Report for exercise 1 from group f

Tasks addressed:	5
Authors:	Shihong Zhang (03764740) Mengshuo Li (03792428) Yuxuan Wang (03767260) Augustin Gaspard Camille Curinier (03784531) Qingyu Wang (03792094)
Last compiled:	2024–05–06

The work on tasks was divided in the following way:

Shihong Zhang (03764740)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Mengshuo Li (03792428)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Yuxuan Wang (03767260)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Augustin Gaspard Camille Curinier (03784531)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Qingyu Wang (03792094)	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

1 Gerneral Infomation

The simulation process primarily involves the initialization of various entities within the `simulation` file. The `gui` file, driven by configurations from the `config` file, creates the graphical user interface (GUI) and updates it in real-time based on the states of these entities. Configuration files for different tasks are generated through `generate_configs.py`.

The control logic of the simulation dictates that all pedestrians move towards their targets. In this model, one grid cell represents 0.4 meters, and each update corresponds to one second of real time. The updating rule for each pedestrian is as follows: identify all areas that the pedestrian can reach, calculate the utility value for these areas—derived from the distance to the nearest target and the nearest other pedestrian. The cell with the highest utility value within the reachable areas is then selected as the new position for the pedestrian. The detailed implementation of this mechanism is further elaborated in subsequent tasks. By continuously updating the positions of the pedestrians, the simulation aims to realistically guide them to their targets.

This simulation holds significant practical value, as illustrated in Task 5, which sets up various real-life scenarios and derives meaningful conclusions, such as the impact of crowd density on actual pedestrian speeds.

This overview serves as the introduction to our exercise, highlighting the functionality and purpose of the simulation within the broader context of crowd dynamics analysis.

2 Report for tasks

Report on task 1, Setting up the modeling environment

In Task 1, we were tasked with enhancing the initial setup of our simulation environment, as specified by the template provided. This required significant improvements to the `Simulation.init()` method, ensuring a comprehensive and dynamic initialization of the cellular automaton model.

The main goal for this task was the effective integration of pedestrians, targets, and obstacles into the simulation grid. This setup was achieved through detailed configuration settings specified in `toy_example.json`, which guided the placement of these elements within the grid.

Furthermore, we developed the `Simulation.get_grid()` method, a critical addition that allows for the retrieval of the grid's current state. This functionality is essential for the real-time visualization of the simulation, enabling us to monitor the placement and interaction of elements throughout the simulation process.

Improvements to the `Simulation.init()` function ensured that all necessary components were properly configured and instantiated at the start of the simulation. By utilizing the `toy_example.json` configuration for our tests, we confirmed that our simulation could accurately display dynamic elements such as pedestrian movement.

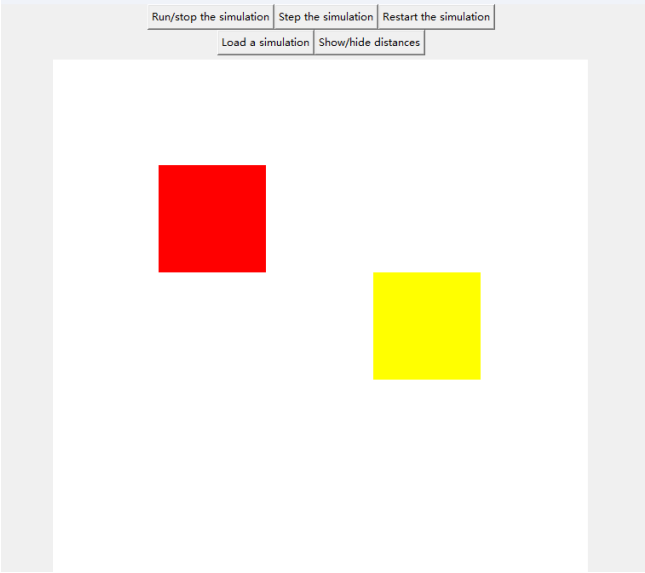

Figure 1: task 1

Report on task 2, First step of a single pedestrian

In this task, our objective was to enable a pedestrian to navigate towards a designated target within a 50 by 50 cell grid. This was achieved through significant modifications to the `update()` function, which now guides the pedestrian through the grid based on dynamic simulations.

The `_get_neighbors()` function plays a pivotal role in determining movement possibilities. It accepts a `position` parameter, consisting of `x` and `y` coordinates that represent specific points on the grid, thoroughly defined in `utils.py`. The function returns a list of positions corresponding to all adjacent cells, ensuring none of these exceed the grid's limits.

The enhanced `update()` function starts by identifying all potential points that the pedestrian can feasibly reach, achieved through the `_get_neighbors()` function. The process involves filtering out any positions that are currently occupied by other pedestrians or blocked by obstacles. Crucially, the pedestrian's current position is included among the potential points to allow for the possibility of remaining stationary, preventing illogical moves to inaccessible areas.

These potential positions are then evaluated by calculating the distance from each to the target. The optimal position for the next step is the one with the minimal distance to the target, ensuring that the pedestrian moves in the most direct route possible, provided there are no intervening obstacles.

We manually created a configuration file that established the scenario outlined in the task requirements. Visualization was pivotal in demonstrating the effectiveness of the implemented logic, clearly showing the pedestrian at initialization and upon successfully reaching the target, as depicted in Figure 2 of the report. This visual documentation verifies the functionality of our simulation by showcasing the starting and end points of the pedestrian's journey.

Through this methodical approach, the simulation not only demonstrates the pedestrian's ability to navigate towards a target but also verifies that the movement logic adheres closely to the planned pathway, highlighted by the pedestrian's progression in the simulation environment.

Figure 2: task 2

Report on task 3, Interaction of pedestrians

In Task 3, significant advancements were made to the simulation's `update()` function. These enhancements are designed to incorporate complex pedestrian avoidance capabilities and to introduce a nuanced concept of speed into the pedestrian's movement dynamics. This task shifted the pedestrian update mechanism from a simple distance-based approach to a more sophisticated utility-based system.

Core Implementations:

- Utility and Loss Function: The model now prioritizes utility values that integrate a loss component, significantly improving pedestrian avoidance capabilities. This loss function is outlined in the attached figure. It functions as a regularization factor, which is sensitive to the distance between pedestrians. Adjustments to pedestrian behavior are made possible through the rmax hyperparameter, which fine-tunes the sensitivity of the avoidance mechanism.

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

- Introduction of Speed Attribute: A `speed` attribute was added to each pedestrian, denoting their movement capability per update cycle. Prior to this, the model updated each pedestrian's position in single, discrete steps. Now, movement is influenced by the `speed` attribute, with a `moving_credit` system introduced to manage movement iterations. Pedestrians with a `moving_credit` reaching one are updated, with the credit reducing by either one or $\sqrt{2}$ for diagonal moves, reflecting the actual Euclidean distance traveled.
- Optimized Movement Logic: The logic now includes an `_is_absorbing` parameter to check if pedestrians should be removed from the simulation upon reaching their target. This ensures efficient management of pedestrian flow, especially in dense scenarios. If a pedestrian reaches a cell that coincides with a target and `_is_absorbing` is true, they are removed from the simulation. This update strategy prevents multiple pedestrians from occupying the same cell by continuously updating the pedestrian states throughout the simulation cycle.
- Configuration and Execution: The described scenario was meticulously set up using a custom `generate_configs.py` script, which produced the configuration files necessary to simulate the environment as depicted in Figure 3. This setup effectively demonstrated the simulation's capability to handle complex pedestrian interactions and movement strategies.

Task 3 has thus successfully demonstrated a significant leap in simulating realistic pedestrian behaviors, providing insights into effective interaction strategies and movement mechanics that are crucial for understanding and analyzing pedestrian dynamics in complex environments.

Figure 3: task 3

Report on task 4, Obstacle avoidance

In Task 4, our focus was on refining the simulation's obstacle avoidance capabilities by integrating Dijkstra's algorithm. This enhancement was aimed at more accurately reflecting real-world pedestrian behavior in scenarios involving obstacles.

Dijkstra's Algorithm Implementation:

We adopted Dijkstra's algorithm to improve how the simulation processes obstacles. The core principle of this algorithm is to compute the shortest paths from a target while treating obstacles as impassable. This is achieved by starting with the distance at the target set to zero and expanding outward, updating the distances only if a shorter, unobstructed path is discovered. This method effectively prevents the algorithm from routing through obstacles, ensuring that each cell's distance value reflects the shortest possible route to the target.

Application and Observations:

We applied this new approach to two specific scenarios: the "bottleneck" and the "chicken test," generated via `generate_configs.py`. The outcomes were observed under both naive and Dijkstra's algorithm-enhanced distance computations:

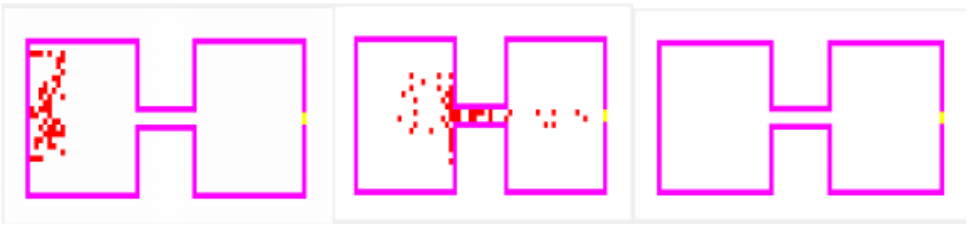

Figure 4: bottleneck with naive algorithm

- Naive Method Results: In chicken test scenario, it failed. In bottleneck scenario, initially, pedestrians approached obstacles directly and rerouted only upon close contact, leading to inefficient pathfinding.

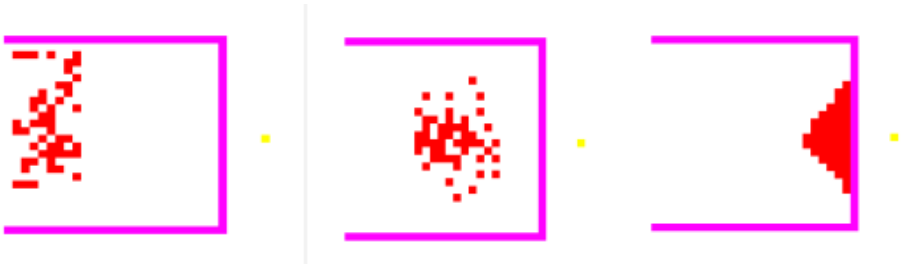

Figure 5: chicken test with naive algorithm

- Dijkstra's Algorithm Results: The algorithm demonstrated a substantial improvement in path efficiency. In the bottleneck scenario, pedestrians approached and accumulated strategically at narrower passages without unnecessary detours. In the chicken test scenario, they found the most direct routes to the target, avoiding obstacles effectively.

The contrast was visually documented in our simulation results. The naive approach showed pedestrians struggling with direct obstacles (Figures 4 and 5), while Dijkstra's algorithm allowed for a more intelligent navigation, clearly visible in the more strategic pedestrian movement patterns (Figures 6 and 7).

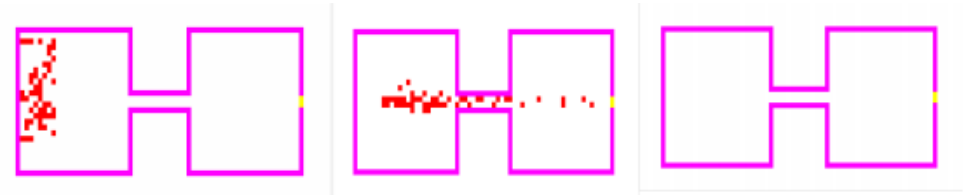

Figure 6: bottleneck with dijkstra algorithm

This task confirmed that Dijkstra's algorithm significantly enhances the simulation's ability to handle complex pedestrian dynamics involving obstacles, aligning the virtual behavior closer to realistic pedestrian movements.

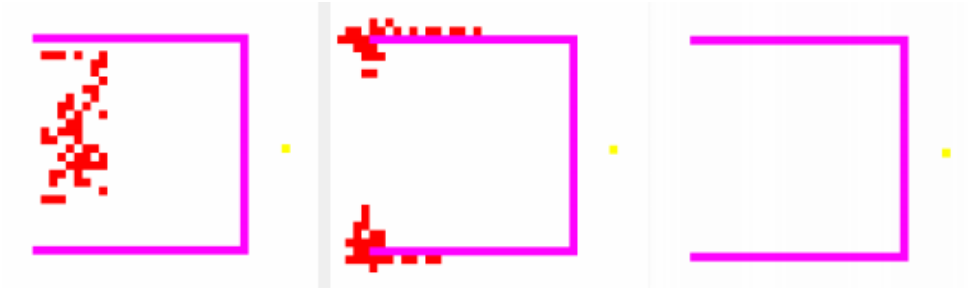

Figure 7: chicken test with dijkstra algorithm

Report on task 5, Tests

In Task 5, we rigorously tested our simulation model against several scenarios outlined in the RiMEA guidelines to validate and verify the software's capability in modeling crowd dynamics. These tests examined various aspects of pedestrian behavior, including movement speed, navigation around obstacles, and demographic variations. Implementation and Results:

Test 1: Straight Line Movement with Correct Speed

For the first test, we simulated a pedestrian moving in a straight line at a prescribed speed, reflecting typical human walking speeds. To accurately model this, we set each simulation cell to represent a body dimension of 0.4 meters, closely approximating the average width of a human body. This scale is crucial for realistic simulation of passability and space occupation.

The premovement time, often included in evacuation simulations, was deemed non-essential for this particular test and was thus omitted from real-time calculations. This decision was based on its minimal impact on the overall simulation outcomes, as it does not affect the pedestrian's movement per step but rather the initiation of movement.

In refining the movement logic of our simulation, an important adjustment was made to address the cost efficiency of pedestrian path choices. Initially, the update logic did not differentiate between moving to a diagonal neighbor and an adjacent neighbor. However, in aiming to guide pedestrians along the shortest path to the target, it became evident that moving diagonally, although potentially quicker in reaching the target, consumes more movement credit compared to moving to an adjacent cell.

To optimize the efficiency of pedestrian movement and minimize unnecessary credit expenditure, we implemented a decision-making mechanism within the simulation's update function. Now, when multiple potential movements have equivalent utility values suggesting similar benefits in terms of progressing towards the target the simulation prioritizes the movement option that requires less credit. This ensures that pedestrians move along the most cost-effective path, conserving movement credit for more critical maneuvers.

This subtle yet significant enhancement in the movement logic ensures that the simulation not only directs pedestrians efficiently towards their targets but also does so in a way that mimics realistic decision-making processes where energy conservation is a factor.

Figure 8: RiMEA scenario 1

The simulation was configured to ensure the pedestrian moved with a speed between 4.5 and 5.1 km/h, translating to a travel time between 26 to 34 seconds to cover a set distance, aligning with the RiMEA standards for pedestrian speed. The outcomes, as shown in Figure 8, were within the expected range, affirming the model's accuracy in replicating realistic pedestrian speeds. Test 2 : Fundamental Diagram Plotting Using MeasuringPoint Class

In this part, we focused on implementing the MeasuringPoint class to track and analyze pedestrian flow through specific points in the simulation. We deployed measuring points that dynamically calculated flow rates during each update cycle. To obtain a reliable measure of pedestrian flow, we averaged the updates over the entire measurement period.

Figure 9: RiMEA scenario 4, 20er

Flow Calculation Methodology: Before each update, we documented the number of pedestrians within the designated measuring area to calculate population density by dividing the total number of pedestrians by the area. After the update, we identified the positions of these pedestrians and calculated their actual speed based on the displacement during the update. The average speed obtained from these calculations was then multiplied by the population density to determine the flow rate for that particular update within the measuring area.

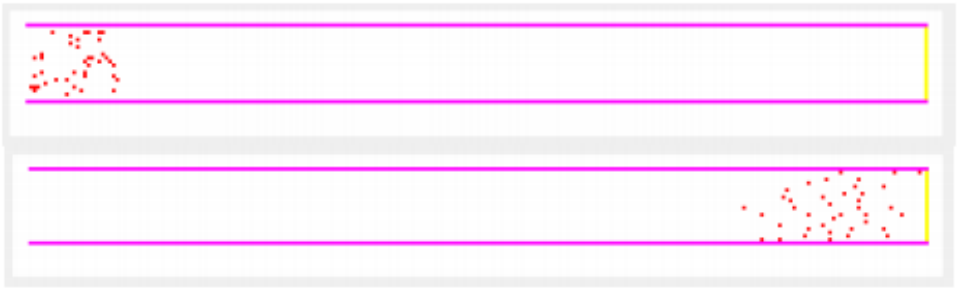

Figure 10: RiMEA scenario 4, 40er

Due to computational limitations, we optimized the scenario configuration by reducing the grid size to 260x35, with the corridor dimensions set at 250x20. We also categorized pedestrian density into five levels: 20/500, 40/500, 80/500, 160/500, and 240/500, corresponding to Figures 9-13. The simulation results clearly demonstrated that pedestrian speed is affected by population density. As the density increased, the simulations showed a broader spread of pedestrian speeds, especially affecting the speed of those at the rear of the crowd, confirming that higher densities impact pedestrian flow dynamics significantly.

Figure 11: RiMEA scenario 4, 80er

Simulation Setup and Results: We created specific configuration files to initialize the simulation environment appropriately for this test. The adjustments made to the grid size were necessary to ensure the computational feasibility of the simulations on available hardware. The results from varying density levels illustrated a clear trend: as population density increases, the overall flow rate within the same simulation duration decreases, highlighting the impact of crowding on pedestrian movement.

Figure 12: RiMEA scenario 4, 160er

This test not only validated the MeasuringPoint class's functionality but also provided essential insights into how pedestrian density influences walking speeds and overall flow dynamics, crucial for planning and managing pedestrian traffic in real-world scenarios.

Figure 13: RiMEA scenario 4, 240er

Test 3: Navigating Around a Corner (RiMEA Scenario 6)

In this part of Task 5, we explored the simulation's ability to accurately model pedestrian movement around corners. This test was particularly focused on ensuring that pedestrians could navigate a corner turning to the left without colliding with or passing through walls.

Implementation and Simulation Setup: Using a configuration file generated by the `generate_configs` script, we set up the scenario to feature twenty individuals moving towards a corridor that turns left, as illustrated in Figure 5. This setup was designed to test the algorithm's efficiency in handling complex pedestrian pathways and ensuring realistic movement dynamics around obstacles.

Results and Observations: The simulation results, depicted in Figure 14, demonstrate that all pedestrians successfully navigated the corner as planned. Each individual followed the path of the corridor and reached the designated target without any deviations through walls or other barriers. This outcome confirms the robustness of our pathfinding algorithms in managing real-world walking scenarios where pedestrians must adapt their routes based on environmental constraints.

Conclusion: This test proved the effectiveness of our simulation model in accurately depicting pedestrian movement in scenarios involving complex directional changes. The ability of the model to guide pedestrians around corners without error is crucial for applications in urban planning and crowd management, where accurate prediction and control of pedestrian flows are necessary for safety and efficiency.

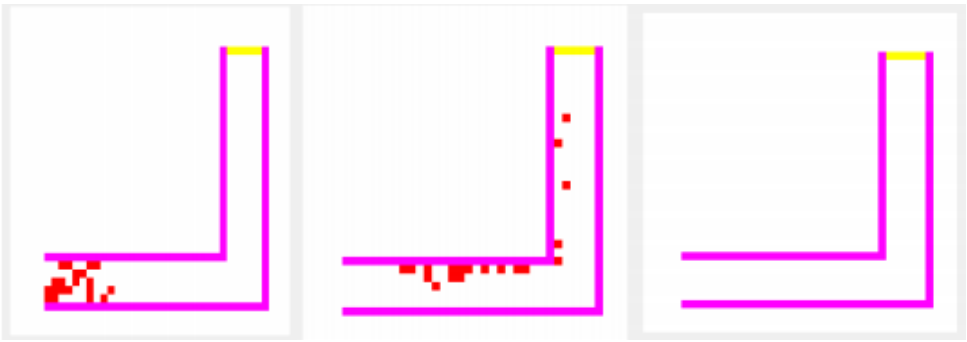

Figure 14: RiMEA scenario 6

Test4: Simulating Demographic Parameters

In Test 4 of Task 5, our focus was on simulating pedestrian speeds based on demographic parameters, specifically age, in line with the RiMEA guidelines. We utilized the data from `configs/rimea_7_speeds.csv` which provided approximations of the mean walking speeds for various age groups.

Implementation and Setup: The simulation environment was configured using the `generate_configs` script to reflect the conditions specified for the test. We implemented a function that adjusted pedestrian speeds according to the age data from the CSV file, ensuring that each pedestrian's parameters were accurately modeled according to their demographic profile.

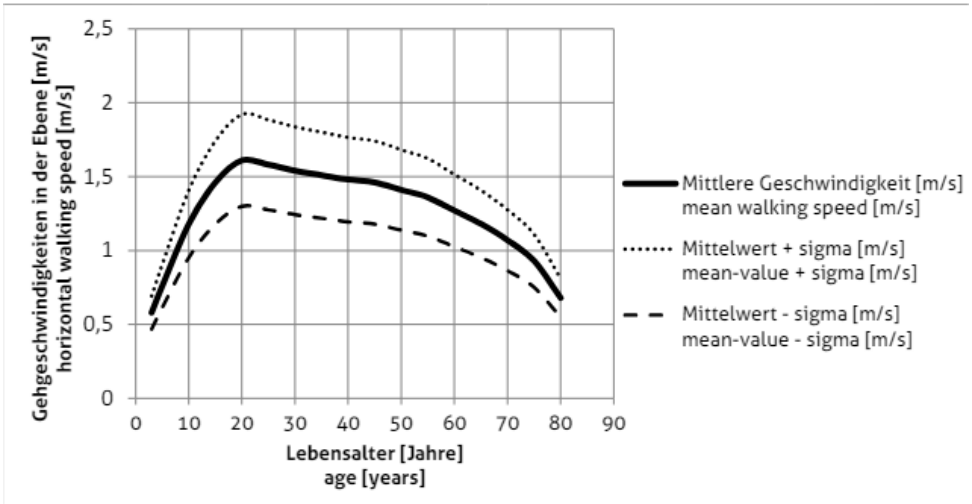

Figure 15: RiMEA scenario 7, Weidmann graph

Calculating and Validating Pedestrian Speeds: To confirm the simulation's accuracy, we developed a function to calculate the actual walking speeds of each pedestrian by monitoring their movements and the time taken. This data was derived from tracking the number of steps each pedestrian took and their start and end positions, allowing us to determine the distances traveled and compute their speeds accordingly.

We also visualized the relationship between walking speeds and ages through a scatter plot to directly compare the simulation results with the expected distributions in the RiMEA guidelines.

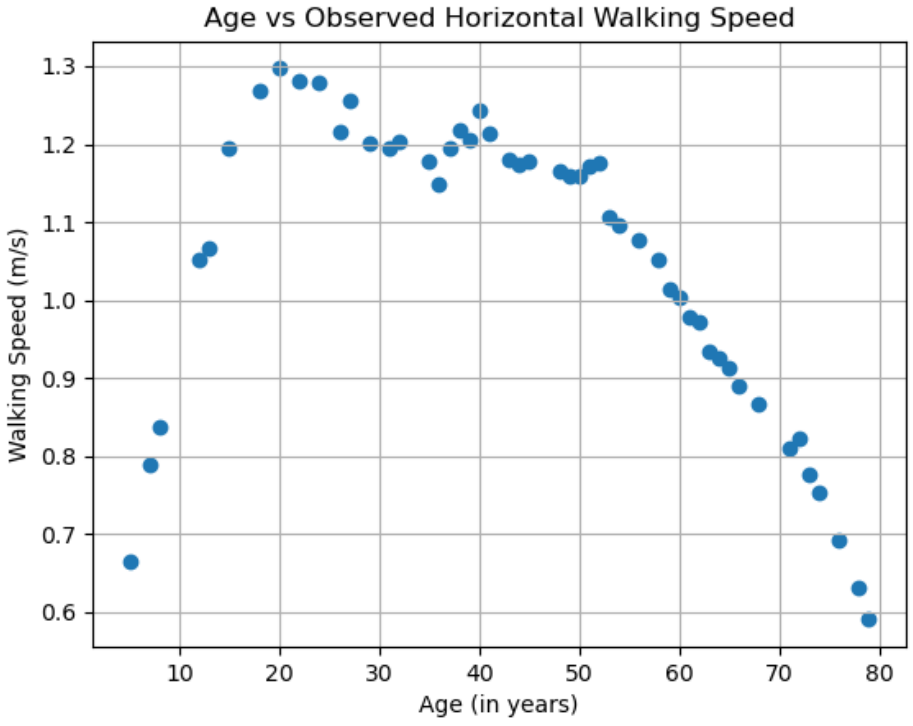

Figure 16: results form simulator

Results and Observations: The scatter plot, shown in Figure 17, demonstrated a trend generally consistent with the expected outcomes, although the overall speeds were slightly lower than anticipated. This deviation might be attributed to the impact of crowd density on walking speeds, a factor highlighted in Test 2. Despite

this, the shape of the distribution closely followed the Weidmann graph, indicating a successful replication of pedestrian behavior as per demographic variations.

The analysis confirmed that while crowd density might influence individual speeds, our model effectively captured the essential dynamics and variations in pedestrian speeds related to age, fulfilling the requirements of Test 4.

Conclusion: This test not only validated our simulation's ability to adapt pedestrian speeds based on age but also demonstrated its capacity to reflect realistic variations, as detailed in demographic data. The results underscored the model's utility in analyzing pedestrian dynamics within a varied population, providing valuable insights for urban planning and crowd management strategies.