Report for exercise 1 from group I

Tasks addressed: 5
Authors: Yizhi Liu (03779947)
Kejia Gao (03779844)
Jingyi Zhang (03785924)

Last compiled: 2024–05–06

The work on tasks was divided in the following way:

Yizhi Liu (03779947)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%
Kejia Gao (03779844)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%
Jingyi Zhang (03785924)	Task 1	33.33%
	Task 2	33.33%
	Task 3	33.33%
	Task 4	33.33%
	Task 5	33.33%

0 Introduction

In the realm of computational modeling, cellular automata stand out as a powerful tool for simulating complex systems. A cellular automaton is composed of multiple cells that can interact with each other. Each cell has multiple states that can be expressed. For example, Conway's "Game of Life" [2] is a classic, zero-player, cellular automaton. It was proposed by the British mathematician John Conway in 1970. It is a zero-player game that simulates the evolution of life. It includes a two-dimensional grid, and each grid can be in two states: alive or dead.
In this experiment, this concept is applied to the field of crowds. By representing individuals as cells and defining rules for their interactions, obstacles, and goals, we can simulate the complex behaviors observed in real-life crowds. Application to practical applications:

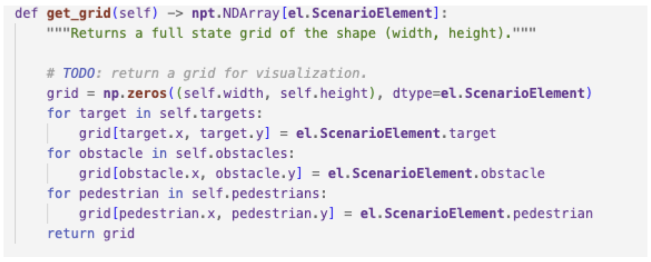
- transportation planning and design: Cellular automata can be used to simulate the movement and interaction of vehicles and pedestrians in urban environments, thereby designing more reasonable transportation routes.
- Emergency evacuation: In emergency situations such as fires and earthquakes, cellular automata can simulate the actions of crowds and help security departments design reasonable escape routes to prevent blockages.

In this report, we start by building a cellular automata, setting up pedestrians, obstacles and targets to simulate how pedestrians move in real conditions. And enhance the authenticity by setting up various traps such as chicken boxes and bottlenecks. In this report, we start by building a cellular automata, setting up pedestrians, obstacles and targets to simulate how pedestrians move in real conditions. And enhance the authenticity by setting up various traps such as chicken boxes and bottlenecks. Through exploration, we aim to understand and compile crowd dynamics, demonstrate the importance of cellular automata in modeling complex systems, and lay the foundation for subsequent approaches to crowd dynamic systems.

1 Task 1

Report on task TASK 1, Setting up the modeling environment

First we need to create a grid containing the locations of pedestrians, obstacles and destinations. The positions of various objects in the grid can be visualized through the GUI and marked with different colors. In order to facilitate observation, in this exercise, pedestrians are set to red, targets are set to blue, and obstacles are set to purple. All simulation steps are carried out in the simulation.py file. Methods such as how to obtain the grid, calculate the distance and update the pedestrian path are defined in the Simulation class. By compiling Simulation.init() and Simulation.get grid(), the visualization effect on the GUI can be achieved.The following figure is a visualization of the toy example file.



```python
def get_grid(self) -> npt.NDArray[el.ScenarioElement]:
    """Returns a full state grid of the shape (width, height)."""

    # TODO: return a grid for visualization.
    grid = np.zeros((self.width, self.height), dtype=el.ScenarioElement)
    for target in self.targets:
        grid[target.x, target.y] = el.ScenarioElement.target
    for obstacle in self.obstacles:
        grid[obstacle.x, obstacle.y] = el.ScenarioElement.obstacle
    for pedestrian in self.pedestrians:
        grid[pedestrian.x, pedestrian.y] = el.ScenarioElement.pedestrian
    return grid
```

Figure 1: get grid() method

The red ones in the picture represent pedestrians, and the yellow ones represent targets. It can be controlled through the provided GUI. The specific functions are:

- Run/Stop the simulation :Clicking on the "Run" button will start the simulation, while clicking on the "Stop" button will pause it.
- Step the simulation :Clicking on the "Step" button will execute a single iteration of the simulation.

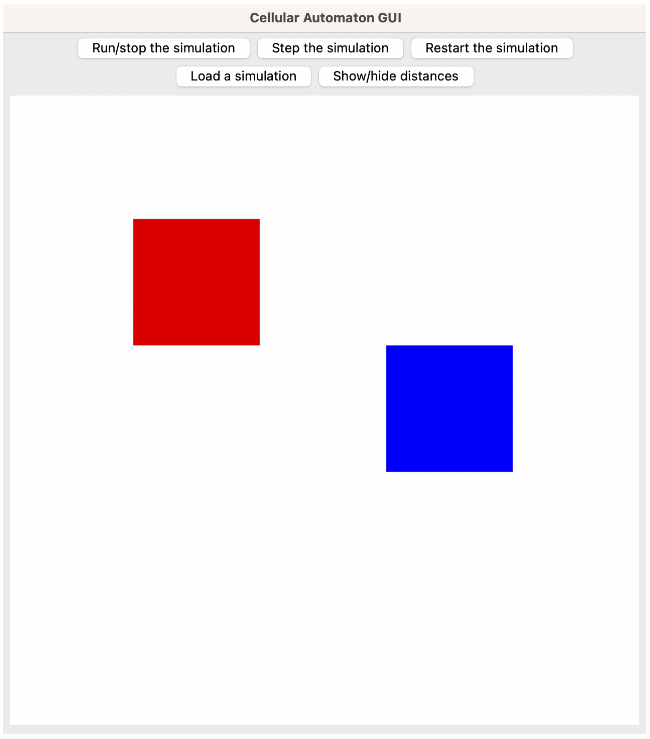


Figure 2: The scenario of Task 1

- Restart the simulation :Clicking on the ”Restart” button will clear any ongoing simulation progress and return all parameters and entities to their starting positions.
- Load a simulation :This function enables users to load a previously saved simulation scenario.
- Show/hide distance :This function toggles the display of distance measurements within the simulation environment.

The toy example file is a json file that sets a specific scene, which specifies the length and width of the grid, the number and location of pedestrians, target points, obstacles, etc. Using this as an example, other scenarios can be generated for subsequent tasks.

2 Task 2

Report on task TASK 2, First step of a single pedestrian

In this task, it needs to be accomplished to move a single pedestrian to a designated target. First generate the corresponding scene, define a grid with a length and width of 50, set the pedestrian's initial position to (5, 25), and set the target to (25, 25). The pedestrian's speed defaults to 1 unit length per time unit. This is expected to reach the target after 25 time units. Compile the corresponding information into a json file, and obtain the initial situation after visualization. In order to complete this task, the following methods need to be used

- `get_neighbors()` method :In this method, the coordinates of any point in the grid can be input, and the coordinates of 8 adjacent points will be returned. This is obtained by adding or subtracting 1 to the x and y of the given coordinates respectively.
- `compute_naive_distance_grid` :In this method, the distance between all points in the grid and the nearest target point is calculated, and a matrix with the same shape as the grid is returned.

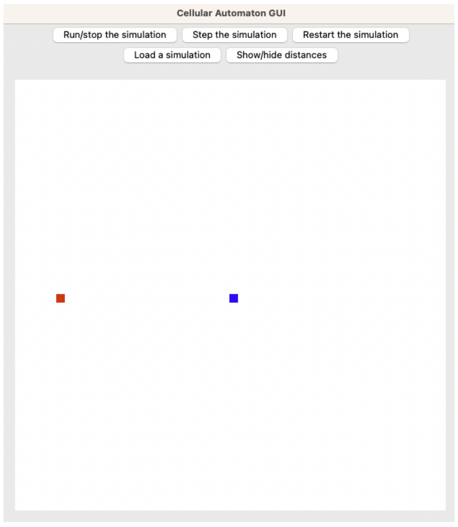
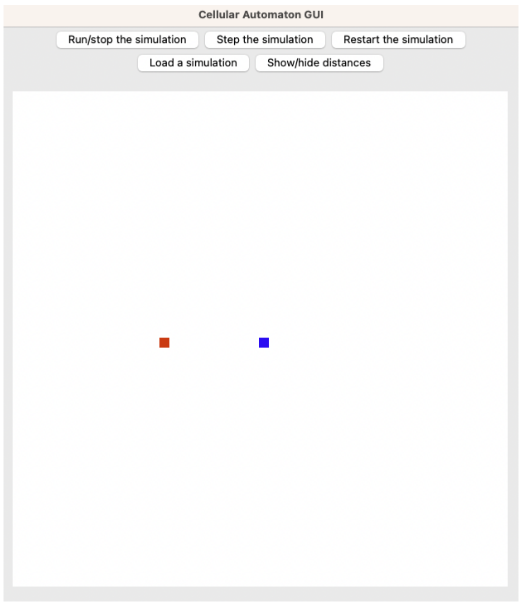


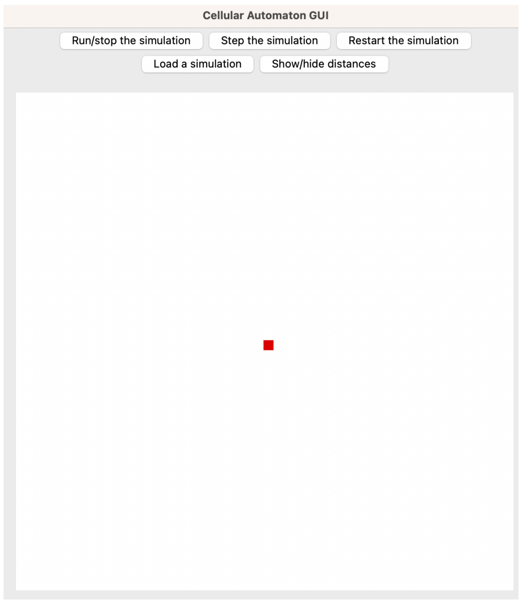
Figure 3: Initial state of task 2

- `update_step`:Through the above two methods, the update of each step of the pedestrian can be completed. The specific idea is to obtain the eight neighbors around the pedestrian based on its position, and then calculate the distance between the eight neighbors and the target point through `compute_naive_distance_grid`. Select the neighbor with the smallest distance and set the next pedestrian's position as the neighbor's position.

After the setting is completed, the pedestrian advances towards the target at a speed of one frame per unit time. After 20 time units, the pedestrian reaches the target.According to the requirements of the question, the pedestrian should be absorbed after reaching the target point, so the method is set to remove the pedestrian when the pedestrian reaches the target point. It can be seen from the visualization results obtained from the GUI that within one time unit after the pedestrian arrives, the color of the target point returns to its original color.



(a) Intermediate state of task 2



(b) Final state of task 2

Figure 4: Task 2

3 Task 3

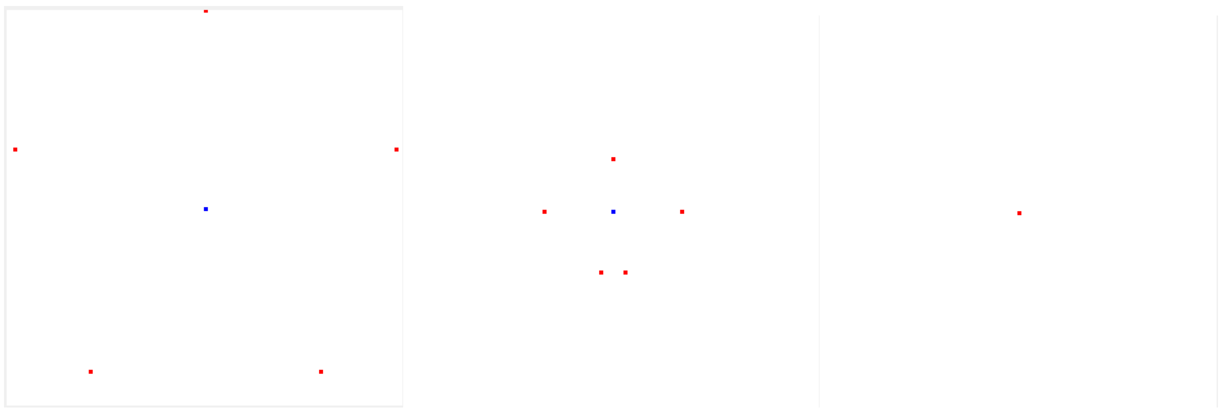
Report on task TASK 3, Interaction of pedestrians

In this task it is required to make pedestrians interact with each other and test whether they can move in arbitrary directions. Pedestrians should be absorbed (removed from the scenario) when arriving at target. The pedestrians are placed on vertices of a regular pentagon with the target on the orthocenter, which means that they are in a equally large distance around the single target and the distance between two neighbored pedestrians is the same for each couple. Besides, it is notable that the word "distance" stands for Euclidean distance instead of number of cells.

3.1 Setup of Scenario

The file configs/task_3.json generated by generate_configs.py saves all the information for this scenario.

Figure 5a demonstrates the initial scenario of Task 3. The grid size is set as 100 × 100, while the target is located at (50, 50), which is assigned as the center of a circle. 5 Pedestrians are placed equally on the circle, whose coordinates are (50, 0), (98, 35), (79, 91), (21, 91), (2, 35), computed by trigonometric functions.



(a) Initial state of Task 3 (b) Intermediate state of Task 3 (c) Final state of Task 3

Figure 5: Visualization of Task 3

3.2 Pedestrian Avoidance

Bresenham's algorithm is widely used to draw line primitives (straight lines) in bitmap images. Shown in Figure 6, it works with integer coordinates and uses only integer addition, subtraction, and bit shifting operations, making the computation efficient [1]. In the implementation of pedestrian or obstacle detection, each pedestrian is supposed to move from one to next possible position along Bresenham path. If the Bresenham path is blocked by other objects, this possible position will be abandoned.

The function avoidance_cost is used to implement pedestrian avoidance, which is adapted from the cost function in exercise sheet. For one pedestrian in an update step, the function takes other pedestrians into consideration, who are in the Bresenham path of that pedestrian to the next position. Afterwards it computes the avoidance costs (Equation 1) between that pedestrian and each filtered pedestrian and sums them up. The endure_factor k is introduced to adjust the maximum of the cost. Smaller k reflects weaker avoidance among the pedestrians, which indicates that they tend to endure the crowd to a greater extent. The value 0.01 is to avoid extreme cases where $r = r_max$ and the denominator thus becomes 0, which makes the cost go to infinity.

$$c(r) = \begin{cases} k \cdot \exp\left(\frac{1}{r^2 - r_{\max}^2 + 0.01}\right) & \text{if } r < r_{\max} \\ 0 & \text{else} \end{cases} \tag{1}$$

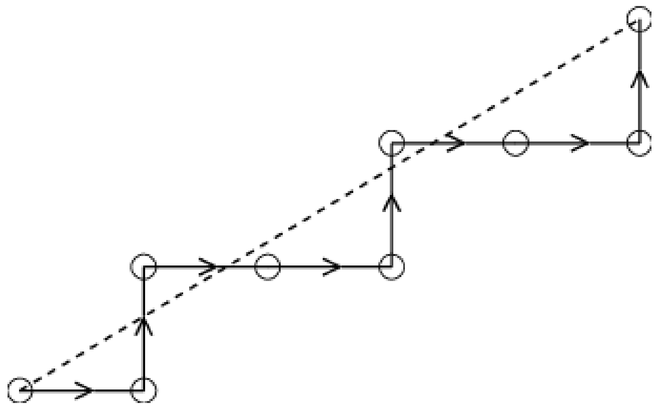


Figure 6: Bresenham path

3.3 Speed Adjustment

The speed adjustment is designed as follows. When the above-mentioned avoidance cost of a pedestrian is equal to 0, which means there is no other pedestrian in the forward direction, the pedestrian will increase the speed by 3%. In order to prevent them from moving too fast, the speed will not increase after it exceeds 1.2 times the initial speed. When the avoidance cost is not equal to 0, the pedestrian will maintain the initial speed. When the distance of the pedestrian to the targets is small enough that he/she can arrive at the targets at next time step, the speed will decrease to avoid overshooting by the function near_target_adjustment().

3.4 Simulation

The speed of each pedestrian is set to 5. When running the scenario, all the pedestrians move in a direction to the target at the center and reach the target one after another, approximately at the same time. The traveling time is 10s, 12s, 13s, 14s and 15s, respectively, which corresponds to the estimated speed $v = 50/(5s^{-1}) = 10s$.

4 Task 4

Report on task TASK 4, Obstacle avoidance

4.1 Setup of Scenarios

There are 2 scenarios in Task 4, "bottleneck" and "chicken test".

In "bottleneck" scenario, the narrow corridor ($5m \times 1m$) connects 2 large squared rooms ($10m \times 10m$). The two ends of the corridor are located at the middle points of the two walls. 150 people, located in the left half of the room on the left, are heading towards the exit in the middle of the wall on the right side of the right room.

In "chicken test", 150 people are supposed to bypass a U-shaped room with 3 walls. They are facing the entrance of the room, while the target is located to the right of the room. Inappropriate implementation of obstacle avoidance may cause the crowd to get stuck in the room.

4.2 Rudimentary Obstacle Avoidance

If one of the neighbors is a target, it will be the next possible position. In most instances, none of the neighbors is a target and the pedestrian will get the next possible position by its current speed and the direction to each neighbor. The direction is achieved by calculating cosine and sine of the angle between the pedestrian and each neighbor.

A set called "blocks" include the real-time coordinates of every obstacle and pedestrian except the specified one. When the coordinates of cells on Bresenham path from current position to next possible position are in the set "blocks", the neighbor corresponding to this possible position will be set to have an infinite avoidance cost, which makes the pedestrian avoid this direction later.

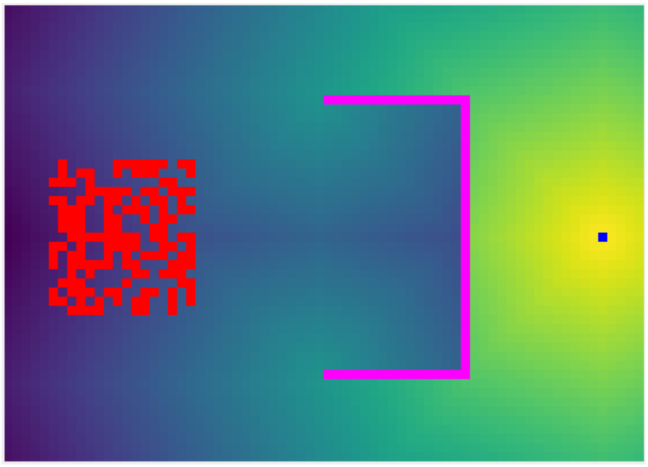


Figure 7: The visualization of Dijkstra distances in “chicken test”

To avoid stepping back, the computed distance of a pedestrian to the target is required to get smaller or at least stay unchanged.

After that, it will find the neighbor with the smallest avoidance cost and obtain the direction at next step.

Therefore, the rudimentary obstacle avoidance will be carried out simultaneously with the pedestrian avoidance. The difference is that no avoidance cost between pedestrians and obstacles is calculated and included in total costs.

4.3 Dijkstra Algorithm

In crowd modeling and simulation, Dijkstra algorithm is often used to find the shortest path from one point to another within a simulated environment. A function named _compute_dijkstra_distance_grid(self, targets: tuple[utils.Position]) ->npt.NDArray[np.float64] is defined to implement Dijkstra algorithm. This function initializes a distance grid filled with infinity values and set the values at targets to be 0. It adds each target to the “edge” list with a priority of 0, which includes their coordinates. The main loop continues until there are no more nodes in edge. In the loop, it extracts the node with the smallest distance (min_node_value) and its coordinates (min_node) from edge. The neighboring cells’ coordinates (new_x, new_y) in 8 directions are obtained and it will check if the new position is within the grid bounds and not an obstacle. The movement cost is to be calculated: 1.4142135623730951 for diagonal moves (since diagonals in a grid have a length of $\sqrt{2}$) and 1 for straight moves. If moving to (new_x, new_y) offers a shorter path than previously known (distances[new_x, new_y] >min_node_value + cost), update the distance and push the new position with its updated distance into “edge” list. Once the priority queue “edge” list is empty, indicating all reachable cells have been evaluated, the function returns the array of distance grid, which contains the shortest distances from any of the target positions to all other positions, considering obstacles. Figure 7 gives a visualization of Dijkstra distances, where the darker the color, the greater is the distance value.

4.4 Results of Simulation

Figure 8 and Figure 9 compare the simulation of “chicken test” with rudimentary obstacle avoidance to that with Dijkstra algorithm. In the simulation with rudimentary obstacle avoidance, the crowd fails to detect the whole form of the U-shaped obstacle and gets totally stuck in the end, without any pedestrian getting out of the trap. In contrast, with Dijkstra algorithm, the crowd is separated into two groups, going upwards and downwards. The pedestrians manage to bypass the obstacles and arrive at the target one by one. Therefore, Dijkstra algorithm is more effective in “chicken test”” scenario.

Figure 10 and Figure 11 compare the simulation of “bottleneck” with rudimentary obstacle avoidance to that with Dijkstra algorithm. In the simulation with rudimentary obstacle avoidance, the crowd gather heavily on both sides of the bottleneck entrance. However, they can still reach the target after a relatively long time. In contrast, with Dijkstra algorithm, the crowd seems to stand in a line before going through the bottleneck. There is no gathering on both sides, which makes the crowd move smoothly. In the end, all of the pedestrians

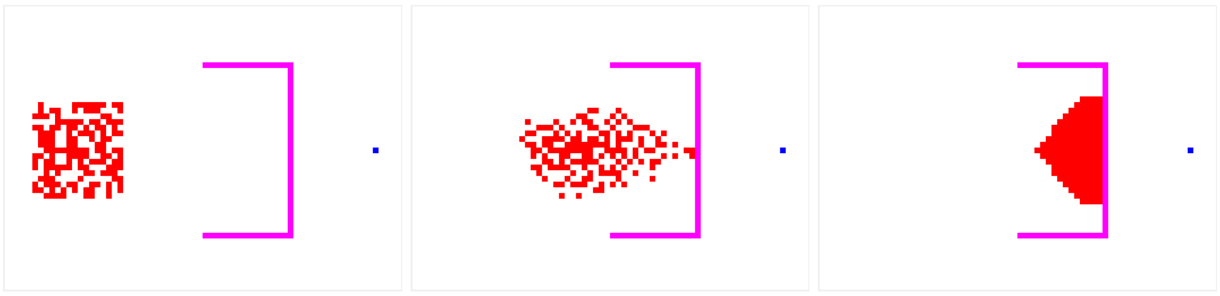

(a) Initial state (b) Intermediate state (c) State near the end

Figure 8: Visualization of "chicken test" in Task 4, with rudimentary obstacle avoidance

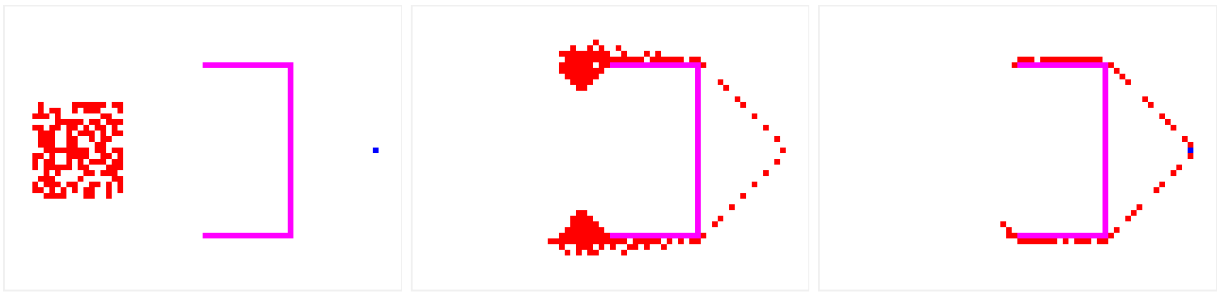

(a) Initial state (b) Intermediate state (c) State near the end

Figure 9: Visualization of "chicken test" in Task 4, with Dijkstra algorithm

get to the target more quickly. As a comparison, the evacuation time of the former scenario ranges from 51s to 168s, while that of the latter scenario ranges from 51s to 142s, which indicates that Dijkstra algorithm is more efficient in "bottleneck" scenario.

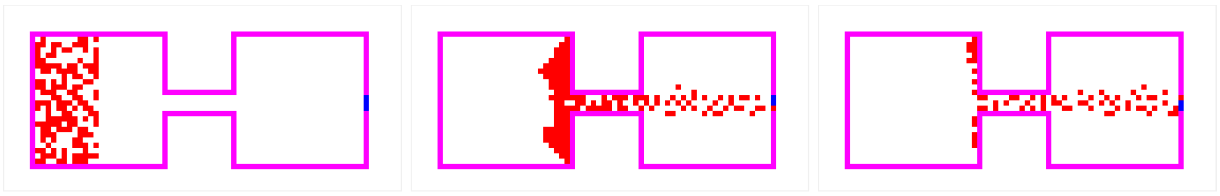

(a) Initial state (b) Intermediate state (c) State near the end

Figure 10: Visualization of "bottleneck" in Task 4, with rudimentary obstacle avoidance

---

5 Task 5

Report on task TASK 5, RiMEA tests

5.1 Test 1: RiMEA Scenario 1, Maintaining the specified walking speed in a corridor

5.1.1 Setup of Scenario

The scenario is a 40m long and 2 m wide corridor and a pedestrian placed on one end of the corridor is supposed to move in a straight line with correct speed towards the target on the other end. No obstacle is set along the path. Each cell is $0.4m \times 0.4m$, so the corridor excluding borders occupies the zone of 100×5 cells. The initial speed of the pedestrian is $1.33m/s = (1.33m/s)/(0.4m/cells) \approx 3.33\ cells/s$. The estimated travel time is $40m/1.33m/s \approx 30.08s$ and the RiMEA requires it to range from 26 to 34 seconds.

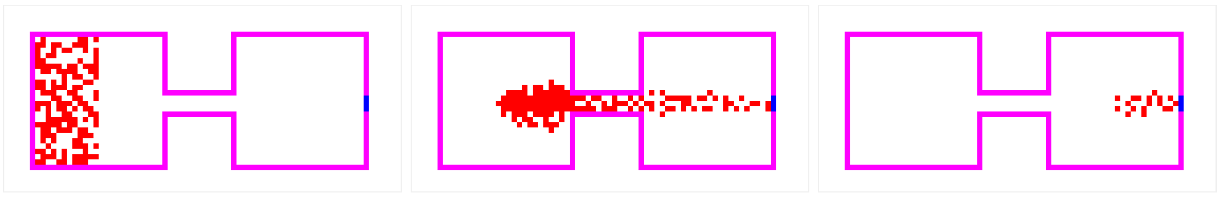

(a) Initial state (b) Intermediate state (c) State near the end

Figure 11: Visualization of "bottleneck" in Task 4, with Dijkstra algorithm

5.1.2 Results of Simulation

Figure 12 illustrates the simulation of Test 1. The pedestrian moves in a straight line and the travel time output in the terminal tells the travel time is $30s$, which is within the given range. After multiple tests, the results remain unchanged, verifying the stability of the algorithm.

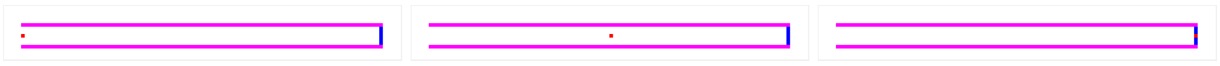

(a) Initial state (b) Intermediate state (c) Final state

Figure 12: Visualization of Test 1 in Task 5

5.2 Test 2: RiMEA Scenario 4

5.2.1 Setup of Scenario

Figure 13 depicts the setup of Test 2 [3], which is a $1000m$ (2500 cells) long and $10m$ (25 cells) wide corridor), where there are three measuring points ($2m \times 2m$ or 5×5 cells). However, this scenario requires huge computational resources. In order to complete the test, the scenario size should be reduced. The cell size is $0.4m \times 0.4m$. The cut-down corridor $16m$ (40 cells) long and $7.6m$ (19 cells) wide. The corridor is to be filled with different densities of persons with free walking speed ($1.2{-}1.4m/s$): $0.5P/m^2, 1P/m^2, 2P/m^2, 3P/m^2, 4P/m^2, 5P/m^2, 6P/m^2$. Each of the measuring point contains 3×3 cells. The coordinates of top-left cells of the measuring points are $(29, 10), (35, 10), (35, 13)$.

To get the number of pedestrians in the displayed zone, densities will be multiplied by the area of this zone ($16m \times 7.6m = 121.6m^2$). The average speed of individuals is to be calculated at designated measurement points across a 6-second interval, given a specified density. The initial 2 seconds of data may be disregarded as they represent a "transient response". Based on the collected data average speed in the range of a measuring point, the flow of the individuals at a measuring point can be calculated by $flow = speed \times n$, where n is the number of people per square meter in a measuring point. After the measuring time is over, the average of the flows over every time step during measuring time will be calculated and output in the terminal, which is recorded and plotted.

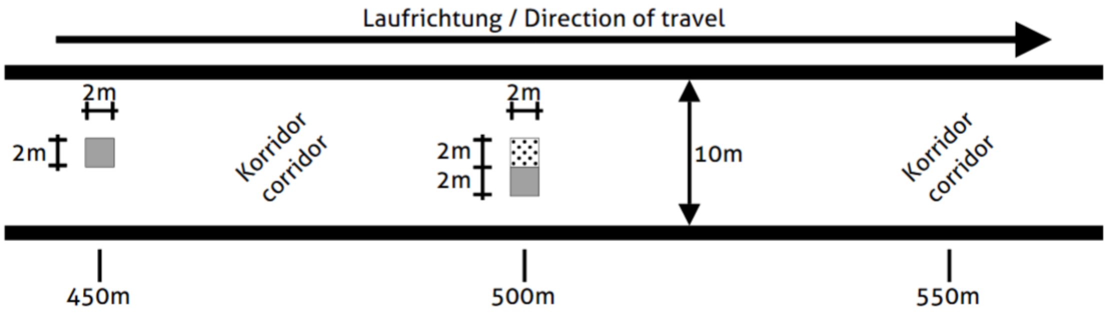


Figure 13: The setup of Test 2 in Task 5

5.2.2 Results of Simulation

As shown in Figure 14, 7 scenarios with $density = 0.5, 1, 2, ...6P/m^2$ are modeled. Figure 15 demonstrates the relationship between flow and density. At measuring point 1, 2 and 3, the flow reaches its maximum when density is equal to 2, 3 and 2, respectively. As density increases, flow increases to maximum and then decreases, which is reasonable, because the space of the scenario is limited and the flow must have an upper limit. When the preassigned density is small enough, the pedestrians can maintain initial speeds and the flow increases with density. When the preassigned density gets larger, the pedestrians slow down and the flow reduces with density.

The exercise sheet raises questions for this test: why can the premovement time be ignored for the tests? Why is the body size (e.g., the cell size in meters) needed to complete the tests?

During the premovement time the measurement is deactivated, making the crowd get ready to start walking. There are always pedestrians of the same distribution getting through the measuring points regardless of the premovement time so that it doesn't affect the measurement. In comparison, cell size contributes to the calculation of the flow. flow $= \frac{\text{average speed (cells/s)} \times \text{cell size (m/cell)} \times n\text{(Persons)}}{\text{width (cells)} \times \text{cell size (m/cell)} \times \text{height (cells)} \times \text{cell size (m/cell)}} = \frac{\text{average speed (cells/s)} \times n\text{(Persons)}}{\text{width (cells)} \times \text{cell size (m/cell)} \times \text{height (cells)}}$, including the variable cell size. Therefore, measured flow is inversely proportional to cell size, which can't be neglected.

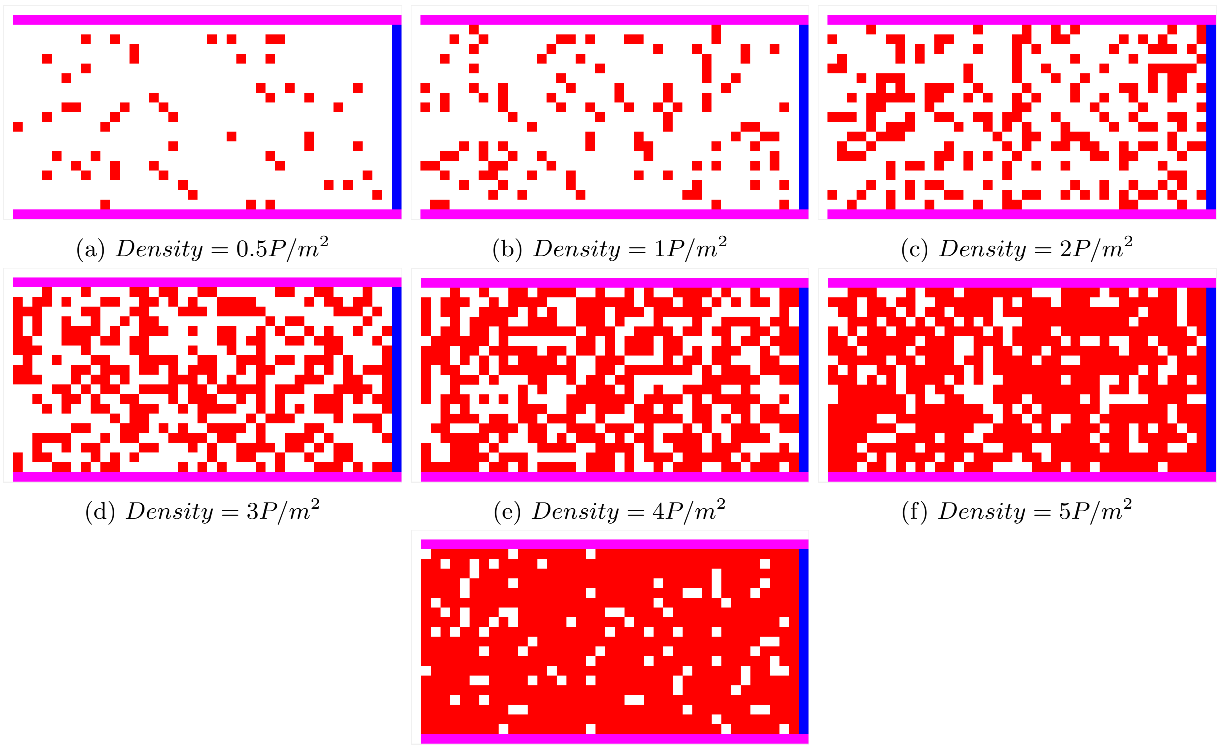

(a) $Density = 0.5P/m^2$
(b) $Density = 1P/m^2$
(c) $Density = 2P/m^2$
(d) $Density = 3P/m^2$
(e) $Density = 4P/m^2$
(f) $Density = 5P/m^2$
(g) $Density = 6P/m^2$

Figure 14: Visualization of Test 2 in Task 5

5.3 Test 3: RiMEA Scenario 6

5.3.1 Setup of Scenario

A group of twenty uniformly distributed pedestrians, heading toward a left-turning corner (as shown in Figure 16) [3], will navigate around it successfully without intersecting any walls. The cell size is still $0.4m \times 0.4m$. The scenario is made up of 40×40 cells and the pedestrians are uniformly placed at bottom left.

5.3.2 Results of Simulation

According to Figure 17, the crowd turns left successfully without getting through the walls and arrives at the targets as planned, which indicates that the algorithm passes Test 3.

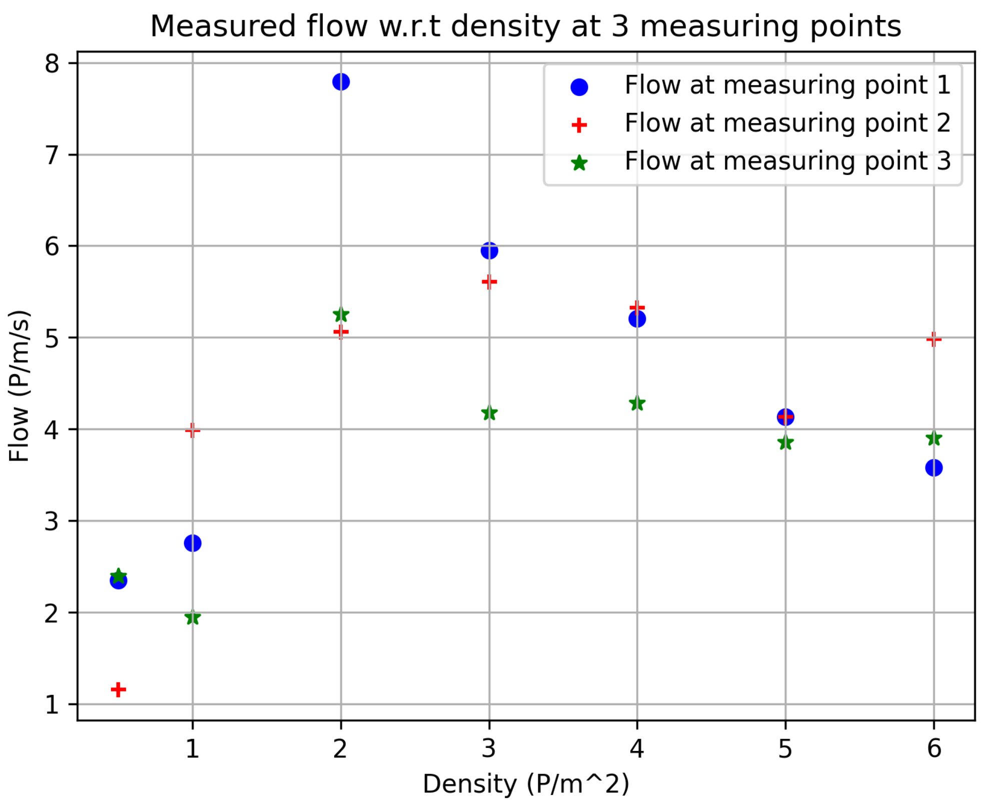


Figure 15: Measured flows w.r.t density at 3 measuring points

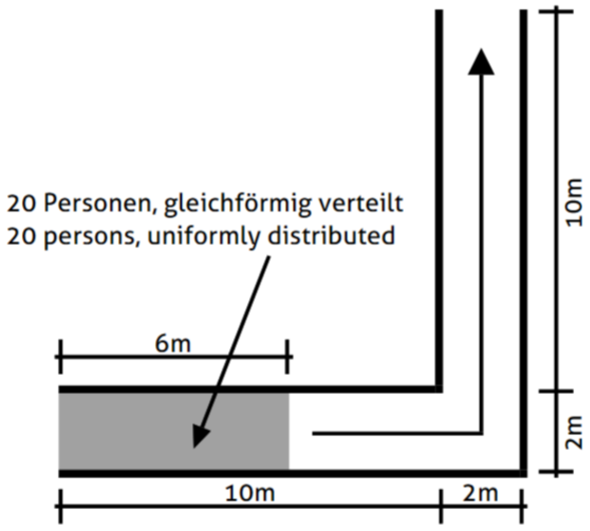
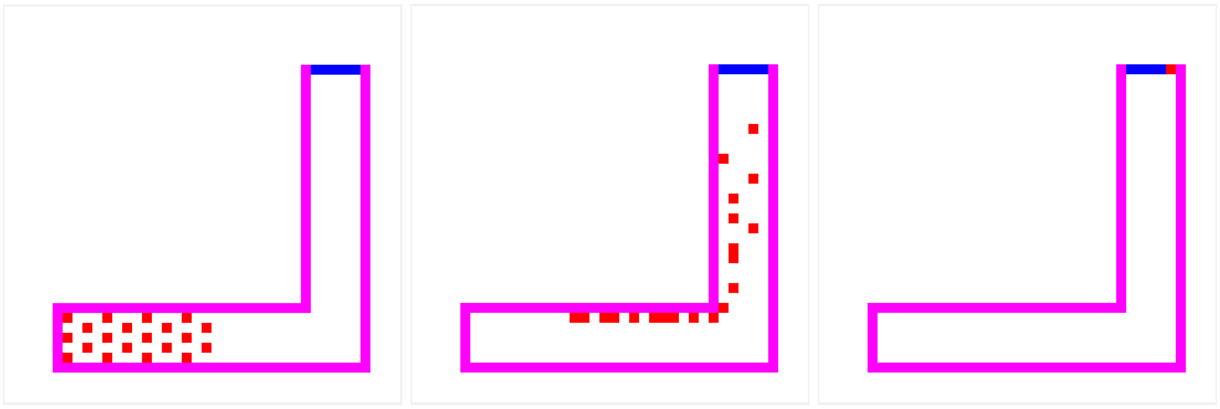

20 Personen, gleichförmig verteilt
20 persons, uniformly distributed
6m
10m
2m
10m
2m

Figure 16: The setup of Test 3 in Task 5



(a) Initial state (b) Intermediate state (c) Final state

Figure 17: Visualization of Test 3 in Task 5

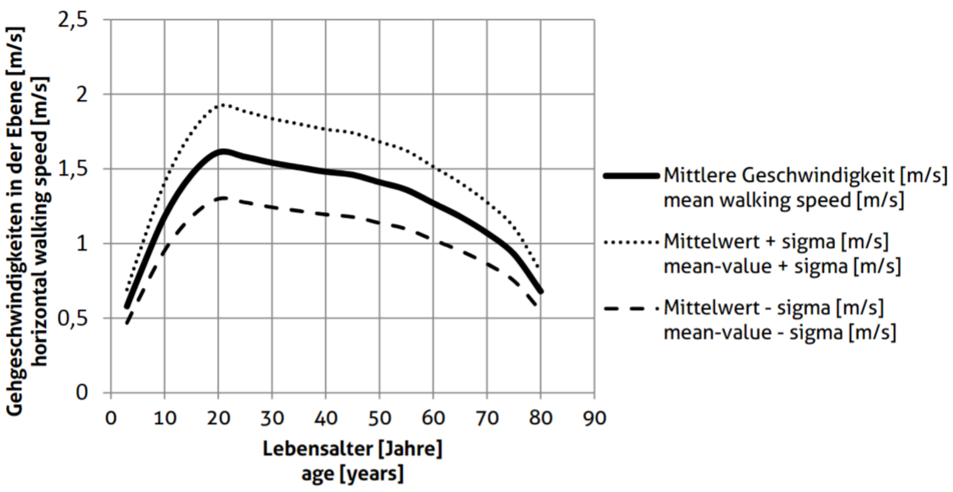


Figure 18: Walking speed in the plane as a function of age based on Weidmann

5.4 Test 4: RiMEA Scenario 7

5.4.1 Setup of Scenario

In Test 4, 50 pedestrians are facing 50 targets $20m$ away from them. The cell size is $1m \times 1m$ for simplification. The pedestrians are randomly sampled from configs/rimea_7_speeds.csv, where each pedestrian is preassigned a speed correlated to the age, as shown in Figure 18 [4]. The pedestrians are supposed to walk straight towards targets so the speed is computed via walking distance ($50m$) divided by time spent by each pedestrian. It is required to demonstrate that the distribution of walking speeds observed in the simulation aligns with the distribution presented in Figure 18.

5.4.2 Results of Simulation

Figure 19 compares the distribution of measured speeds in Test 4 and preassigned speeds from rimea_7_speeds.csv. It shows that most of the measured speeds are below the preassigned speeds. However, the preassigned speeds will be approximately aligned with the measured ones if given an offset of $-0.33m/s$. The potential reason is that the pedestrians are placed in a line in the beginning, which makes them have an effect on each other at the start. They tend to avoid colliding with each other, which causes a delay of a few seconds. After the crowd scatters, the influence of avoidance becomes very weak and the pedestrians can move with preassigned speeds respectively. Therefore, an offset exists together with outliers, while the distribution in practice is roughly consistent with the ideal distribution.

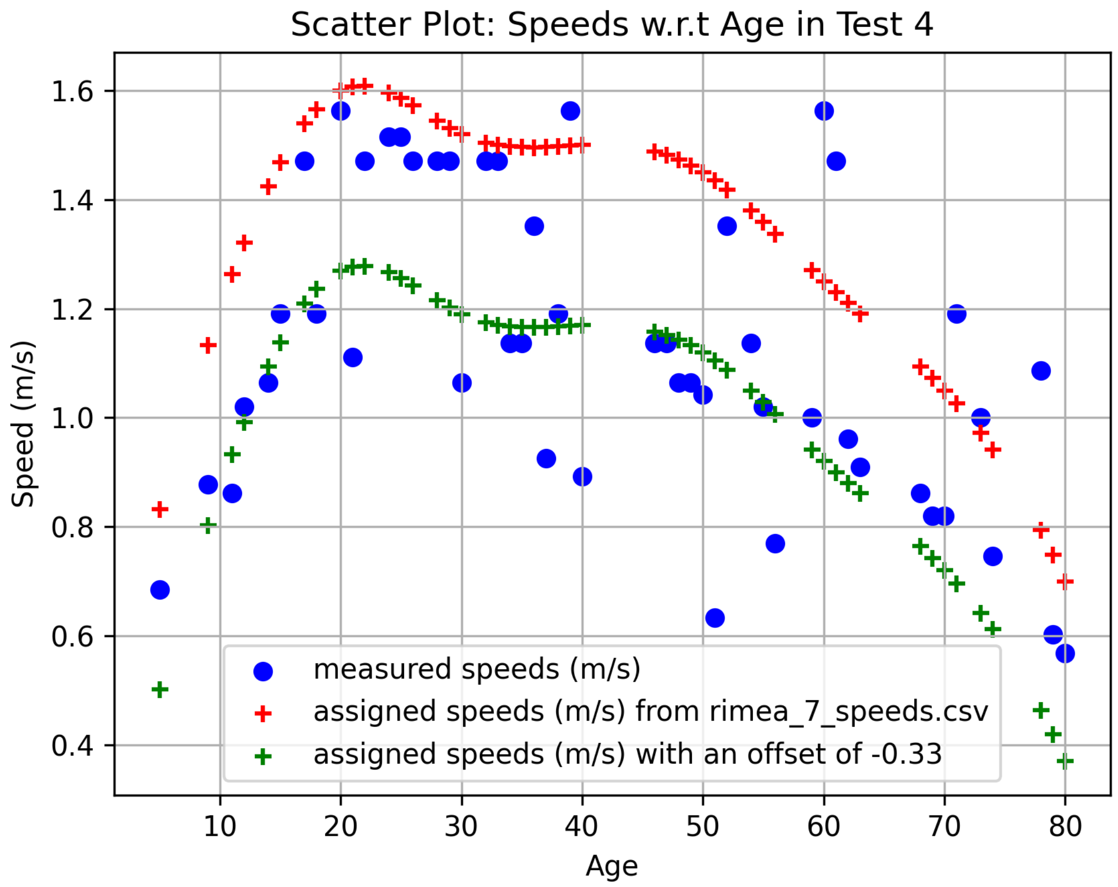


Figure 19: Distribution of measured speeds in Test 4 and preassigned speeds from rimea_7_speeds.csv w.r.t age

References

[1] Bram Bolder, Thorsten Struckmann, Gunnar Bali, Norbert Eicker, Thomas Lippert, Boris Orth, Kathy Schilling, and Peer Ueberholz. High precision study of the q q ¯ potential from wilson loops at large distances. *Physical Review D*, 63, 03 2001.

[2] Martin Gardner. Mathematical games. *Scientific american*, 222(6):132–140, 1970.

[3] RIMEA. *Guideline for Microscopic Evacuation Analysis.*

[4] U Weidmann. Transporttechnik der fussgänger-schriftenreihe ivt-bericte 90. *Institut für Verkehrsplanung, Transporttechnik, Strassen-und Eisenbahnbau*, 1993.