avoid the obstacle in the chicken test, and each other in the bottleneck test using Dijkstra's algorithm. As seen in Figure 6, the target is on the same y axis as the pedestrian. If the obstacles were to be ignored, the shortest path would have been to go straight down, but the pedestrian would have gotten stuck.

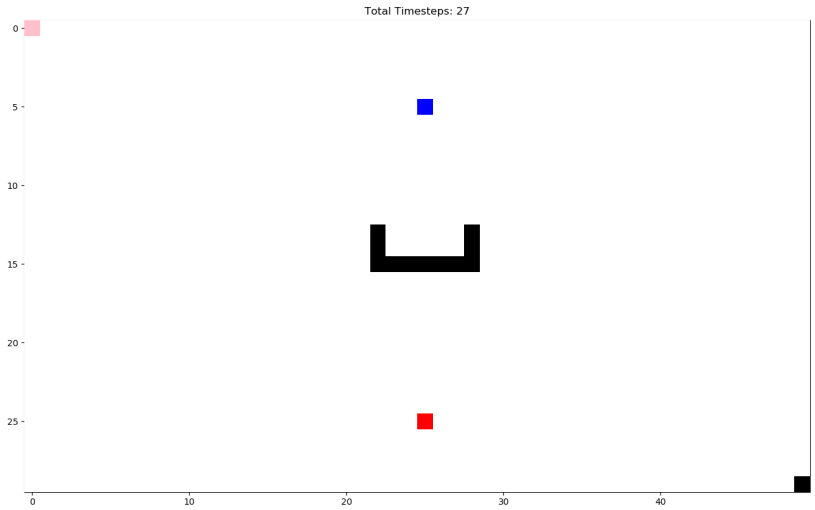

Figure 6: Initialization of the chicken test.

However, as seen in Figure 7, the pedestrian walks around the obstacle and reaches the target with the shortest path possible, since Dijkstra's algorithm guarantees completeness and optimality.

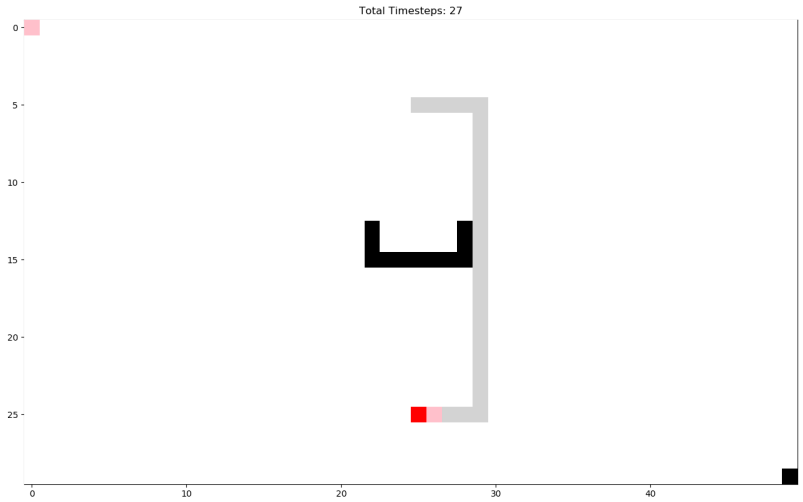

Figure 7: The pedestrian successfully avoids the concave obstacle.

Moreover, obstacle avoidance has been tested with a case of possible congestion as well. As seen in Figure 8, there are 3 pedestrians next to each other in an enclosed area with a 1 block wide exit.

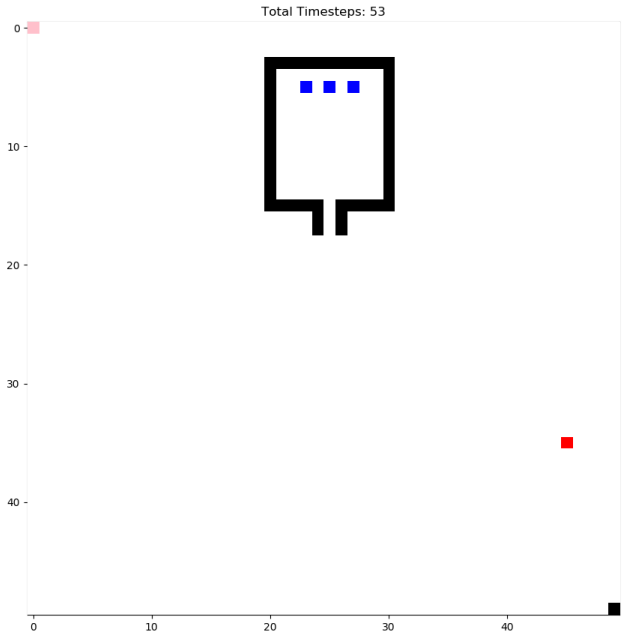

Figure 8: Initialisation of the congestion test.

As they reach the tight passage, pedestrians avoid hitting each other and wait for the other pedestrians to go through (see Figure 9).

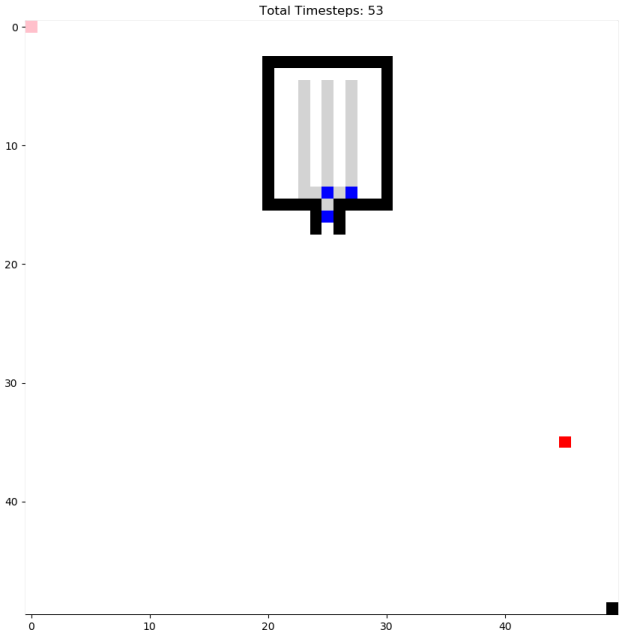

Figure 9: The middle pedestrian goes through first. The right one goes 1 block right and waits for the left one to go through.

Every pedestrian reaches the target successfully and they even take different optimal paths as seen in Figure 10.

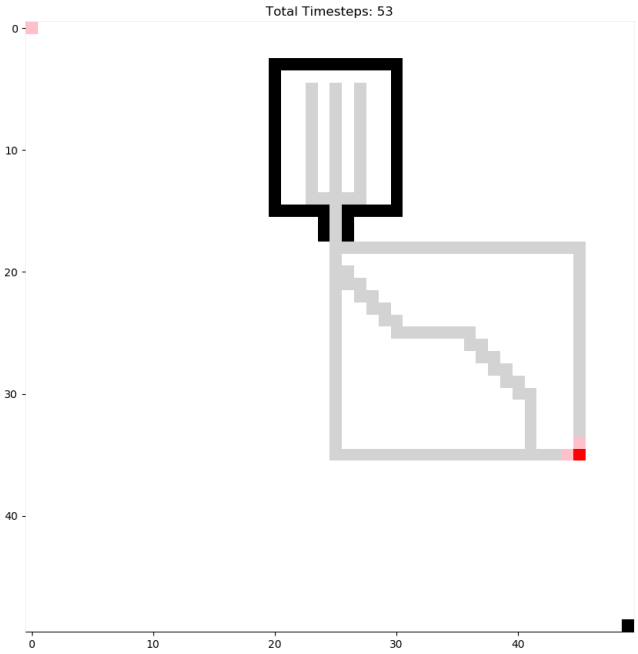

Figure 10: The pedestrians successfully avoid each other at the tight passage and reach the target.

If obstacle avoidance were not implemented, only the pedestrian in the middle could have got out and the other two would be stuck as illustrated in Figure 11.

Figure 11: The middle pedestrian manages to get out whilst the others just get stuck.

Report on task 5, Tests

TEST1: The RiMEA scenario 1 (straight line, ignore premovement time)

- The first test corresponds to the RiMEA scenario one, where one pedestrian which corresponds to 40 centimeters walks down a floor with a length of 40 meters and a width of 2 meters in between 26 and 34 seconds. The pedestrian has to walk with a speed between 4.5 km/h and 5.1 km/h.
- The pedestrian has to be 40 centimeters. The side of one cell is 40 centimeters long, since in our cellular automaton a cell corresponds to a pedestrian. As a consequence of that the floor has a length of 100 cells and a width of 5 cells. The setup can be seen in Figure 12.

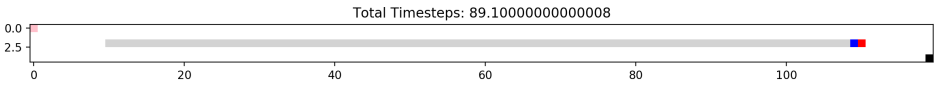

Figure 12: Setup of the RiMEA scenario one

One timestep in our simulation corresponds to $\frac{1}{3}$ of a second. The pedestrian has a walking speed of 1.33 $\frac{m}{s}$. It arrives after roughly 89 timesteps at the target (see Fig. 13). Consequently it takes 29,67 seconds for the pedestrian to walk down the floor. This is exactly in the given range of 26 to 34 seconds. So the test is passed successfully

- test successful

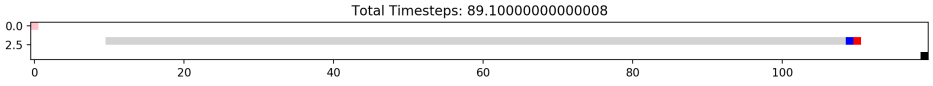

Figure 13: Arrival at the end of the floor after roughly 89 timesteps

TEST2: RiMEA scenario 4 (fundamental diagram, be careful with periodic boundary conditions).
- The second test corresponds to the RiMEA scenario four, where multiple pedestrians uniformly placed with a density value are going from left to right in a 1000 meter long passage with a width of 10 meters. Again, the width of a pedestrian is set 40 centimeters, thus a side of a cell is 40 centimeters long. The pedestrians walk at a speed of 1,2 $\frac{m}{s}$. There are measuring points of area 2 by 2 meters at 450 and 500 meters. The latter is the main measuring point.
- A single time step is set as a third of a second. So the pedestrians move 1 cell per time step. The length of the corridor is 2500 blocks and its width is 25 blocks.
- At each time step ranging from the 1^{st} to the 180^{th} time step (total of 60 seconds), the number of pedestrians present on the measuring point are added to the sum. After the simulation, the sum is divided by the time passed to get an average number of people per unit time in that area. Then the value is divided by the are of measuring point to get the density in that area. Finally, the density is multiplied by the pedestrian speed to get the flow of the measuring point. As seen in Figure 14, as the pedestrian density at initialization increases, the flow increases linearly.
- test successful -

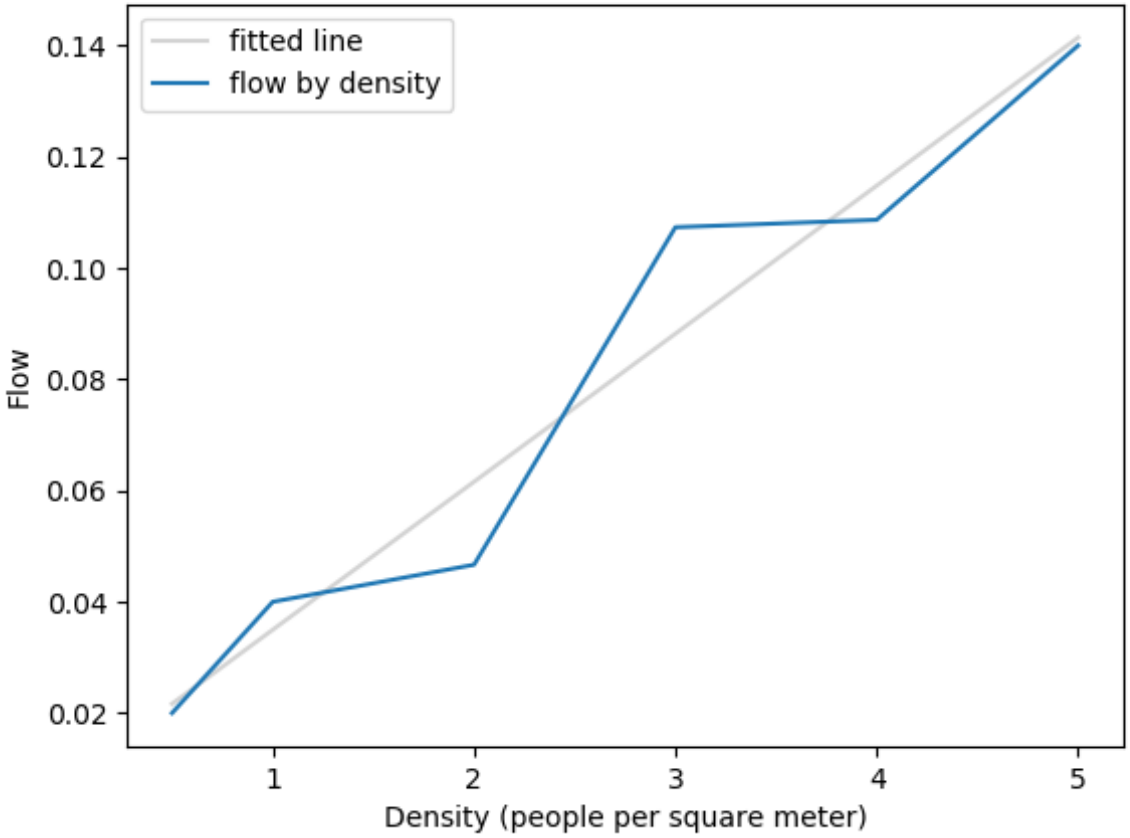

Figure 14: The increase of flow per density is linear.

TEST3: RiMEA scenario 6 (movement around a corner).
- The third test corresponds to RiMEA scenario 6 which is executed successfully if twenty uniformly distributed people walk around a left corner without passing through any walls. The starting setup can be seen in Figure 15.

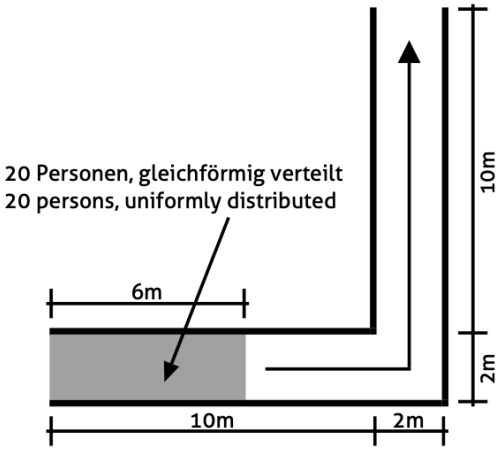

Figure 15: Setup of RiMEA scenario 6

Figure 16 shows one setup scenario of our simulation where 20 pedestrians are uniformly distributed before the corner.

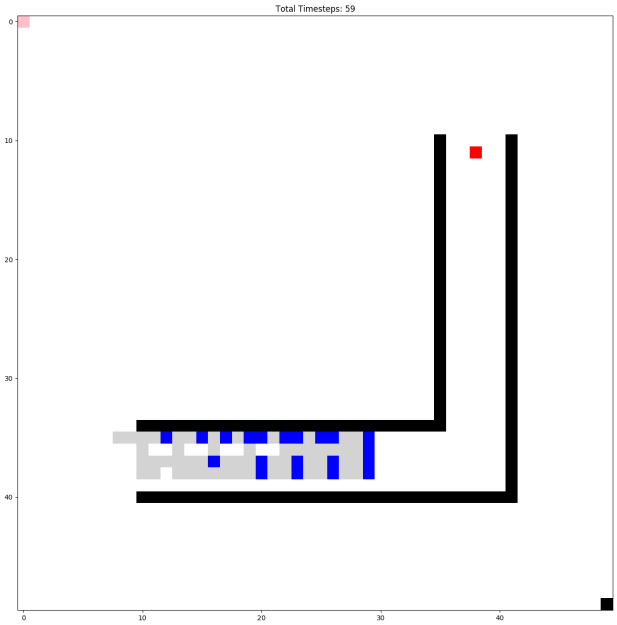

Figure 16: Uniformly distributed pedestrians for setup of RiMEA scenario 6

- All twenty pedestrians walked around the corner successfully without passing the walls. Figure 17 shows that all the pedestrians walked successfully around the corner and at figure 18 they all reached the

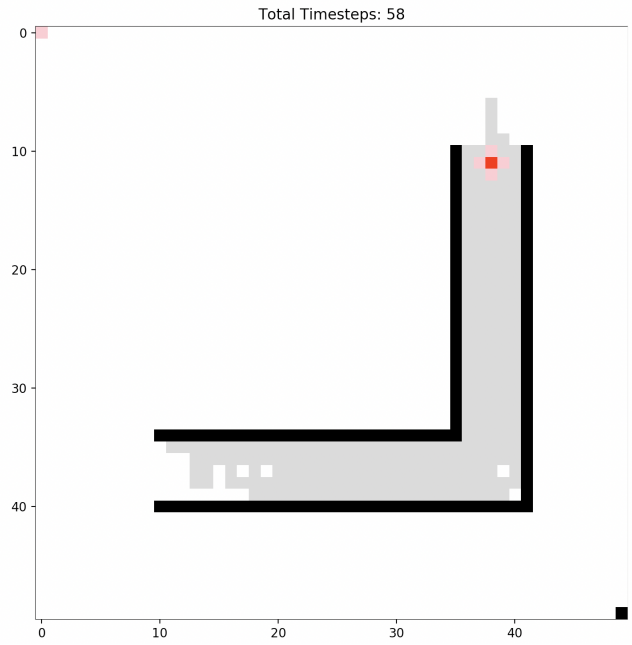

Figure 18: All passengers reached the target after the corner successfully

red colored target. For this scenario we implemented that the pedestrians are disappearing after having reached the target, because the goal of the test is to see if the pedestrians are able to walk around the corner correctly. If we wouldn't let them disappear the pedestrians would queue around the target and block each other.

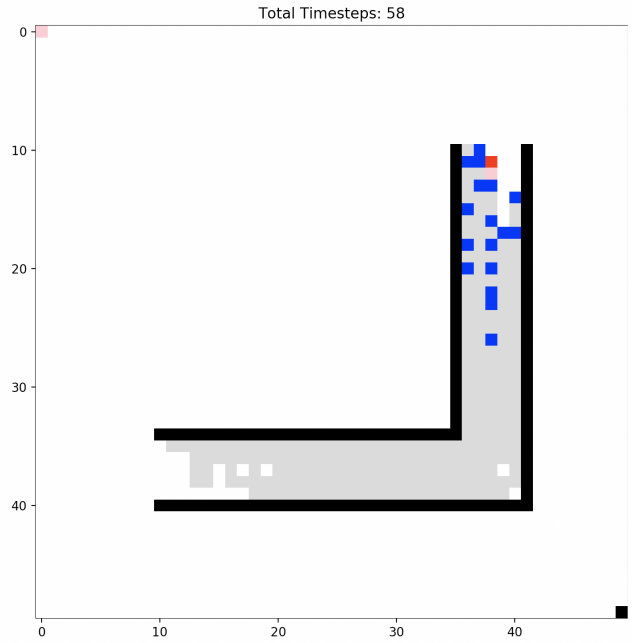

Figure 17: All 20 pedestrians walked around the corner successfully

From Figure 18 can be seen that almost all cells (grey colored) were used to reach the target (red colored) around the corner. All pedestrians walked around the corner successfully. Consequently the test is passed successfully. - test successful -