Report for exercise 1 from group K

Tasks addressed:	5
Authors:	Oliver Beck (03685783) Junle Li (03748878) Chenqi Zhou (03734992)
Last compiled:	2021–04–28
Source code:	https://github.com/Linnore/MLCMS-EX1-GroupK

The work on tasks was divided in the following way:

Oliver Beck (03685783)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%
Junle Li (03748878)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%
Chenqi Zhou (03734992)	Task 1	33,3%
	Task 2	33,3%
	Task 3	33,3%
	Task 4	33,3%
	Task 5	33,3%

Abstract

In this exercise, we learned how to model and simulate human crowds with the simple cellular automaton. We created a GUI for the cellular automaton crowd simulator, and our model successfully passed some required RiMEA tests.

In general, the cellular automaton is a computational tool to model a complex system, i.e. a system that is built up of many interacting parts. Typically, each individual part is relatively simple, but the interactions can lead to very compex behavior.

A crowd of humans can be modelled using the cellular automaton, where pedestrians are placed in cells and are driven to move by the distance cost to targets and the cost contributed by avoidance.

Python	3.8.8
PyCharm Community Edition	2020.1.2
Visual Studio Code	1.55.2
Jupyter Notebook	6.3.0

Table 1: software versions

Report on task 1, Setting up the modeling environment

Task description Setup your implementation and simulation environment with the following requirements:

1. Basic visualization.
2. Adding pedestrians in cells.
3. Adding targets in cells.
4. Adding obstacles by making certain cells inaccessible.
5. Simulation of the scenario (being able to move the pedestrians).

Basic Requirements of Task 1

Basic Visualization We visualize the cellular automaton by implementing a GUI (Figure 1). It is developed with the python library tkinter. Widgets like Label, Button, Radio button, Entry and Canvas are used.

On the left hand side of our GUI, labels, buttons and an entry build the "Control Panel". Users are able to change the states of cellular automaton, launch a preset task or RiMEA test, and run/pause the simulation.

On the right hand side of our GUI, the cellular automaton is visualized by a Canvas object, as it has a 2D grid by nature. Default size of the cellular automaton is "50*50", but the users can still set up the size of cellular automaton by typing the required size into the entry field in the format of "width*height" and clicking the button "Reset". Each cell in the cellular automaton is represented by a rectangle in the Canvas whose scale can be changed. Thus each cell can be better visualized. Under the current setting, the pipelines of task3, RiMEA test 4 and test 7 will change the scale to 4 pixels to adopt larger cellular automaton; other than this, the scale is 8 pixels by default. Powered by such design, the users can watch the simulation comfortably.

Adding Pedestrians in Red By clicking the button "Add Pedestrian", the GUI will switch to the drawing mode for adding pedestrians, then the users can freely draw the pedestrians in **Red** onto the canvas using the mouse. Notice that when the mouse moves into the canvas, the cell pointed by the mouse will be highlighted in **Pink**. The users then sees his current cursor position and can therefore place objects precisely. Figure 2(a) shows a free drawing using our GUI.

Adding Targets in Green This is the same as adding pedestrians. One can draw the targets in **Green** onto the canvas after clicking the button "Add Target".

Note that it is allowed to add multiple targets in our design

Adding Obstacles in White This part is works similar to adding pedestrians and targets. One can draw the obstacles in **White** on the canvas after clicking the button "Add Obstacle".

Launch the Simulation After adding pedestrians, targets and some optional obstacles, the users can launch the simulation by clicking the button "Run". Once the simulation starts, the timer next to the button "Pause" will start counting. If the users want to observe behaviors of pedestrians more clearly during the simulation, just click the button "Pause".

Other Settings in our Program

About Timer The timer on our GUI reports the theoretical time in the simulation. Therefore, the timer shows the simulated time and not the actually passed time. This is noticeable when the number of pedestrians is relatively large (e.g. > 150). This is due to the time consumption and offsets we set in order to achieve the behaviour of the cellular automaton, also updating the canvas then takes longer during the simulation.

About Unit In our visualization, each cell of the cellular automaton represent a square area of size $\frac{1}{3}m \times \frac{1}{3}m$. The overall timer proceeds the System every 100ms.

About Speed The default speed of the pedestrians are set as 1.33m/s (i.e. stepping 4 cells per second). The speed configured in our program is with respect to the theoretical time of the simulation (the time that timer shows). The speed of the pedestrians is allowed to be changed in the codes, but not through the GUI. In RiMEA test 1 and test 7, the simulation will show pedestrians with different walking speed. As for how we successfully simulated a defined speed in the program, see the report on task 5 RiMEA test 1.

About Movement Movement of pedestrians is only allowed horizontally or vertically, not diagonally.

About Utility/Cost Function This will be further illustrated in the report on task 3 and task 4.

About Reaching Target In our setting, we consider the target as exit to the cellular automaton. Therefore, when a pedestrian reaches the target, everything about this pedestrian will be deleted. This includes its avoidance cost contributing to other pedestrians and etc. Under such design, it is actually possible that in seldom scenarios pedestrians want to enter the area surrounding the target at the same time and don't proceed due to avoidance cost. This shows that the specific behaviour of the Cellular Automaton and its characteristics takes place. To avoid this kind of "Deadlock" there would be many easy solutions such as asynchronous update, but we like to keep this characteristic as it is part of the Cellular Automaton behaviour.

Result

We successfully set up the modeling environment using Python ans especially the library tkinter. Figure 2(b) illustrates an example visualization of the state of a cellular automaton with 50 by 50 cells (2500 in total): two cells in state P (red) at position (9,15) (30,10) respectively; one cell in state T (green) at position (25,25); seven cells in state O (white); the rest of cells are in state E (black).

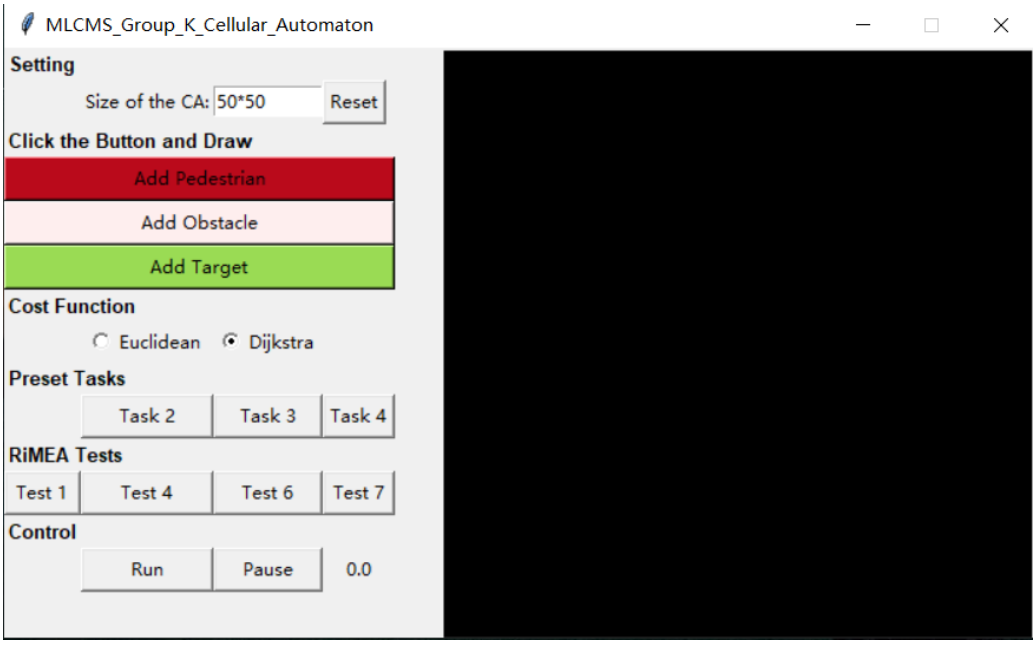
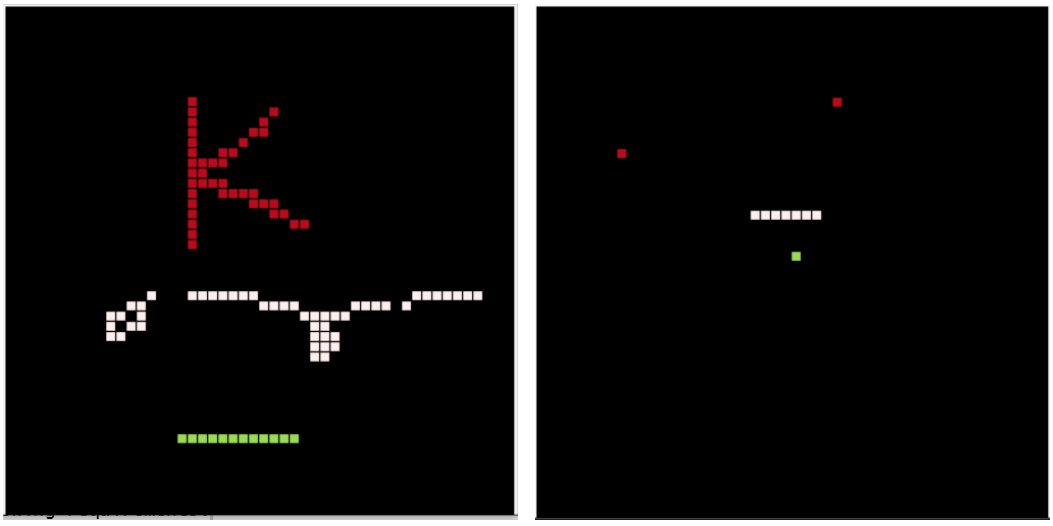


Figure 1: Our GUI



(a) A free drawing of pedestrians (red), obstacles (white) and targets (green)

(b) An example state of the cellular automaton

Figure 2: Two example states created by our GUI

Report on task 2, First step of a single pedestrian

Task description 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). Simulate the scenario with the cellular automaton for 25 time steps, so that the pedestrian moves towards the target and waits there.

Solution Task 2 requires the realization of the basic application of our cellular automaton, i.e. setting the size (width, height) of the cellular automaton, adding pedestrians and adding targets etc., which is illustrated in the report on task 1.

We add a single pedestrian cell at the specified position and a single target cell at the specified position as required.

Result By clicking the button "Task 2", the GUI will update the cellular automaton into the required setting of this task. After that, click "Run" the simulation will launch. Figure 3 shows the result of this task, i.e. in a square cell grid of 50 by 50 cells (2500 in total), after 25 time steps, a single pedestrian at position (5, 25) reaches the target at (25, 25) and then disappears. (Based on our own settings, not "waiting there" required in the exercise sheet.)

Figure 3 has four sub figures, fig. 3(a) is the beginning of the scenario, fig. 3(b) is the the process of the pedestrian moving towards the target, fig. 3(c) shows that the pedestrian reaches the target, and fig. 3(d) shows that the pedestrian disappears at the target.

(a) t=0s (b) t=2.8s (c) t=5.0s (d) t=25s

Figure 3: Scenario with a single pedestrian

Report on task 3, Interaction of pedestrians

Task description Now the pedestrians have to interact with each other. Insert five pedestrians in the previous scenario, arranged in a circle in a fairly large distance (30-50m) around a single target in the center of the scenario. Run the scenario and report your findings.

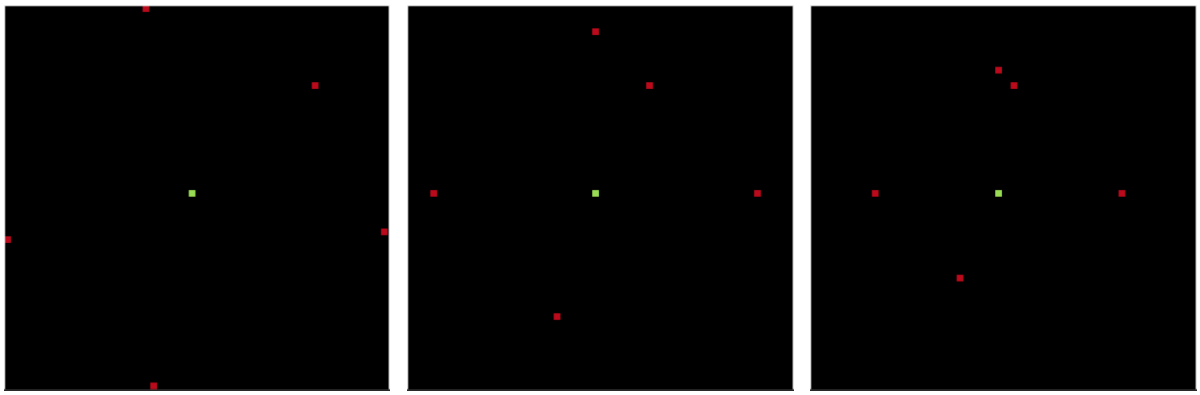
Solution Task 3 requires the realization of pedestrian avoidance of our cellular automaton. Interactions of individuals with others is typically modelled through cost functions. We insert five pedestrians in the previous scenario, arranged in a circle around a single target in the center of the scenario. We used `add_avoidance_cost` to implement pedestrian avoidance.

The cost function c(r) modeling the interaction between two individuals at distance r can be simplified to:

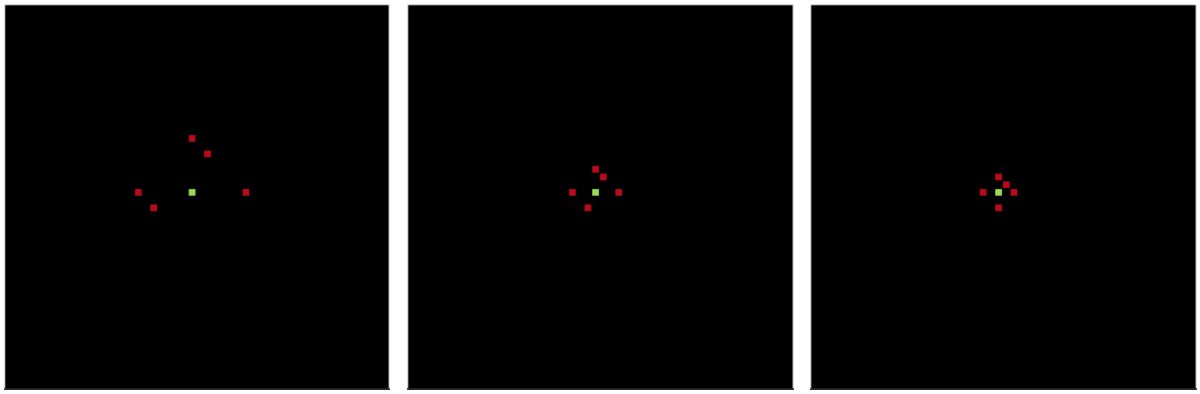
$$c(r) = exp(\frac{1}{r^2 - r_{max}^2}), if r < r_{max}, \tag{1}$$

Result Initially all pedestrians are arranged in a circle around a single target in the center of the scenario and move towards the target. When two pedestrians approach, they will interact and avoid each other. The pedestrians all reach the target roughly at the same time.

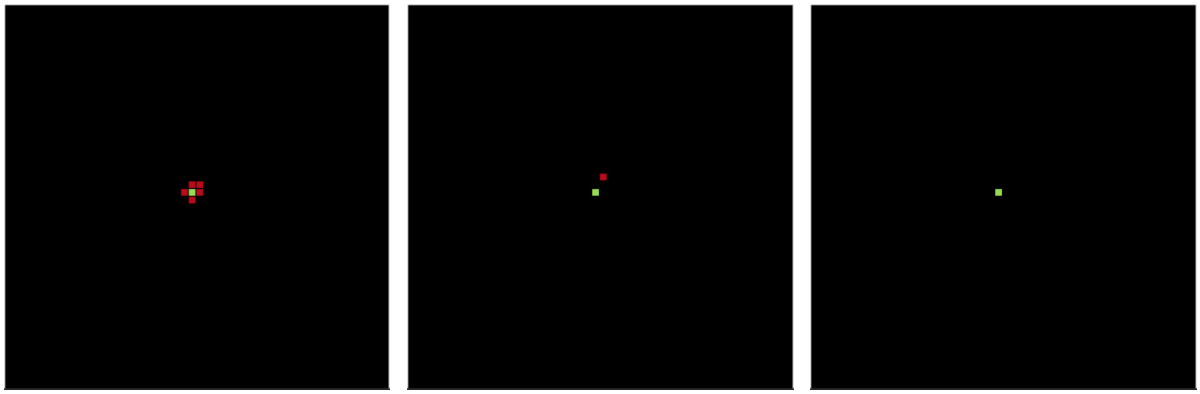
Figure 4 has nine sub figures, fig. 4(a) is the beginning of the scenario, the remaining figures show that all pedestrians move towards the target in the center of the scenario, interact with each other when approaching, and reach the target roughly at the same time.



(a) t=0s (b) t=2.3s (c) t=3.5s



(d) t=5.7s (e) t=6.6s (f) t=7.0s



(g) t=7.2s (h) t=7.6s (i) t=8.2s

Figure 4: Simulation of pedestrian avoidance

Report on task 4, Obstacle avoidance

Task description Up to now, there were no obstacles in the path to the target. Implement rudimentary obstacle avoidance for pedestrians by adding a penalty (large cost in the cost function) for stepping onto an obstacle cell. Determine the impact of obstacle avoidance on whether pedestrians can reach the target in the "chicken test" scenario (see fig. 5) .

Implement the Dijkstra algorithm to flood the cells with shortest-path distance values, starting with zero distance value at the target, such that obstacle cells are not included in the set of possible cells.

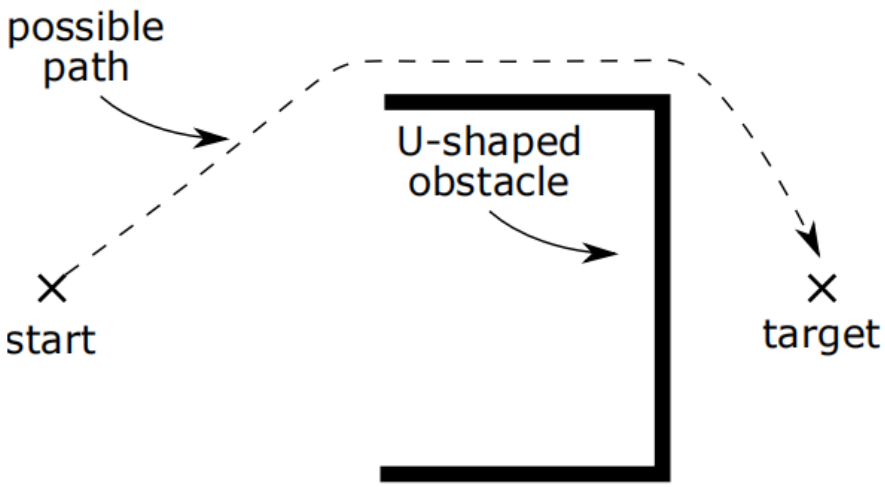


Figure 5: The "chicken test"

Solution Task 4 requires the realization of obstacle avoidance. We set up a "chicken test" scenario with U-shaped obstacles by adding pedestrians, adding targets, and adding obstacles, which are the fundamental functions of our cellular automaton.

About Euclidean distance Function In order to advance the system to the next state, we use Euclidean Distance to compute the distance between current cells and the target.

$$d(current_{cell}, target) = \sqrt{(i-k)^2 + (j-l)^2} \tag{2}$$

that is, the Euclidean distance between the cell indices ij and kl, where i and k are the row indices, and j and l are the column indices of the cells.

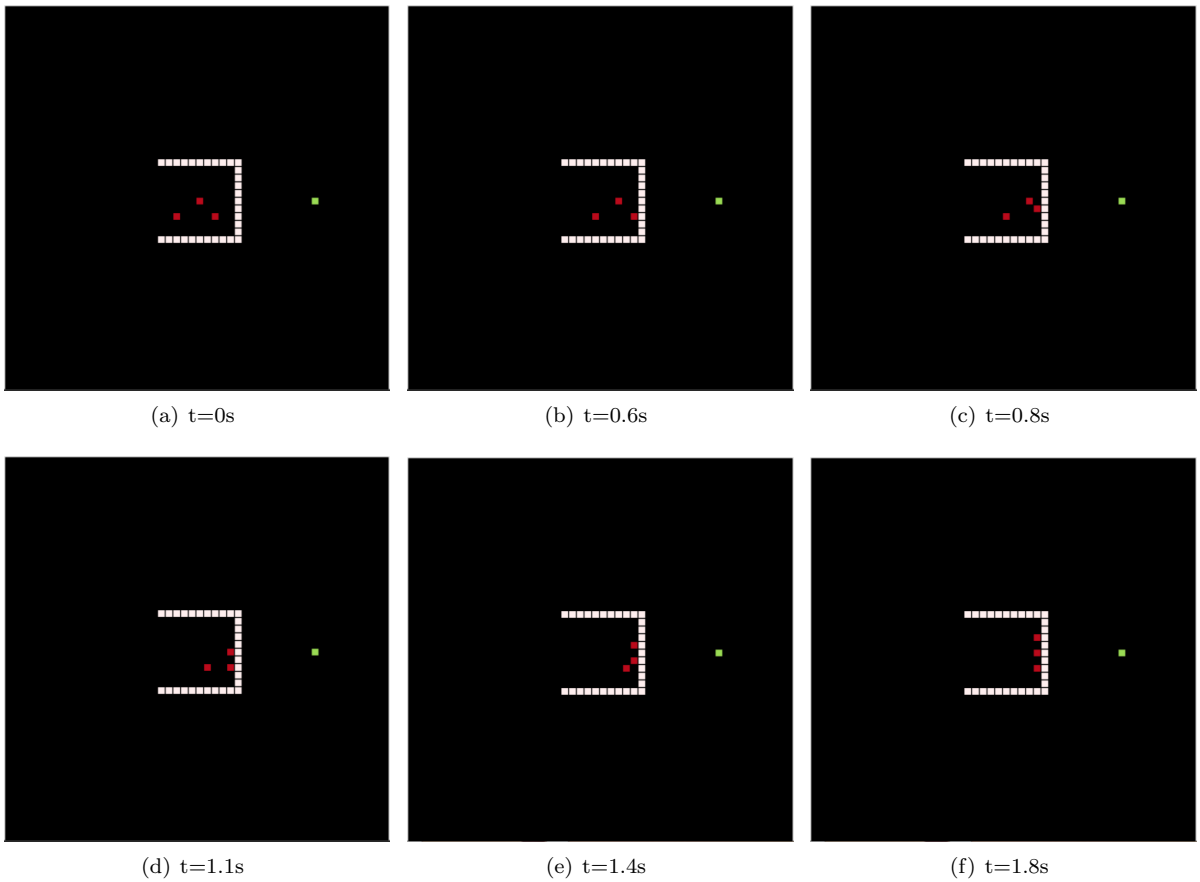
About Dijkstra algorithm To avoid dead ends it is necessary to implement some kind of "foresight". This is achieved by running the Dijkstra algorithm from the target nodes, across the whole grid. The result is a distance map that ignores dead ends and shows preferable paths. This distance map can be used in a cost function. Every potential pedestrian now can easily locally look up neighboured cells and their respective distance to the nearest target and can form an "informed" decision based on its individual cost function that considers with the distance map considers global elements.

Since we concentrate on vertical and horizontal movements only, and not on diagonal ones the distance map also only has to in cooperate this, e.g. the distances are Manhattan-distances. This allows us to implement Dijkstra very smoothly. This very likely outperforms any Fast Marching Algorithm since lots of the overhead isn't needed in this specific case.

We start with all target nodes in a list. We set the rest of the map to infinite or unknown distance. From the target nodes on we add all the immediate neighbours and set them to their respective distance: 1. From this set of neighbours that for all the targets has the same distance in each iterations step, we proceed to the neighbours neighbours add the current distance +1 and so on. This happens simultaneously for all target nodes. Once a distance value is assigned it doesn't need to be updated. Every known value in the neighbourhood is correct and can be ignored. When no unknown neighbour appears the algorithm is finished. If values are left unknown they are unreachable.

Result If obstacle avoidance is not implemented, pedestrians in the "chicken test" will get stuck and cannot reach the target.

Figure 6 has six sub figures, fig. 6(a) is the beginning of the scenario, and fig. 6(f) is the configuration of the pedestrians after the simulation, showing that all pedestrians will be blocked by the U-shaped obstacle and trapped in it, unable to reach the target.



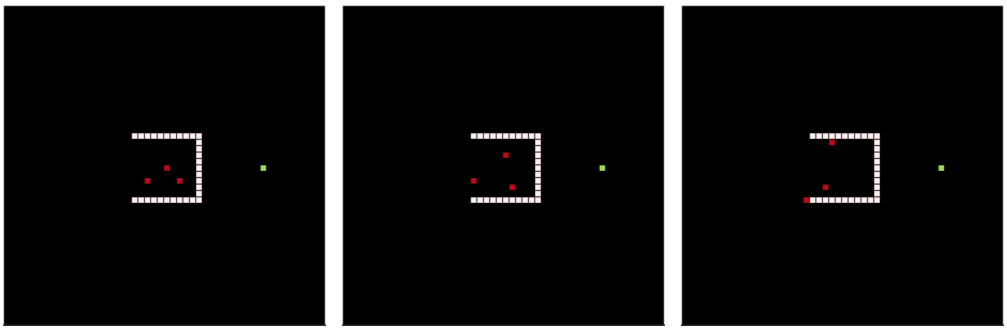
(a) t=0s (b) t=0.6s (c) t=0.8s

(d) t=1.1s (e) t=1.4s (f) t=1.8s

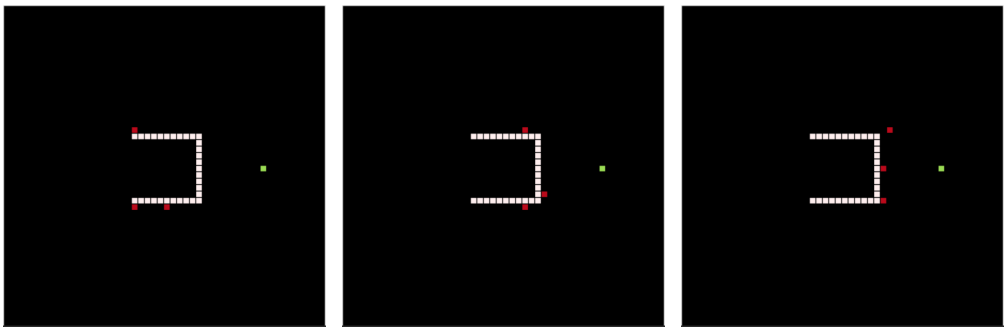
Figure 6: Without obstacle avoidance

After implementing the Dijkstra algorithm and filling cells with distance values, pedestrians can avoid obstacles and reach the target. In the "chicken test" scenario, pedestrians can bypass the U-shaped obstacle and reach the target one after another.

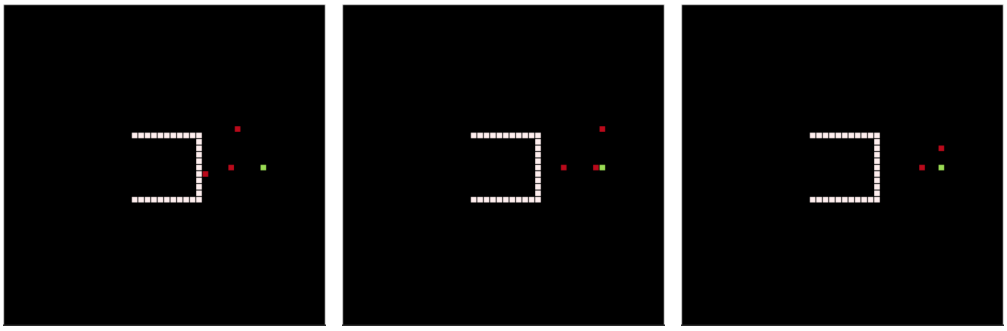
Figure 7 has eleven sub figures, fig. 7(a) is the beginning of the scenario, the remaining figures are the process of pedestrians bypassing the U-shaped obstacle and reaching the target one by one.



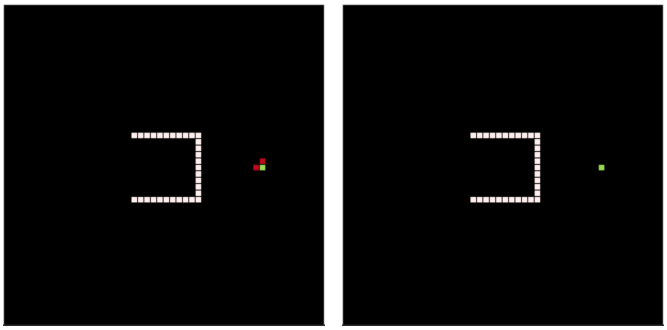
(a) t=0s (b) t=0.6s (c) t=1.5s



(d) t=3.0s (e) t=5.1s (f) t=5.9s



(g) t=6.9s (h) t=7.6s (i) t=8.4s



(j) t=8.6s (k) t=8.9s

Figure 7: With obstacle avoidance

Report on task 5, Tests

Task description

After you implemented pedestrian and obstacle avoidance, you have to test your implementation with the following scenarios from the RiMEA guidelines. They provide support for verification and validation of simulation software for crowds. The tests in the guideline may contain features that are not implemented in your cellular automaton. Discuss why you need to implement them to complete the test, or why you can neglect the particular feature and still obtain reasonable test results.

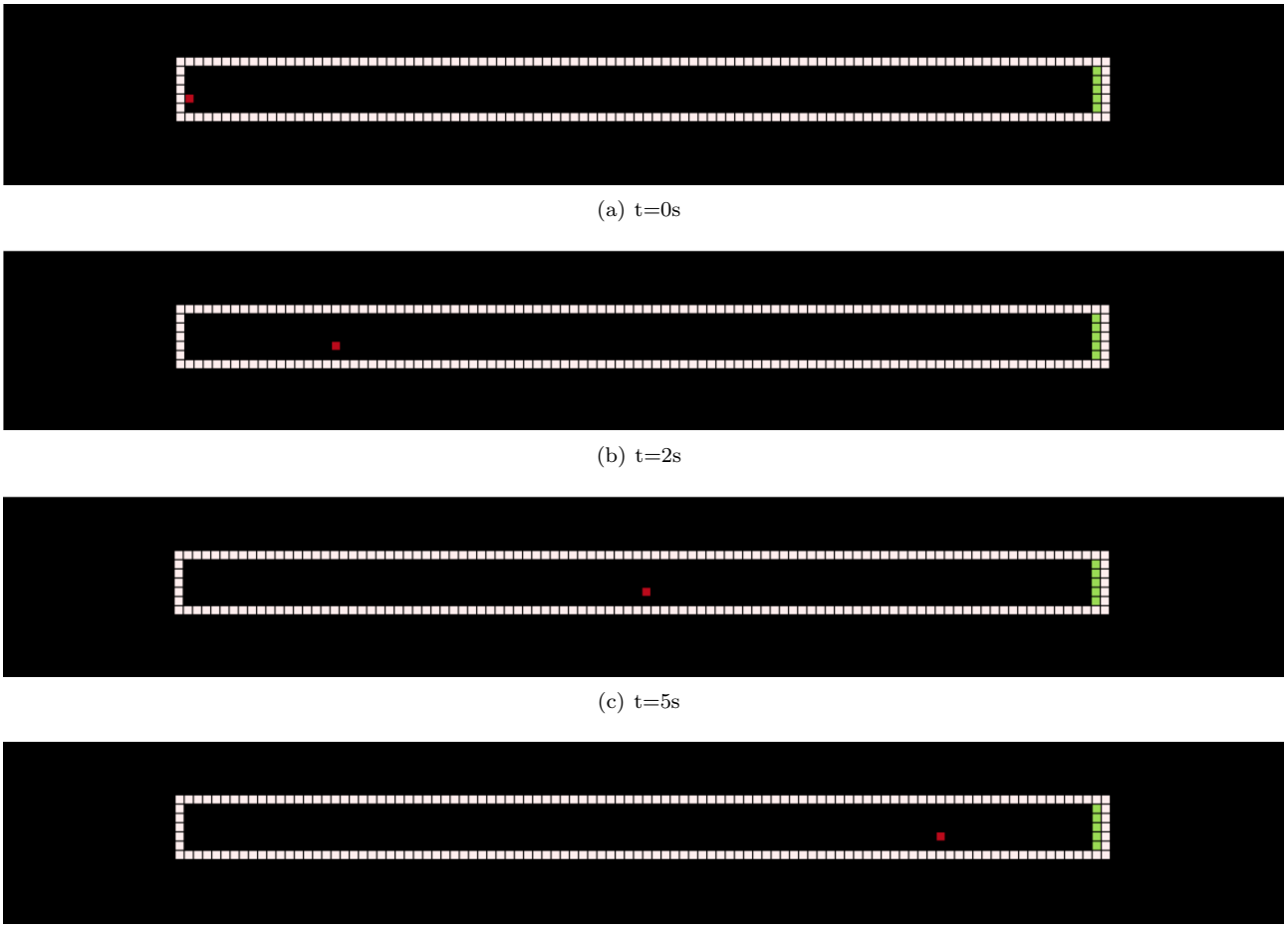
TEST1: RiMEA scenario 1 time

Task description Simulate pedestrians with well defined speed. It is to be proven that a person in a 2 m wide and 40 m long corridor with a defined walking speed will cover the distance in the corresponding time period. If 40 cm (body dimension), 1 second (premovement time) and 5 percent (walking speed) are set as imprecise values, then the following requirement results with a typical pedestrian speed of 1.33 m/sec: the speed should be set at a value between 4.5 and 5.1 km/h. The travel time should lie in the range of 26 to 34 seconds when 1.33 m/sec is set as the speed.

Test goal Pedestrians with a random speed between 4.5 and 5.1 km/h should have a travel time between 26 to 34.

About individual pedestrian speed Different speeds of pedestrian are absolutely necessary for this test. The different speed of pedestrians was realised with a variable for each pedestrian that decides about the update frequency and therefore about the overall speed of the respective pedestrian. The overall clock for the Cellular Automaton updates every 100ms. The clock of the Cellular Automaton then checks which pedestrians are currently allowed to update according to their respective speed and their update schedule. The individual pedestrian speed was needed in the different RiMEA test cases and implemented this same way for all of them. For Example a Pedestrian that is allowed to advance one field, defined as 1/3 meter, every 300 ms has a speed of 1m/s. We specifically chose the clock to update at a granularity of 100ms and therefore limit the updates of the pedestrians to those intervals to keep the Cellular Automaton properties, that otherwise would have been lost in a continuous setting.

Result Since the body dimension that was required is 40cm we did a simple rescaling and reinterpreted speed and distance for this exercises. If the granularity is 40 cm instead of 33cm then it takes longer to cross one field for example. We didn't take any premovement time into account since the principle of the Cellular Automaton is to decide based only on the current state of the system. Any premovement of pedestrians thus wouldn't have changed the current movement of the system. With this required adaptions to full fill the desired start scenario, the travel time of pedestrians was within the desired time frame with the already existing implementation. The output of the testrun can be found in the directory "output".



(a) t=0s

(b) t=2s

(c) t=5s

(d) t=8s

Figure 8: Different time stamps of a randomly chosen speed and randomly chosen start point within the desired interval

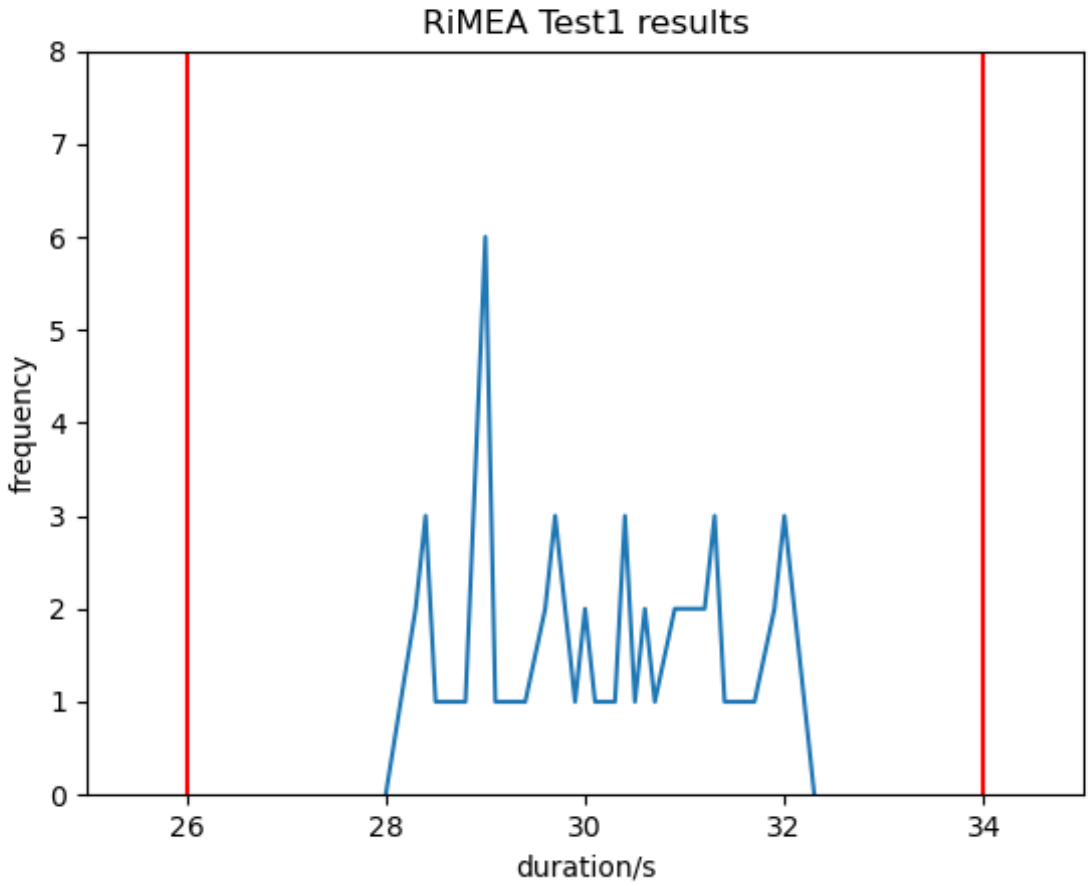


Figure 9: 50 Iterations with random speed of Pedestrian

TEST2: RiMEA scenario 4 (fundamental diagram, be careful with periodic boundary conditions).

Task description According to the RiMEA guideline, a corridor of size $1000m \times 10m$ filled by different density should be simulated. With different density values ($0.5P/m^2$, $1P/m^2$, $2P/m^2$, $3P/m^2$, $4P/m^2$, $5P/m^2$, $6P/m^2$), the program should be able to measure the speed of a crowd at certain measuring points. Then create the fundamental diagram (flow against density) using flow = speed*density.

Test goal Show that our model can produce the required fundamental diagram.

Solution

In our setting, each cell of the cellular automaton has size $\frac{1}{3}m \times \frac{1}{3}m$. Therefore, a corridor of size $1000m \times 10m$ should be modeled as a cellular automaton of 3000×30 cells. This number exceeds the capacity of our model; therefore, we reduced the size of corridor to $50m \times 5m$ (i.e.$150 \times 15 = 2250$ cells).

Settings about the corridor. Figure 10(a) shows the corridor we used in this test. At the end of the corridor, an exit with width 1m is placed as the target (green). In the right third of the corridor, a measuring area ($\frac{4}{3}m \times \frac{4}{3}$) is placed (yellow). Pedestrians that enter the measuring area from its left boundary and leave the measuring area from its right boundary will be recorded and measured.

Settings about measuring speed Say P and Q are the positions that a speed-measured, a pedestrian enters and leaves the measuring area respectively, then the measured speed of this pedestrian will be $v = Manhattan(P, Q)/Duration$, where Manhattan (P, Q) denotes the Manhattan distance between P and Q. We used the Manhattan distance because our cellular automaton does not allow the pedestrians to move diagonally.

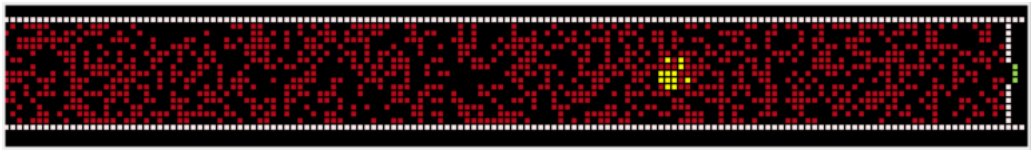
In each run of RiMEA Test 4, the program will start measuring the speed of pedestrians that pass through the measuring area at t=5.0s. The measuring process will maintain 30s. After t=35.0s, the simulation will stop and information of the speed-measured pedestrians including average speed will be output as a text file in the folder "./output/". Figure 11 shows the first few lines of measuring result when density is $4P/m^2$.

The information about average speed at the measuring area is crucial for the plotting of fundamental diagram, in which the flow is computed by average speed times density.

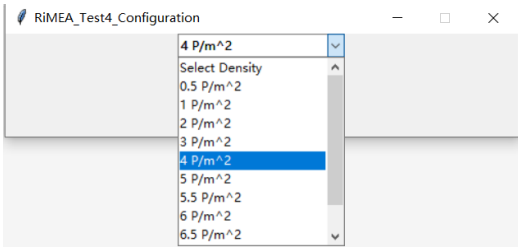
These additions were crucial to implement the test according to the guidelines.

Settings about density. To fill the corridor with a specific density ρ, our program would generate $150 * 15\rho$ random number from $[0, 150 * 15]$ without replacement, then get the corresponding coordinates for the pedestrians according to these random numbers. Figure 10(b) shows the corridor when it is filled pedestrians with density $\rho = 4P/m^2$.The user can select different density values for RiMEA test 4 in GUI. Right after clicking the button "Test 4" in our GUI, a small window will prompt out to ask the user to select a density value (Figure 10(c)) and confirm it (Figure 10(d)). The cellular automaton in the GUI will automatically fill itself by the selected density.

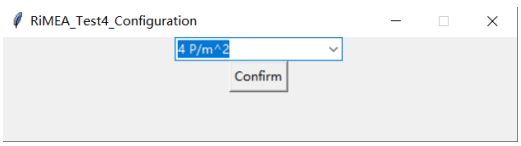
(a) An empty corridor with size $50m \times 5m$ (153*21 cells in total). Measuring area is in yellow.



(b) Corridor filled by density 4 P/m^2, t=0s



(c) GUI: Selecting density



(d) GUI: Confirm selected density

Figure 10: RiMEA Test 4 Set up

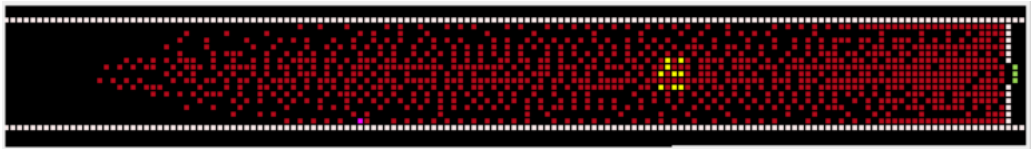
```
Number of pedestrians that pass through the measuring area: 72
Average speed: 0.81m/s
Pedestrian list:
Pedestrian 1, 1.11m/s, from (98, 12) to (103, 12), consumed 1.50s
Pedestrian 2, 1.39m/s, from (98, 10) to (103, 10), consumed 1.20s
Pedestrian 3, 1.28m/s, from (98, 8) to (103, 8), consumed 1.30s
Pedestrian 4, 1.39m/s, from (98, 10) to (103, 10), consumed 1.20s
Pedestrian 5, 0.67m/s, from (98, 10) to (103, 11), consumed 3.00s
```

Figure 11: An example output of RiMEA Test 4 pipeline when density is $4P/m^2$

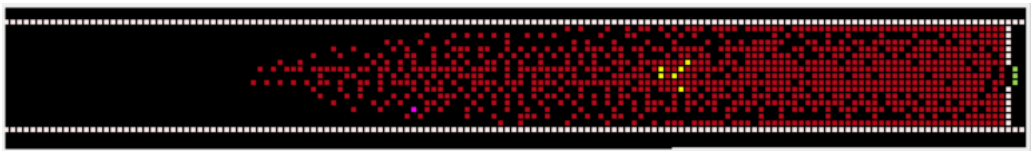
Results Our model can successfully simulate the crowd in the corridor filled by the required density. Figure 10(b), Figure 12 shows the simulation of the corridor when density is $4P/m^2$ during the test. Furthermore, based on the average speed measured in rounds of simulation, the fundamental diagram is plotted using the library matplotlib of Python (Figure 13).

Notice that the fundamental diagram has the shape the same as the one in RiMEA guideline. After the flow is situated around $\rho = 4P/m^2$, it drops down. When the density is $\rho = 8P/m^2$, the crowd is stuck and no pedestrian is able to pass through the measuring area.

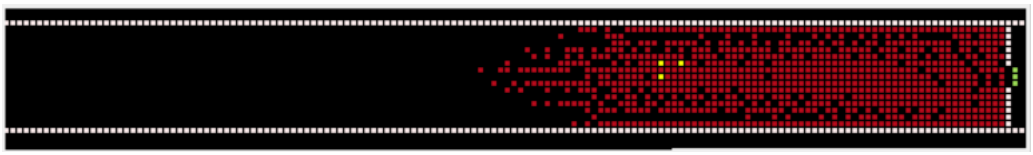
To make our model reasonable enough and generate a satisfactory fundamental diagram, we explored different ideas about the updating step. One crucial setting that should be highlighted here is that the pedestrians cannot be "wise" in our model. At each time step, we allow the pedestrians that are about to move to choose the next position according to values of cost function at the last time step. After collecting all such "next positions", we handle the update of the cellular automaton. It is vital to allow several pedestrians to choose the same next position "unwisely". Thus following the discrete update scheme of the cellular automaton. However, only one of them will actually move into his expected next position. As for this pedestrian's competitors, unfortunately, they have to slam the brake to avoid collision. This setting is reasonable in reality, and guarantees that the updating choice of any pedestrian is independent of the updating choice of other pedestrian. Otherwise, pedestrians would be "wise" and never consider a next position that will make them potentially collide with others. Our group implemented the pedestrian-all-wise model at first, and the result is that the flow would never drop down and remain saturated even when all cells are full.



(a) t=10.1s



(b) t=20.1s



(c) t=35.0s

Figure 12: An example run of RiMEA Test 4 when density is $4P/m^2$

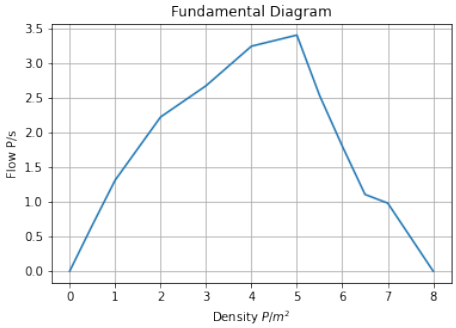


Figure 13: RiMEA Test 4: Fundamental Diagram

TEST3: RiMEA scenario 6 (movement around a corner).

Task description

Movement around a corner: Twenty persons moving towards a corner which turns to the left will successfully go around it without passing through walls.

Test goal

The goal of the test is testing basic functionality of our Cellular Automaton in a slightly advanced setting. Weaknesses and strengths of different approaches might be revealed.

Usage of features

Basically all features required for this test were all implemented with the first 4 tasks of this exercise. What we made a lot use of in the RiMEA tests is avoidance of other pedestrians. Important are the cost function and avoidance of other pedestrians and obstacles. Dijkstra or Euclidean cost functions don't really produce any different results, since there is no wrong way that isn't clearly indicated by the presence of an obstacle wall. Moving up or moving right for the most part is an equally good option before the bend.

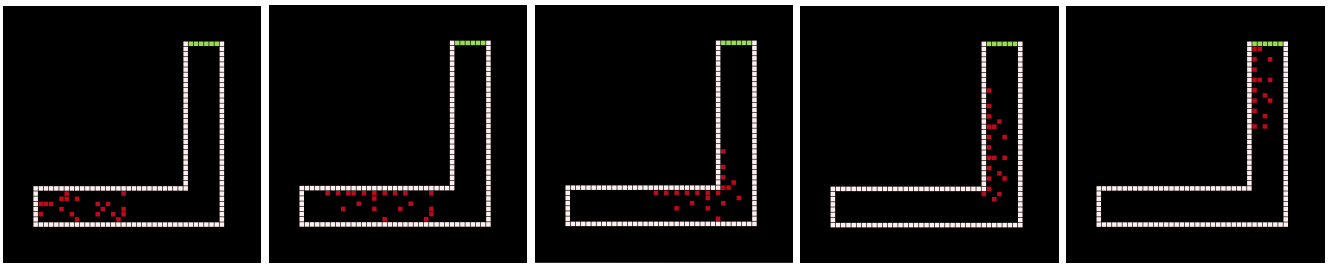
Result

The basic outline of the test with boundaries, avoidance and reaching the target in a reasonable time are full filled for all the test runs indicating a robust implementation. Interesting behaviour we could observe was that depending on the weighting of avoidance versus distance matrix the simulation behaved very differently.

It was possible for us to weight the distance matrix and the avoidance cost variably for our overall cost function that we used for the pedestrian behaviour. According to how we weighted these we perceived different behaviours that we explain in the following:

If the pedestrian avoidance was "dominant" something we called a "queue" formed. It starts of by the pedestrians nearest to the targets moving towards it and pedestrians behind them filling the void that they left. In order to hold the distance pedestrians move as far to the inner side as possible and "hug" the wall. Sometimes there are more queues forming than the one at the inner wall. But generally speaking pedestrians that are on the outside will prefer to use the opportunity and en queue in the "inner lane", if a possibility unfolds. Possibilities arise when others make space in order to hold their appropriate distance, or the leaders advance further and leave potential space behind them. All in all a lot of movement is spent on keeping distance and in comparison little is used on advancing towards the target. Some nodes might even need to move backwards to keep their distance.

The Distance matrix is "dominant" in our setting, so we perceive that there are multiple lanes being used for the horizontal part and also many, but a little fewer for the vertical part. The full width of the starting corridor is used to advance forwards. There never is any backward movement. The nodes therefore reach their target roughly 50 percent faster. This is the weighting distribution we finally implemented and that is shown in the graphics.



(a) t=0s (b) t=2s (c) t=5s (d) t=8s (e) t=11s

Figure 14: A time lapse of different stages of the test with distance dominant cost function

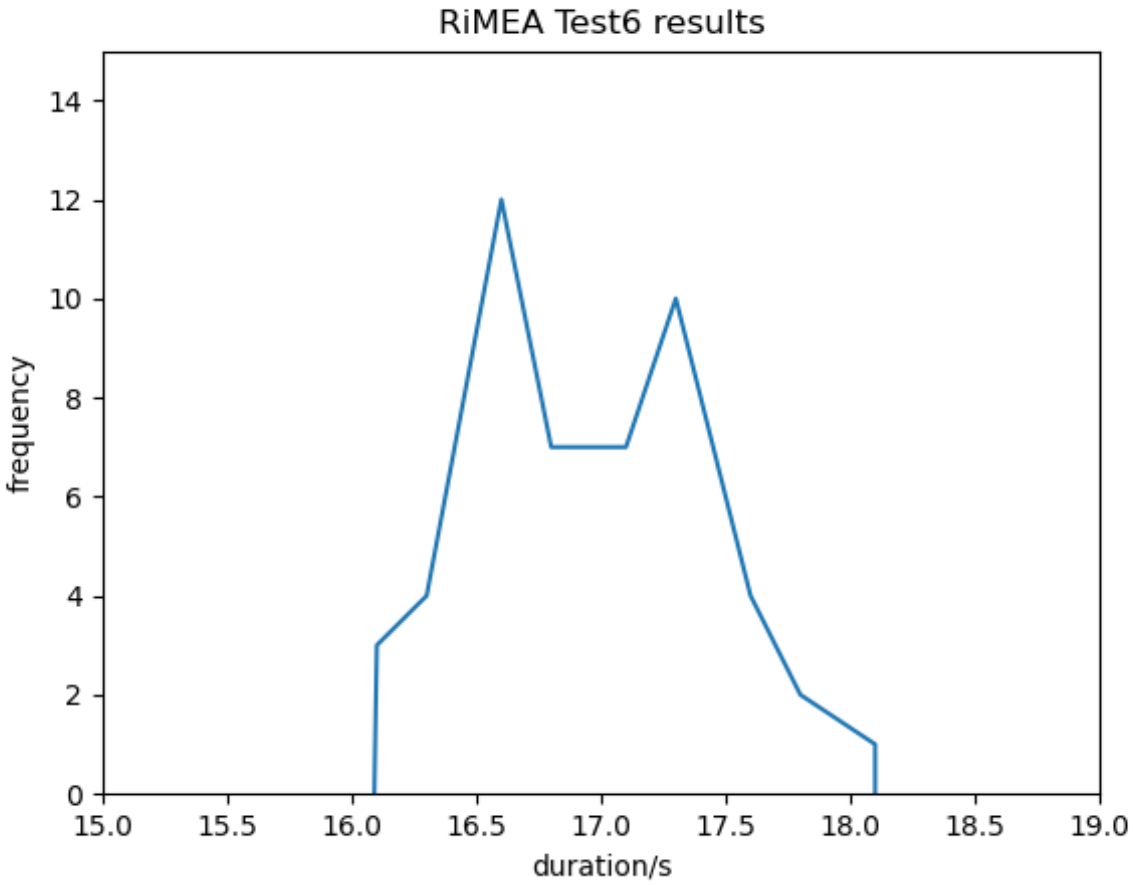


Figure 15: 50 Iterations with random starting distributions

The overall performance with the randomized input leads to results that all lie within a two seconds interval. They are dependent on the randomized input, but yet lead to similar results.

TEST4: RiMEA scenario 7

Task description Assign demographic parameters to the population. Generate a pedestrian sample with size=50 according to the age distribution given by Figure 3 in RiMEA guideline (i.e., normal distri-

No.	Age	Assigned_Speed	Simulated_Speed
1	61	1.2600	1.4121
2	56	1.3350	1.3803
3	25	1.5900	1.6066
4	5	0.6000	0.6516
5	76	0.9000	0.8893
6	64	1.2075	1.1667
7	44	1.4550	2.2273
8	23	1.6050	1.3351
9	53	1.3600	1.0316
10	5	0.6000	0.5968
11	38	1.5000	1.5756
12	34	1.5225	1.5806
13	24	1.5975	1.7626
14	67	1.1550	1.0746
15	57	1.3200	1.3462
16	53	1.3600	1.5312
17	48	1.4200	1.5909
18	79	0.7500	0.7562
19	20	1.6000	1.5312
20	80	0.7000	0.6844
21	50	1.4000	1.4412
22	80	0.7000	0.6703
23	60	1.2750	1.1980
24	80	0.7000	0.6941
25	47	1.4300	1.2564

No.	Age	Assigned_Speed	Simulated_Speed
26	78	0.8000	0.7753
27	67	1.1550	1.1502
28	30	1.5525	1.2861
29	24	1.5975	1.5858
30	62	1.2425	1.2039
31	46	1.4400	1.6897
32	63	1.2250	1.2039
33	51	1.3900	1.3425
34	75	0.9500	0.9839
35	61	1.2600	1.4203
36	25	1.5900	1.4759
37	72	1.0675	0.9646
38	28	1.5675	1.2250
39	78	0.8000	0.7815
40	43	1.4625	1.2827
41	7	0.8000	0.9459
42	49	1.4100	1.0338
43	19	1.5800	1.4203
44	45	1.4500	1.6118
45	50	1.4000	1.0294
46	56	1.3350	1.0359
47	68	1.1375	1.1213
48	28	1.5675	1.7314
49	80	0.7000	0.6960
50	39	1.4925	1.1779

Table 2: An example output from one run of RiMEA Test 7 pipeline

bution with mean=50 and standard deviation=20). Then assign corresponding speed to each pedestrian according to their age, where the speed-age relation is given by RiMEA guideline Figure 2.

Test goal Show that our model can assign demographic parameters to the pedestrians, and the simulated results are consistent with RiMEA guideline Figure 2.

Solution To generate a random sample for the normal distribution, our program used the random.normal function in numpy library. Then we round up the generated age into integers and assign them to the pedestrian sample. To assign the speed according to the pedestrian's age, we referred to Figure 2 in RiMEA guideline(Figure 17(a)). Since the guideline does not provide a table form of the speed-age relationship, we manually read Figure 2 of RiMEA guideline into a table (Table 4), and store it into "speed_age_config.txt" at the root of our GitHub repository together with our source code. Using this table, we are able to generate the speed from the normal distribution for each given age.

To test the simulated speed of the pedestrians, our program aligned them to the left hand side of a track (50m*16.67m), and let them run to the other hand (Figure 16(a)). Consumed time for each pedestrian to reach the target is counted to calculate the simulated speed.

To show that our model can simulate the speed-age relationship correctly, by clicking the button "Test 7" in our GUI, the RiMEA Test 7 pipeline will launch 10 runs of such race. The program would collect the simulated speed of 50*10 demographically parameterized pedestrians and output their information into text files in the folder "./output/". Then we could visualize the speed-age relationship in our simulation using matplotlib according to the simulated results (Figure 17(b)).

Results Firstly, Figure 16 shows an example run of our test. Although the pedestrians start running on the same line, some are soon running ahead of others. This illustrates that our model can handle pedestrians with different speed.

Table 2 shows the output of one example run of RiMEA Test 7 of our program. In this output, the simulated speed varied around the mean speed given by Figure 2 from RiMEA guideline (i.e. the column Assigned_Speed). This illustrates that our model assigned the speed according to RiMEA guideline correctly. After 10 runs of the RiMEA Test 7 pipeline, we can compute the average speed for a different age group (Table 3) using those 500 pedestrians' simulated speed. Meanwhile, Figure 17(b) gives visualization for the simulated speed-age relation in our model. It provides strong evidence that our model can assign the demographic parameters correctly, since Figure 17(b) shares the same shape as Figure 2 of RiMEA guideline.

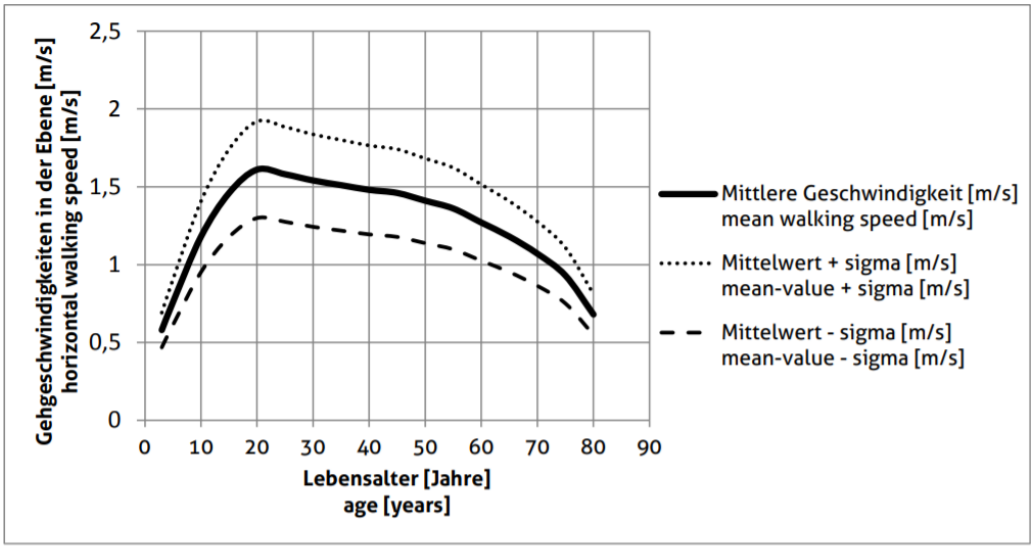
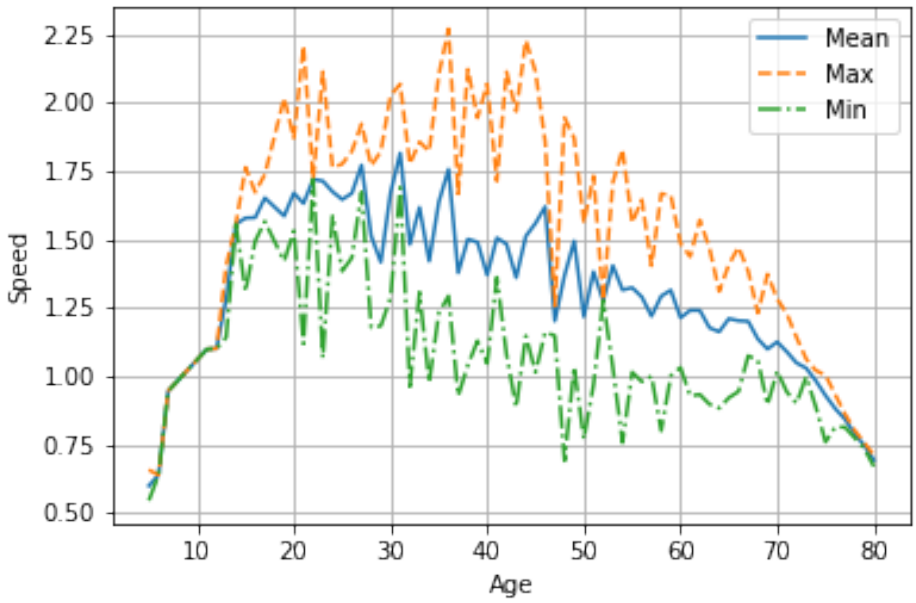


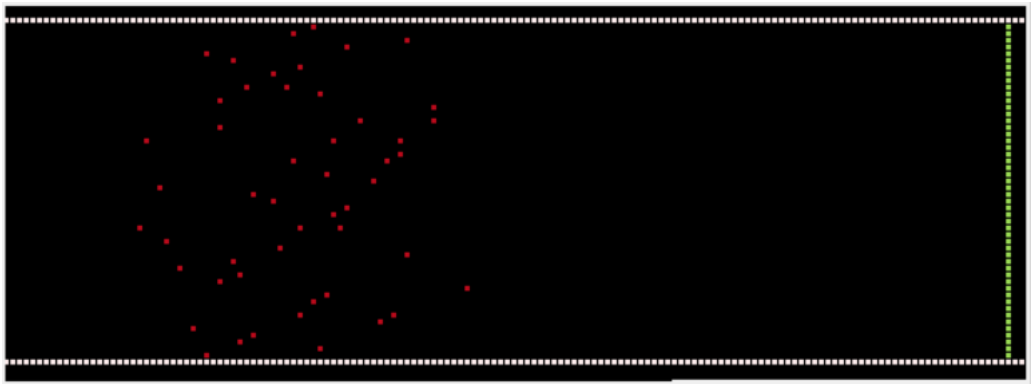
Abbildung 2: Gehgeschwindigkeit in der Ebene in Abhängigkeit des Alters in Anlehnung an Weidmann [2]
Figure 2: Walking speed in the plane as a function of age based on Weidmann [2]

(a) RiMEA guideline figure 2

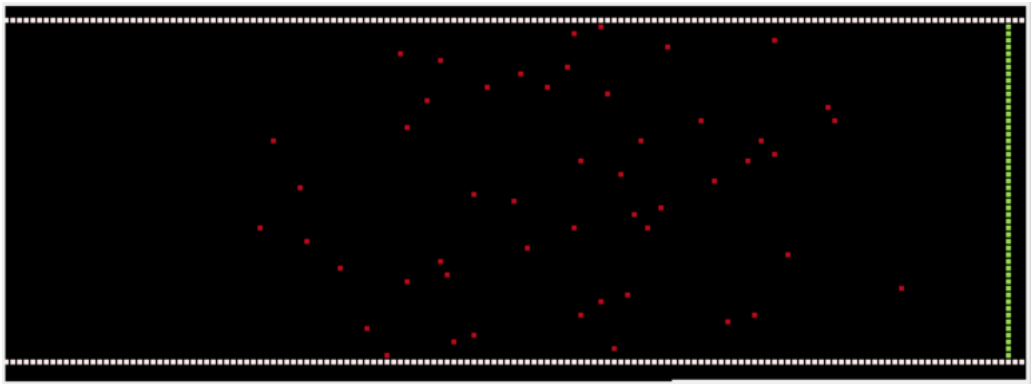


(b) Simulated speed against age relation using 10 runs of RiMEA test 7 pipeline and 500 pedestrians in total

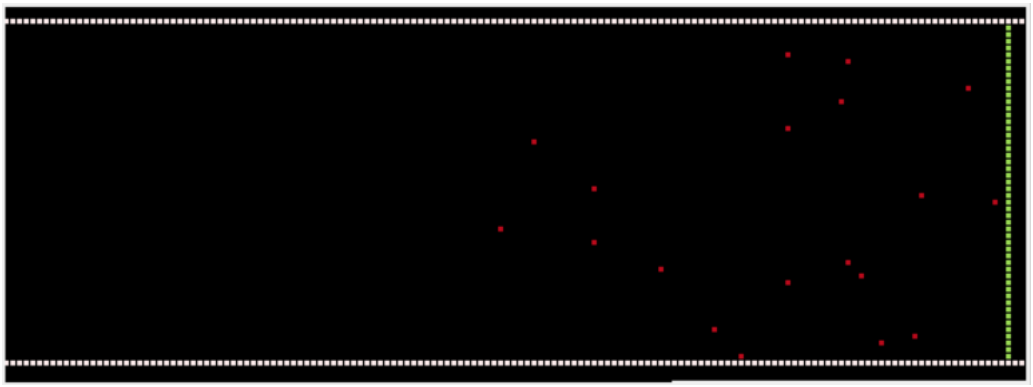
(a) t=0s: 50 demographically parameterized pedestrians line up and ready to run



(b) t=10.0s



(c) t=20.0s



(d) t=40.0s

Figure 16: An example run of RiMEA Test 7

Age	<30	>=30 and <50	>=50
Mean Speed (m/s)	1.4445	1.5175	1.1174

Table 3: Average speed for different age group

Age	Mean Speed(m/s)	Standard Deviation
5	0.6000	0.05
6	0.7000	0.06
7	0.8000	0.07
8	0.9000	0.08
9	1.0000	0.09
10	1.1000	0.1
11	1.1800	0.11
12	1.2600	0.12
13	1.3200	0.13
14	1.4000	0.15
15	1.4800	0.17
16	1.5200	0.19
17	1.5400	0.21
18	1.5600	0.24
19	1.5800	0.27
20	1.6000	0.30
21	1.6200	0.30
22	1.6125	0.30
23	1.6050	0.30
24	1.5975	0.30
25	1.5900	0.30
26	1.5825	0.30
27	1.5750	0.30
28	1.5675	0.30
29	1.5600	0.30
30	1.5525	0.30
31	1.5450	0.30
32	1.5375	0.30
33	1.5300	0.30
34	1.5225	0.30
35	1.5150	0.30
36	1.5075	0.30
37	1.5000	0.30
38	1.5000	0.30
39	1.4925	0.30
40	1.4850	0.30
41	1.4775	0.30
42	1.4700	0.30
43	1.4625	0.30

Age	Mean Speed(m/s)	Standard Deviation
44	1.4550	0.30
45	1.4500	0.30
46	1.4400	0.30
47	1.4300	0.30
48	1.4200	0.30
49	1.4100	0.30
50	1.4000	0.30
51	1.3900	0.29
52	1.3800	0.28
52	1.3700	0.27
53	1.3600	0.26
54	1.3550	0.25
55	1.3500	0.25
56	1.3350	0.24
57	1.3200	0.23
58	1.3050	0.22
59	1.2900	0.21
60	1.2750	0.20
61	1.2600	0.19
62	1.2425	0.18
63	1.2250	0.17
64	1.2075	0.16
65	1.1900	0.15
66	1.1725	0.14
67	1.1550	0.13
68	1.1375	0.12
69	1.1200	0.11
70	1.1025	0.10
71	1.0850	0.09
72	1.0675	0.08
73	1.0500	0.07
74	1.0000	0.06
75	0.9500	0.05
76	0.9000	0.04
77	0.8500	0.03
78	0.8000	0.02
79	0.7500	0.01
80	0.7000	0.01

Table 4: Translated table from Figure 2 in RiMEA Guideline