Report for exercise 1 from group F

Tasks addressed:	5
Authors:	Hao Chen (03764817) Yang Cheng (03765398) Jianzhe Liu (03751196) Pemba Sherpa (03760783)
Last compiled:	2023–05–01
Source code:	https://github.com/ThePembaSherpa/TUMPraktikumCrowdSim.git

The work on tasks was divided in the following way:

Hao Chen (03764817) **Project lead**	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Yang Cheng (03765398)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Jianzhe Liu (03751196)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
Pemba Sherpa (03760783)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%

Report on task TASK 1, Setting up the modeling environment

The model environment and the simulation setup is implemented using various python and tkinter [1] packages. The project is split into 3 python parts - MainGui, Gui, Scenario. Each classes and their functions contains a detailed description of the inputs, outputs and their usage.

The **Gui** class contains all the Gui functions which initialize the tkinter and geometry with the grid, titles and the interactive buttons. It also provides a startGui function, which can create a new scenario object and is attached to the canvas which displays the simulation. The **Scenario** class is aim to define settings and parameters of all the elements in the simulation and the method of how it update the whole simulation or the time steps. This class also includes the definition of pedestrains, for example the speed of the pedestrains, and their act when meeting other neighbor pedestrians [2] or obstacles. At last, The **Main GUI** part acts as the main body of the program, which ensures the running of the whole program.

Figure 1: Cellular Automation GUI

As shown in Figure 1 above, the cellular automaton GUI contains various buttons on the top side to manage the simulation. For the basic Visulatization, the base GUI contains an empty grid with interactive buttons on the top for making changes in the simulation environment. In addition, some functions are included in the menu interface named **Simulation** in the upper left corner. The function of the buttons are first to introduced:
The RUN button can be used to execute the simulation until the pedestrian reaches the target or cannot progress any further. Once it's clicked, the progress will automatically run towards the end, can only be stopped by the reset button.
The STEP button on the other hand, can only forward the simulation by one time-step. Users can stop the simulation at a certain point in the process by stopping clicking the STEP button at any time. By using step button, it is also possible to forward the simulation even if it's over.
The RESET button has a function to restore the GUI interface to the scene before the simulation started. Whether it is a preloaded json file or a scene with manually added parameters, as long as it's clicked during the simulation or after the simulation, the scene will return to its original version. It's also worth to mention that if the simulation has not started, the reset button will be disabled.
The METHON button allows the user to choose which algorithm to use to simulate the process of pedestrians walking towards the target. The two algorithms provided by this program are Euclidean's algorithm (basic) and Dijkstra's algorithm.

It can be noticed that besides the four basic function buttons mentioned above, there are 2 additional buttons on the GUI interface. This is because during the program implementation, most of the scene are realized by loading json files. But scenario4 requires to randomly generate a large number of pedestrians, which makes it hard to write json files. So here we use the preset parameters to define the different densities of scenarios4.

As for the menu interface, as shown in Figure 2, there are two functions: editing and loading.

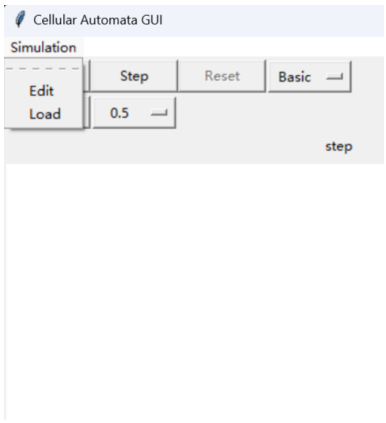

Figure 2: Menu Function

Edit allows user to add targets, obstacles and pedestrians in cells. Users can add targets and obstacles by entering x and y coordinates in the text boxes. Additionally, When adding a pedestrian, not only the coordinates but also the speed of the pedestrian is needed (in Figure 3). And if user wants to delete an element, you only need the corresponding left side, click the remove button, and the element will be removed.

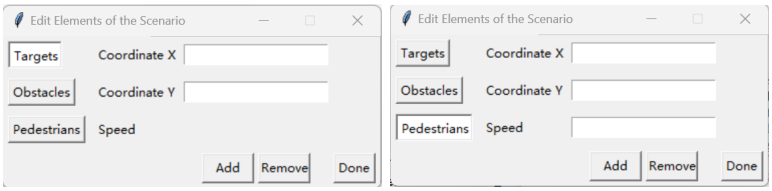

Figure 3: Entering Coordinates

In terms of visualization, our color definitions for targets, obstacles and pedestrians follow the definitions in ExampleGUI, that is, 'EMPTY' is white: (255, 255, 255), 'PEDESTRIAN' is red: (255, 0, 0), 'TARGET' is blue: (0, 0, 255), 'OBSTACLE' is pink: (255, 0, 255).(in Figure 4)

Figure 4: Color of the Elements

Another function **Load** allows users to directly load the pre-edited Json file from the GUI interface, as shown in the figure 5 . After selecting different json files in the directory, the program can run any scenario in task 2-5 (except scenario4 in Task 5).

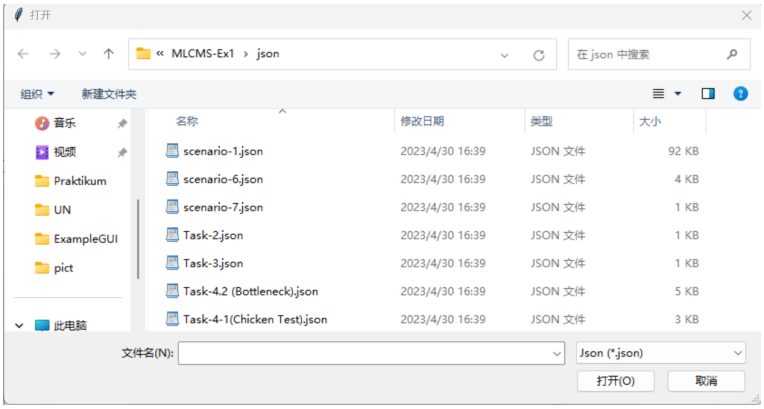

Figure 5: Json Files

And as mentioned above, after loading a scene and performing simulation, clicking the reset button will reset the scene. If user wants to change the scene, then it's needed to click **load** in the drop-down menu again, and select another scene.

Report on task TASK 2, Simulation

In this task, a 50 by 50 cells sceranio is created with a single pedestrian and a target. The pedestrian is placed at position (5,25) with the target at position (25, 25).

According to the requirements of the topic, the pedestrain must take the shortest path to the target and in this case, the pedestrian will walk 20 steps along the x-axis to reach the target. As show in Figure 6 below, this scenario is simulated by the celular automaton using either the run method or step method for 25 times. The run method contains a loop in the backend which runs the step method 25 times. On the other hand, the step method only forwards the scenario by one time step. In both case, the pedestrian reaches the target at 20 time steps and stays there remains there for the remaining time steps.

Figure 6: Movement process of Task2

Report on task TASK 3, Interaction of pedestrians

In this task, five pedestrians are asked to be placed on a circle, and their distance from the target at the center of the circle is the same. When the pedestrian is moving forward, the pedestrian will move forward in the way closest to the target. To be precise, when the pedestrian and the target are on the same horizontal or vertical line, the pedestrian will advance a distance of 1 unit. In addition, pedestrians will update to $\sqrt{2}$ distance units along the diagonal direction.From this we find that the step size is either 1 (integer) or $\sqrt{2}$ (irrational number).because $k \cdot \mathbf{Z} \notin \mathbf{I}$, we cannot make pedestrians with different step sizes advance the same distance in integer time.

Therefore, we can only simulate it approximately, so that pedestrians in different forward ways can advance the same distance as much as possible in the same time. Here, we adopt a rule: advance 1 distance per second, and the distance that pedestrians can advance cannot exceed the allowed distance. For example, a pedestrian walking in a straight line, one distance is advanced per second, and 1 distance is allowed to be advanced per second, so this pedestrian update 1 distance every second. Another example,a pedestrian walking diagonally, he advances $\sqrt{2}$ distances per second, but he is only allowed to advance 1 distance in the first second, so he does not move in the first second.In the second second the distance allowed to advance is 2, so this pedestrian advances one space diagonally with $\sqrt{2}$ distances. In the third second the distance allowed to advance is 3, so advance one space diagonally again with $2\sqrt{2}$ distances.

In Task3, we set the positions of Target and Pedestrian 1-5 according to the Pythagorean theorem. Make the distance from each Pedestrian to Target be 26. Refer to Table 1 for specific location distribution. In this task, we edit the task-3.json file to store the position.We could open it with Simulation- Load - task-3.

Target	(30,30)
Pedestrian 1	(20,54)
Pedestrian 2	(6,20)
Pedestrian 3	(54,20)
Pedestrian 4	(20,6)
Pedestrian 5	(40,54)

Table 1: Positions of Task3

In that way we could edit simulate the process of task3 . As the result shown in Figure 3,we could find that the pedestrians reach the target at the same time.

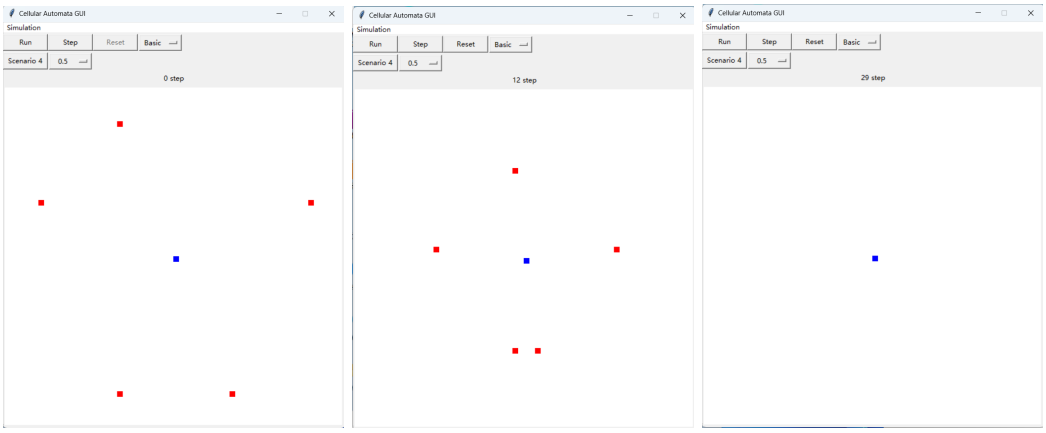

Figure 7: Movement process of Task3

Report on task TASK 4, Obstacle avoidance

In this task, we are asked to simulate how pedestrians will move to reach targets considering obstacles. We first have to answer: How will pedestrians move when they adopt a rudimentary obstacle avoidance strategy (that is, add a large cost function to the Euclidean distance of the obstacle)? My answer is: There are two possibilities, one is that the pedestrian will bypass the obstacle, and the other is that the pedestrian will be blocked by the obstacle. As for how will the pedestrian move in the end, it depends on the obstacle. In order to illustrate this, first, let's assume that at least one of the obstacles has the shortest Euclidean distance to the goal among all the pedestrian's neighbors (because if none, the obstacle has no effect at all, after all, the pedestrian will choose the position of this neighbor who is not an obstacle but has the shortest Euclidean distance to the target as his next position). Now, we want to ask, among the pedestrian's neighbors except for obstacles, is

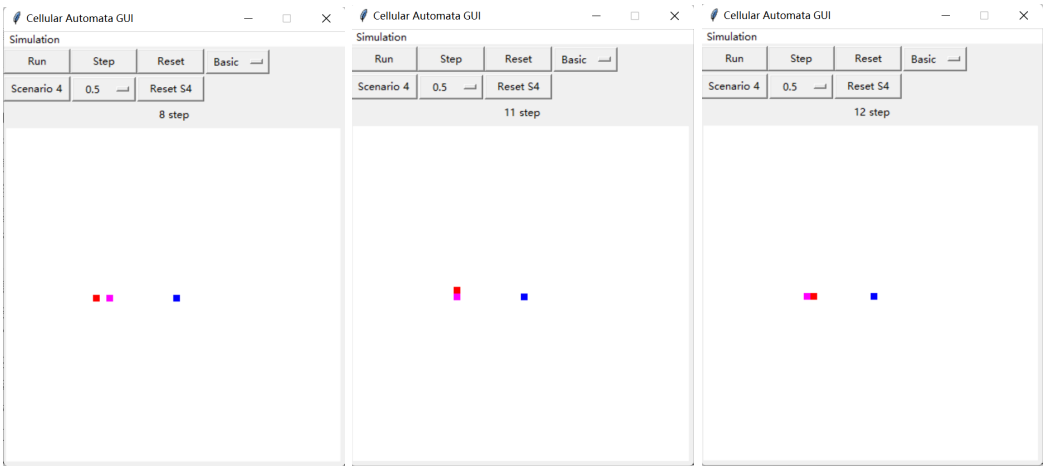

Figure 8: The movement of pedestrian of the first possible scenario

there any neighbor whose Euclidean distance to the target is smaller than the distance from the pedestrian's own location to the target? If there is, then the pedestrian will choose the position of this neighbor as his next position. Then the pedestrian will eventually bypass the obstacle and reach the target, as shown in Figure 8. If not, the pedestrian will choose to stay where he is now (from another perspective, the pedestrian is blocked by the obstacle), as shown in Figure 9.

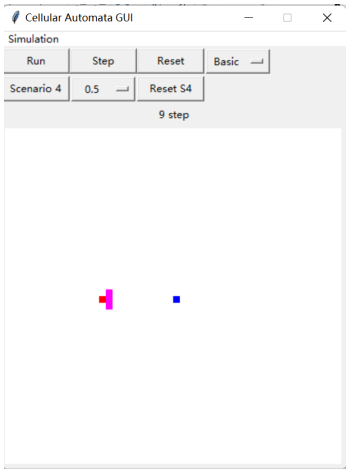

Figure 9: The movement of pedestrian of the second possible scenario

Now, let's answer the next question of this task: when the pedestrian does not adopt any obstacle avoidance strategy, how will the pedestrian move in the "bottleneck" and "chicken test" scenarios respectively. To this end, we designed these two scenarios in a 100x100 grid, respectively, and placed 150 pedestrians in a specific area in the two scenarios for simulation respectively, as shown in Figure 10 and Figure 11.

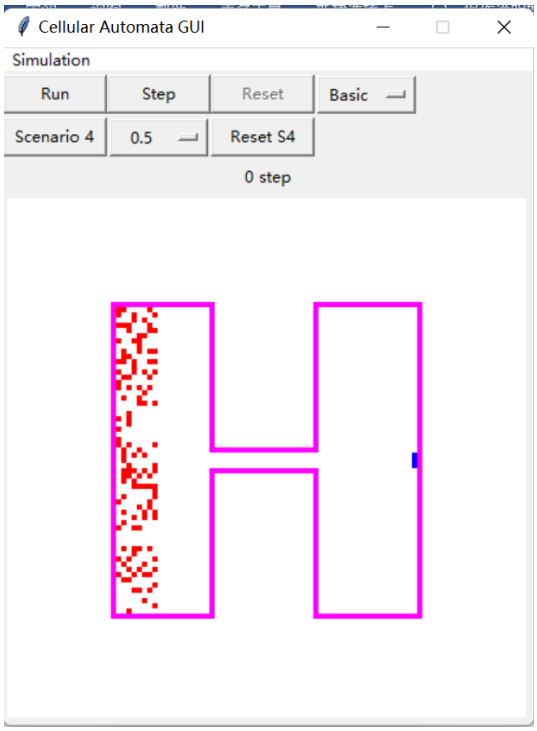

Figure 10: Bottleneck scenario

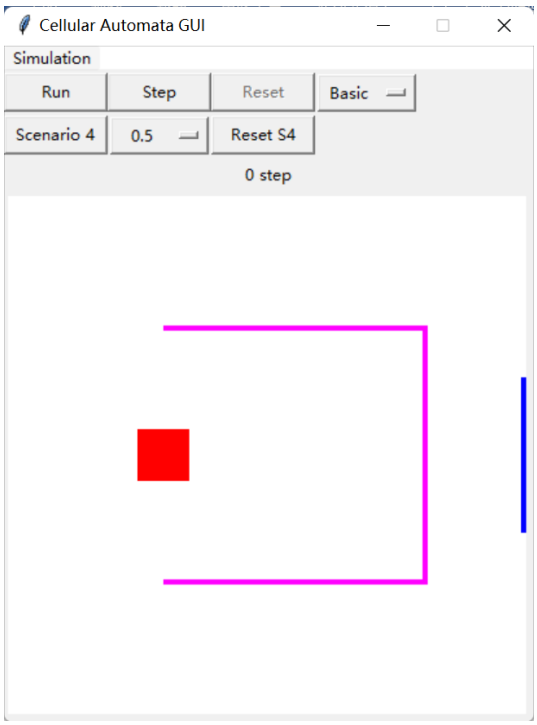

Figure 11: Chicken Test scenario

The simulation result is (as shown in Figure 12):

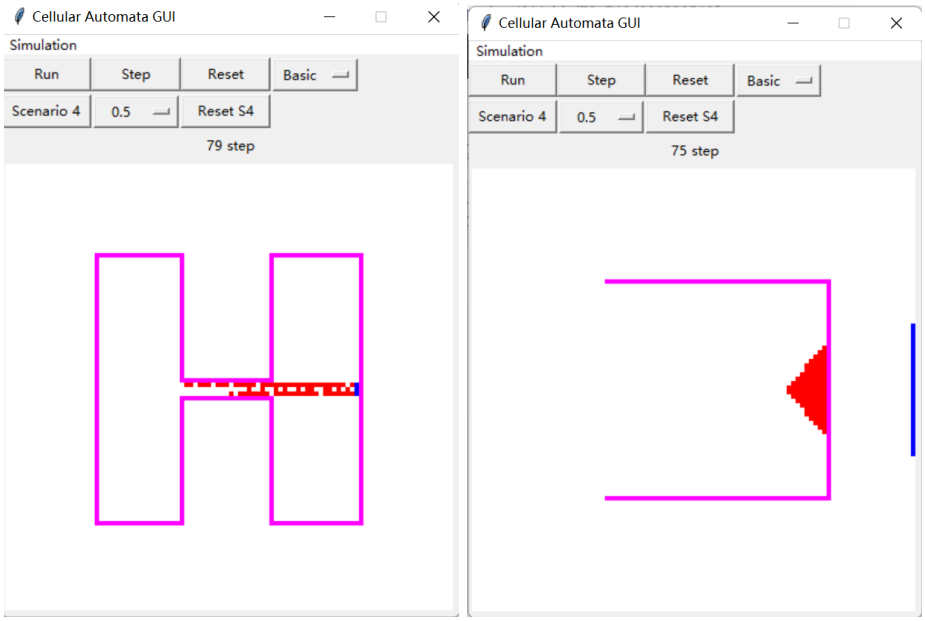

Figure 12: Different behaviors of pedestrians when no obstacle avoidance strategy is implemented

In the "bottleneck" scenario, 150 pedestrians will reach their destination through a narrow passage. In contrast, all 150 pedestrians in the "chicken test" scenario were trapped in the U-shaped barrier.The reason why pedestrians can pass through the "bottleneck" is that the empty grids at the entrance of the narrow passage have a smaller Euclidean distance to the target than obstacles, and pedestrians can occupy these empty grids in turns to enter the narrow passage and eventually reach the target. The reason why pedestrians cannot pass the U-shaped obstacles is that the obstacles occupy those grids with smaller Euclidean distance to the target, so pedestrians will choose these obstacles as their next positions, but obstacles are different from empty grids, because they cannot be reached by pedestrians, so pedestrians can only stay in place, therefore there is a scene where pedestrians are trapped in U-shaped obstacles. How to help pedestrians trapped in U-shaped obstacles to reach their goals? We are going to use the Dijkstra algorithm, the main idea of the algorithm is as follows:

1. Initialize sets S and Q. Add the source vertex to set S, and add all other vertices to set Q.
2. Initialize the distance of the source vertex to 0 and the distance of all other vertices to positive infinity.
3. While set Q is not empty:
   (a) Select the vertex u with the minimum distance in set Q, and add u to set S.
   (b) For each neighbor v of u:
      i. Calculate the distance d from the source vertex to v via u.
      ii. If d is less than the current distance of v, update the distance of v to d.
4. Return the shortest distance to each vertex.

With this algorithm, the pedestrian can pass the chicken test, as shown in Figure 13,because the pedestrian chooses the next position not according to the Euclidean distance between all neighbors and the target, but the minimum distance between all non-obstacle neighbors and the target, he will choose a minimum among all the minimum distances, and then update his position. 14,

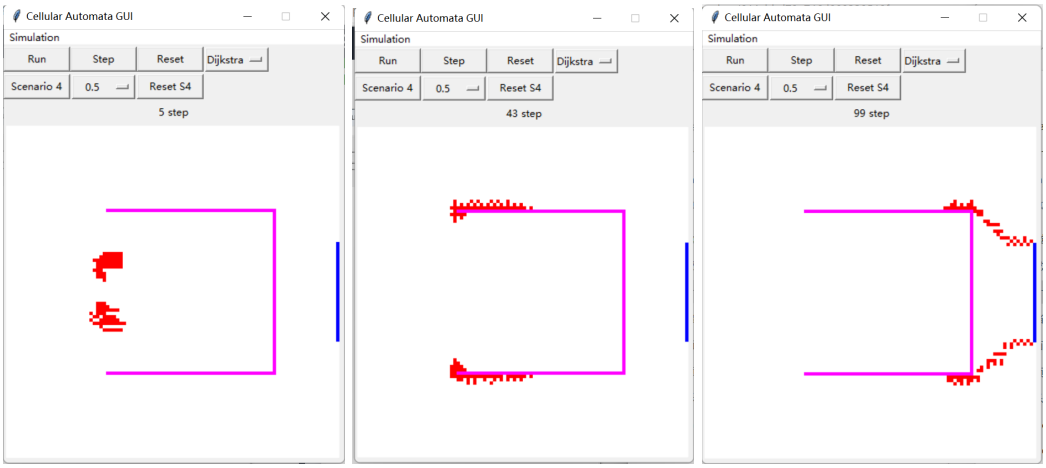

Figure 13: Pedetrians pass the Chicken Test with Dijkstra algorithm

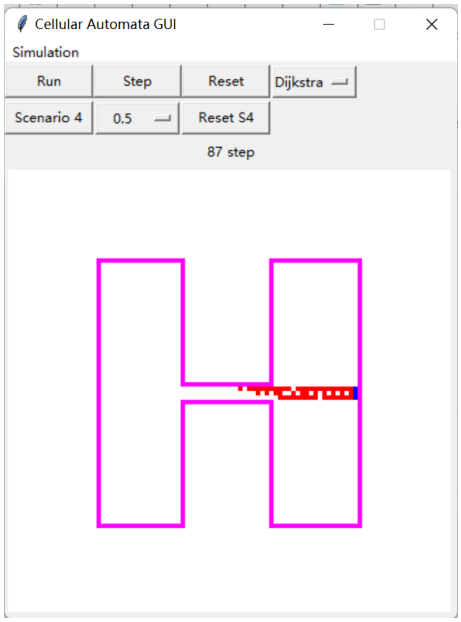

Figure 14: Pedestrians pass the bottleneck with Dijkstra algorithm

In the "bottleneck" scenario, can pedestrians use the Dijkstra algorithm to reach the target point? The answer is obvious, he can, as shown in Figure and compared to the scenario of not taking any obstacle avoidance strategy, using the Dijkstra algorithm can avoid traffic jams as much as possible (because pedestrians will try to go to other empty grids that are closer to the target than themselves at every step, instead, if no obstacle avoidance strategy is adopted, pedestrians will stop in front of obstacles, which causes traffic jams), so pedestrians can reach their targets faster and more smoothly.

Report on task TASK 5, Tests

TEST1: RiMEA scenario 1 ((straight line)

The RiMEA [3] scenario 1 describes a single pedestrian with 40 cm body dimension located in the beginningnof a 2m wide and 40m long corridor, and wants to show that a pedestrian with a defined walking speed will cover the distance in the corresponding time period. To solve this one, we first set up some basic parameters. Since the corridor is 40 meters long and 2 meters wide, and the size of the human body is 40 cm, it can be concluded that if the pedestrians are placed in one pixel, then the width of the corridor should be 5 pixels, and the length should be 100 pixels. As required in RiMEA scenario 1, we should set the speed of the pedestrian to 1.33m/s, which is 3.33 pixels per step. From here we can easily calculate that the estimated time of the whole process should be 31 steps.

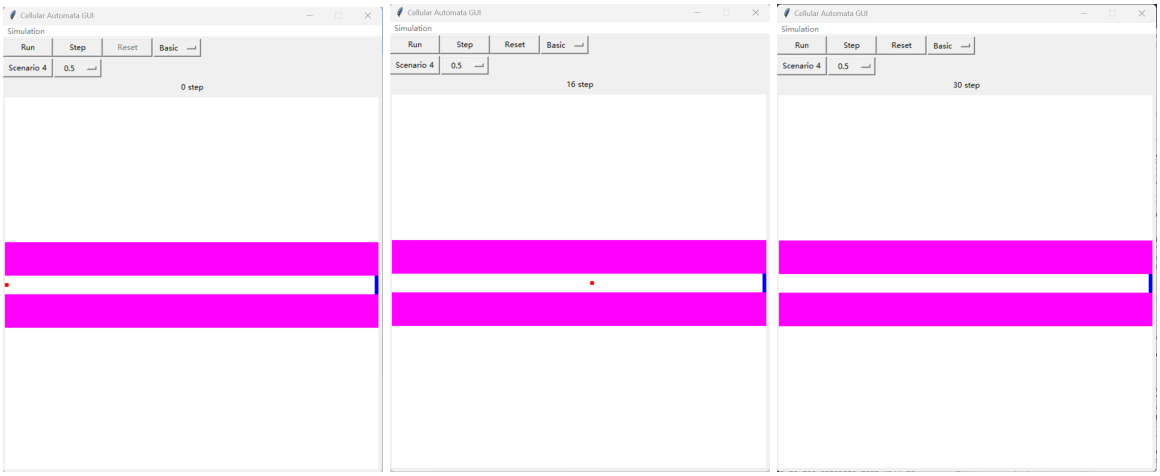

Figure 15: Movement of the Pedestrian in given Speed

It can be seen from the figure 15 that the pedestrian has already walked half of the corridor around the 16th step, and has reached the target at the 31st step. This shows that the program has successfully simulated Scenario 1.

TEST2: RiMEA scenario 4 ((fundamental diagram)

The RiMEA scenario 4 originally describes a corridor with 1000 m long, 10 m wide, and it is to be filled with different densities of persons with an equal as possible free walking speed (for example 1.2 - 1.4m/s): the density ranges from 0.5P/m^2 to 6 P/m^2. But 1000m will be too long for its realization in this GUI, and the scale of the pedestrian and measuring points will be too small compared to that. So we rescale the scene to be 300 pixels long, where we take 3 pixels as 1 meter, so the whole corridor will be 100m in total. According to this, the average speed of the pedestrians will be 3.9 pixels per step.

Figure 16: Different Density of Pedestrians

As shown in the Figure 16, after selecting different density parameters, the user can click the scenario4 button to generate the corresponding crowd in the GUI interface. It's also to be seen that after about 77 steps, the pedestrian who stands exactly at the beginning of the corridor can finish the whole progress(in Figure 17).

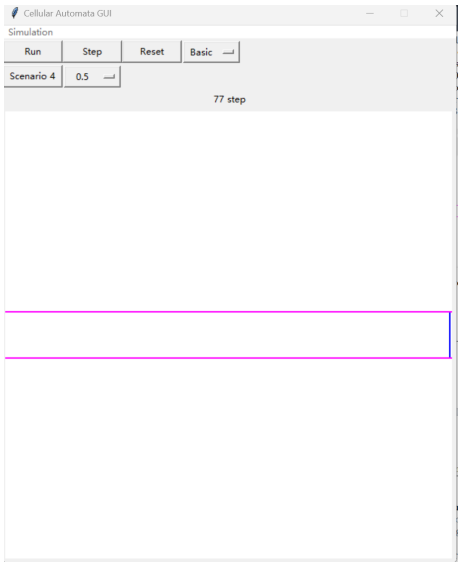

Figure 17: Steps to go across the Corridor

Next, in order to show the result of the calculation of the flow, they are listed in a table.

Density(P/m^2)	Speed(pixel)	Flow(P/pixel*step)
0.5	3.9	0.217
1	3.9	0.433
2	3.9	0.867
3	3.9	1.300
4	3.9	1.733
5	3.9	2.167
6	3.9	2.600

Table 2: Calculation of the Flow

TEST3: RiMEA scenario 6 (movement around a corner)

RiMEA Scenario 6 describes a group of 20 pedestrians walking along a left-turning corner, and eventually they will successfully reach their destination. The goal of this scenario is to show that the pedestrian can avoid the obstacle (the wall) and reach the destination. The result of the test is that no matter whether Dijkstra's algorithm is adopted or no obstacle avoidance strategy is adopted, the pedestrian will reach the destination bypassing the corner, but the route taken by the pedestrian in these two scenarios is quite different. When pedestrians do not adopt any obstacle avoidance strategy, they will try to walk along the wall as much as possible, because the road closer to the wall means the shorter the walking distance required to reach the goal,as shown in Figure 18

Figure 18: Pedetrians walk along the wall with no obstacle avoidance strategy

When pedestrians adopt the Dijkstra algorithm, pedestrians who are far away from the wall at the beginning will not squeeze in, and they will try to keep on their own tracks, and only when cornering, some pedestrians who were originally in the outer tracks will appear to walk closer to the wall,as shown in Figure 19

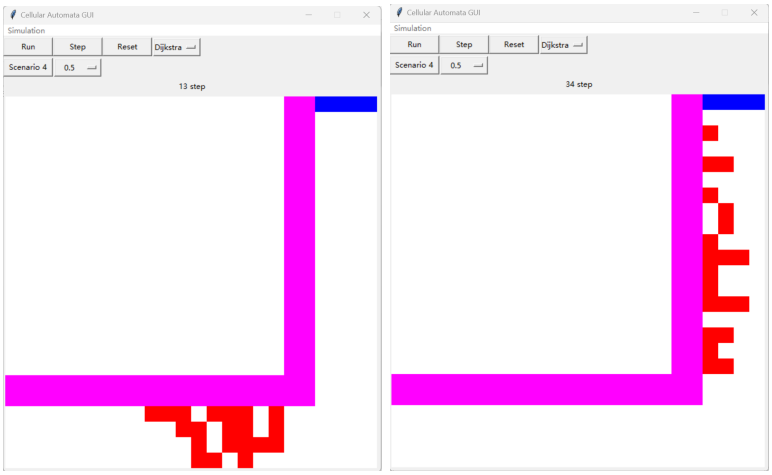

Figure 19: Pedetrians keep on their own track with Dijkstra algorithm

TEST4: RiMEA scenario 7 (demographic parameters)

In scenario , 50 pedestrians aging from 3 to 80 are simulated to go forward with their own different speed. According to the age distribution, we could In scenario 7, 50 pedestrians aging from 3 to 80 are simulated to go forward with their own different speed. We could estimate the walking speed in different age groups with Figure Test4-1. Because we could not get the crisp function, we need to divide the age into several small age groups. Besides, the corresponding speed of each age group can also be roughly estimated.In that way, We divided 5 age groups and determined their intervals. The age of pedestrians will be randomly assigned, and their corresponding speed will also be randomly generated according to the speed range.

age group	speed range	expected speed
3-10 years old	(0.6,1.1)	0.90
11-20 years old	(1.1,1.6)	1.30
21-50 years old	(1.6,1.3)	1.5
51-70 years old	(1.3,1.1)	1.2
71-80 years old 5	(1.1,0.7)	0.8

Table 3: Speed Range

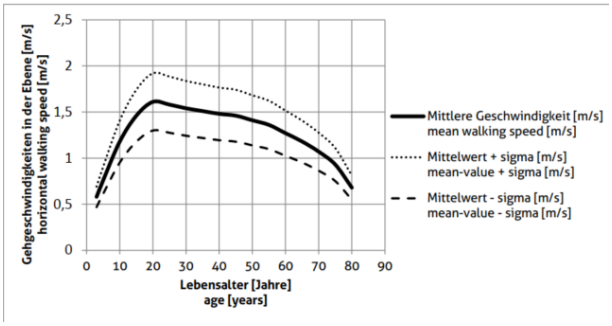

Figure 20: Walking speed in the plane as a function of age based on Weidmann [3]

According to the scenario 7 requirements, we are setting up a 50×50 scene. We place all pedestrians on the far left and objects on the far right. We need to simulate and calculate the measured speed.We could observe that, the pedestrians walks forward with different speed. While some faster points refer to middle-aged man, the slower points refer to old people or babies.

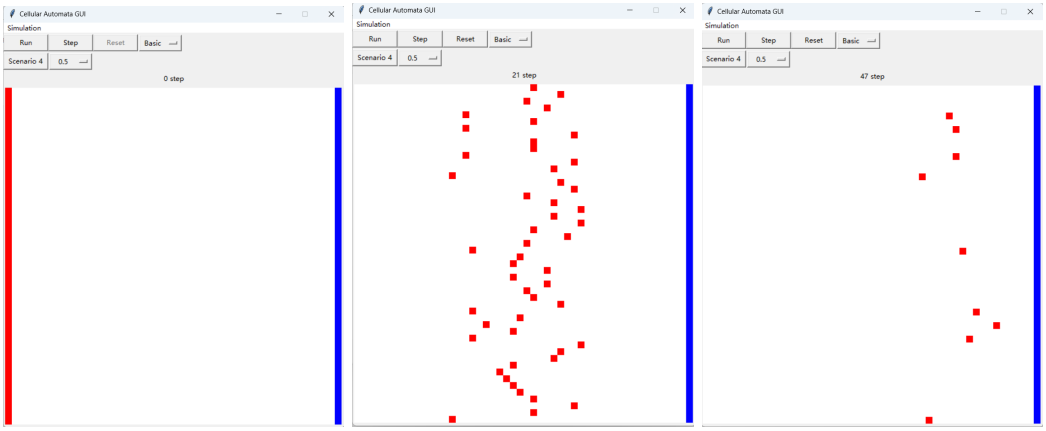

Figure 21: Selected scenes from scenario 7

In the following part, we want to prove that the measured speed is almost same as the expected. According to the Figure 20, we could roughly estimate the expected speed in every age groups as their mean speed. During the simulation , we use the step button the update the pedestrians step by step. When one reach the target , we record its ID number and the corresponding taken time. Because the total distance is 50m, we could easily calculate the measured speed by $Speed = \frac{Distance}{Time}$. In that way we obtain each pedestrian's measured speed and expected speed. As the result in Table4 , could find out that the difference is quite small.

The difference between the measured speed and expected speed results from the simulation step. First, because the expected speed is roughly read from the curve, the expected speed might be a little inaccurate.Meanwhile the step we taken here is 1 second, so that the for some pedestrians with different speed could also reach the target at the same time. For example , pedestrian A with speed 1.3m/s and pedestrian B with 1.2m/s. And their final distance between the target is both 1m. They will reach the target at the same time , also their speed is not the same.

pedestrian	taken time	measured speed	expected speed
No 1	35	1.43	1.45
No 2	43	1.16	1.2
No 3	41	1.22	1.2
No 4	55	0.91	0.9
No 5	42	1.19	1.2
No 6	43	1.16	1.2
No 7	38	1.32	1.3
No 8	45	1.11	1.2
No 9	36	1.39	1.3
No 10	33	1.52	1.5
No 11	60	0.83	0.8
No 12	41	1.22	1.2
No 13	39	1.28	1.3
No 14	69	0.72	0.8
No 15	31	1.61	1.5
No 16	34	1.47	1.5
No 17	39	1.28	1.3
No 18	82	0.61	0.8
No 19	34	1.47	1.5
No 20	37	1.35	1.3
No 21	44	1.14	1.2
No 22	33	1.52	1.5
No 23	41	1.22	1.2
No 24	55	0.91	0.9
No 25	45	1.11	1.2
No 26	34	1.47	1.5
No 27	35	1.42	1.5
No 28	36	1.38	1.3
No 29	56	0.89	0.9
No 30	36	1.38	1.3
No 31	55	0.91	0.9
No 32	33	1.51	1.5
No 33	51	0.98	0.9
No 34	45	1.11	1.2
No 35	38	1.31	1.3
No 36	42	1.19	1.2
No 37	45	1.11	1.2
No 38	42	1.19	1.2
No 39	49	1.02	0.9
No 40	34	1.47	1.5
No 41	38	1.31	1.3
No 42	34	1.47	1.5
No 43	43	1.16	1.2
No 44	39	1.28	1.2
No 45	44	1.13	1.2
No 46	37	1.35	1.3
No 47	49	1.02	0.9
No 48	42	1.19	1.2
No 49	50	1	0.9
No 50	33	1.51	1.5
Average	-	1.18	1.14

Table 4: Compare the measured speed and expected speed

References

[1] Python's tkinter package, https://docs.python.org/3/library/tkinter.html.

[2] Felix Dietrich, Gerta Köster, Michael Seitz, Isabella von Sivers, Bridging the gap: From cellular automata to differential equation models for pedestrian dynamics, Journal of Computational Science, 2014, Pages 841-846, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2014.06.005.

[3] RiMEA. Guideline for Microscopic Evacuation Analysis. 2016. www.rimea.de.