Report for exercise 1 from group A

Tasks addressed:	4
Authors:	Fehmi Şener (03778853)
	Didem Öngü (03763740)
	Tigran Bunarjyan (03755503)
Last compiled:	2023–11–02
Source code:	https://github.com/fehmisener/MLCMS-praktikum/tree/main/exercise-1

The work on tasks was divided in the following way:

Fehmi Şener (03778853) **Project lead**	Task 1	34%
	Task 2	34%
	Task 3	34%
	Task 4	34%
Didem Öngü (03763740)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%
Tigran Bunarjyan (03755503)	Task 1	33%
	Task 2	33%
	Task 3	33%
	Task 4	33%

Report on task 1, Setting up the modeling environment

In Task 1, the focus was on establishing the initial framework for modeling and simulating a human crowd using a cellular automaton. You can follow the instructions in the *README.md* file for the installation of the project in the local environment. This is our Technology Stack:

- Python: The core programming language driving the simulation and modeling environment. Python's versatility and rich libraries make it an ideal choice for this project.
- Python Tkinter GUI Library: The graphical user interface (GUI) was built using the Tkinter library, which is a standard choice for creating user-friendly interfaces in Python. It offers buttons and dialogs for user interaction and control.
- GitHub: GitHub was employed as a version control and collaborative development platform integrated with Visual Studio Code (VSC). It facilitated collaborative coding, version tracking, and project management.
- Overleaf: For documentation collaboration, Overleaf was chosen as the platform for creating and maintaining project documentation. Overleaf offers a collaborative, cloud-based environment for LaTeX document preparation.

Initially, a graphical representation of the project was produced. The design has a grid where the dimensions and side lengths of each cell can be customized. The *tkinter* package was used to practically implement this in Python, handling the graphical elements of the project. We also tried the project on two different operating systems, both macOS and Windows. The requirements include the following key components:

1. Basic Visualization: We leveraged a GUI created with Python's Tkinter library, providing users with an interactive interface for crowd simulation. This GUI offers the following functions:

- **Create Simulation:** Users can manually add targets, pedestrians, and obstacles at specific locations, with the added ability to set pedestrian speed.
- **Restart Simulation:** This button allows for the restart of the most recently created or loaded simulation, ensuring flexibility in managing scenarios.
- **Step Simulation:** The stepwise simulation progression enables users to advance crowd behavior one step at a time, facilitating observation and analysis.
- **Load Simulation:** A file dialog integrated into the GUI enables users to load simulation data from JSON files. This method of importing data ensures ease of use and adaptability for different scenarios.

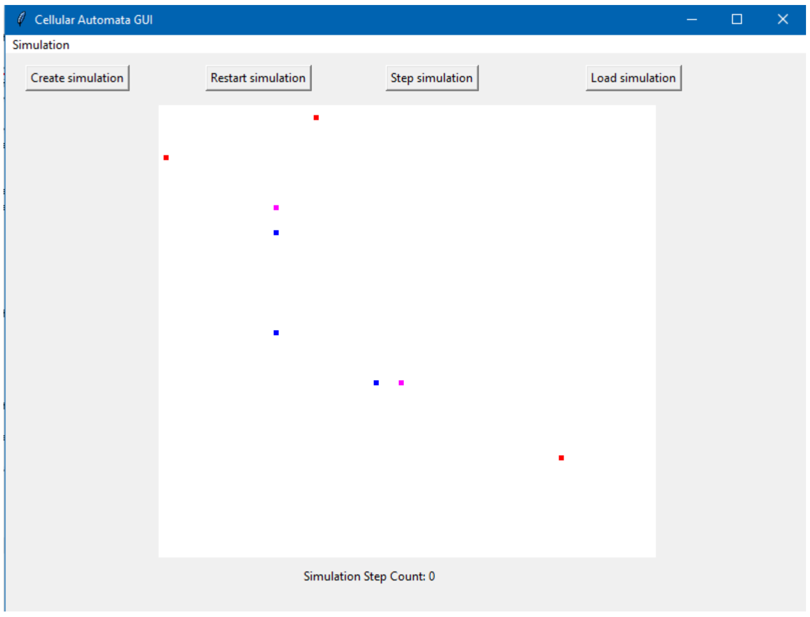


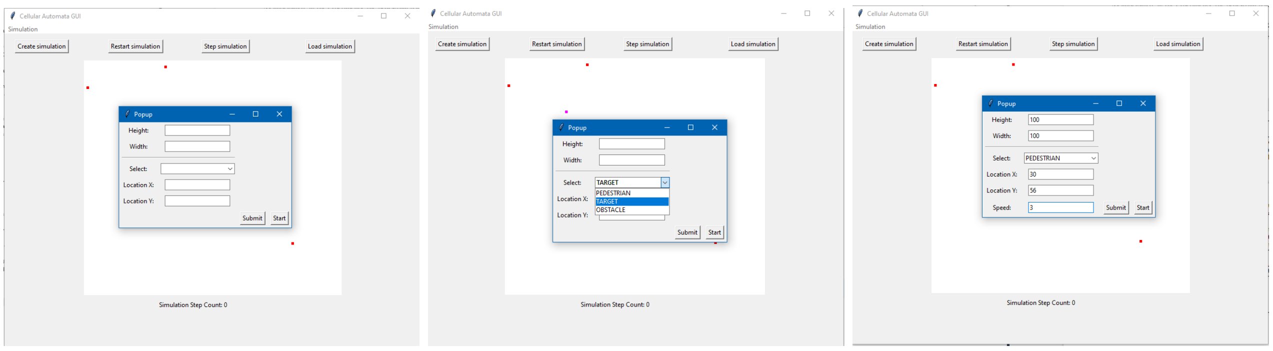
Figure 1: Simple GUI Start Page

2. Adding Pedestrians in Cells: Pedestrian placement, a fundamental aspect of crowd simulation, was successfully implemented. Pedestrians can be introduced to the simulation through two methods:

- **Loading Simulation Data:** Pedestrians can be included in the simulation by importing data from JSON files. This offers a convenient and structured way to define pedestrian locations and their associated speeds.
- **Manual Creation:** Users have the option to manually add pedestrians to specific cells by specifying their X and Y coordinates. Additionally, the ability to set pedestrian speed provides further control over their behavior.

3. Adding Targets in Cells: Similar to adding pedestrians, adding targets to the simulation can be done manually or by importing them from a JSON file, offering flexibility in modeling different scenarios.

4. Adding Obstacles: Obstacles play a crucial role in crowd simulation by creating inaccessible cells for pedestrians. The implementation ensures that pedestrians intelligently adapt their paths when obstacles are present, bypassing these blocked cells to reach their destinations.

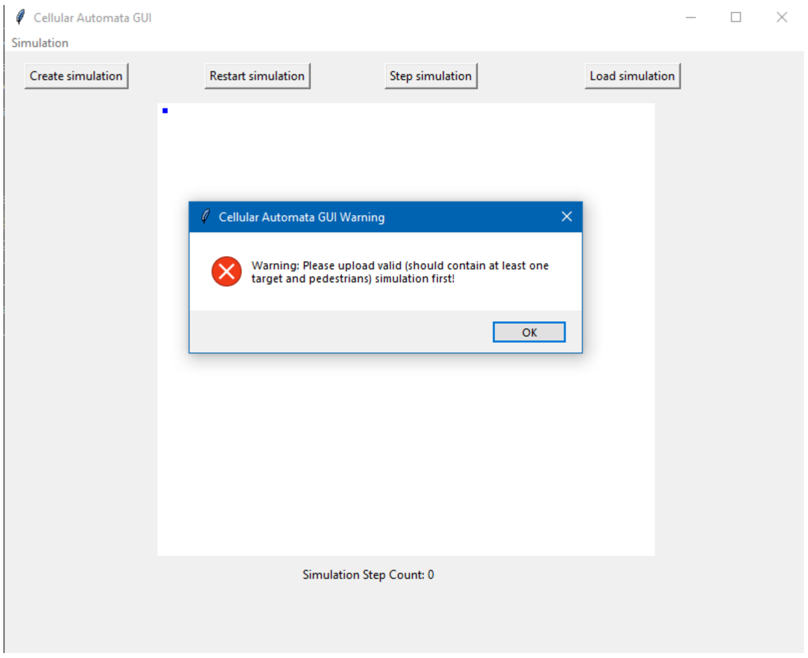


Select: TARGET

Height: 100
Width: 100
Select: PEDESTRIAN
Location X: 30
Location Y: 56
Speed: 3

Figure 2: Manuel scenario creation

As can be seen in the figure below, a simulation can be created manually with the popup that opens when the Create Simulation button is pressed. Each time the Submit button is pressed, it adds the selected object and it is necessary to press the start button to start the simulation with these values. If the start button is **not pressed**, the test cannot be started properly and the following error is received.



Warning: Please upload valid (should contain at least one target and pedestrians) simulation first!

Simulation Step Count: 0

Figure 3: Error dialog box

In summary, the task has been accomplished effectively, resulting in a well-structured modeling environment for simulation. The utilization of a GUI interface and the flexibility of importing and defining scenarios through JSON files contribute to a user-friendly and adaptable system. Successful handling of pedestrians, targets, and obstacles creates a solid foundation for further exploration and development in subsequent project tasks.

Report on task 2, First step of a single pedestrian

Task 2 consists of having a pedestrian 20 cells away from a target. They are located at coordinates (5, 25) and (25, 25), respectively.

A grid will be created to represent the scenario exactly as described. You can observe the pedestrian walking in a straight line toward the target at a fixed speed figure (4. In this simulation, each cell corresponds to a distance of 1 meter. The pedestrian's walking speed is set at 1 meter per second, which means that each step along the cardinal axes covers 1 meter and takes 1 second to complete. Additionally, diagonal steps cover approximately 1.4 meters and require around 1.4 seconds. As anticipated, in the experiments, the pedestrian reached the target in 20 seconds (steps).

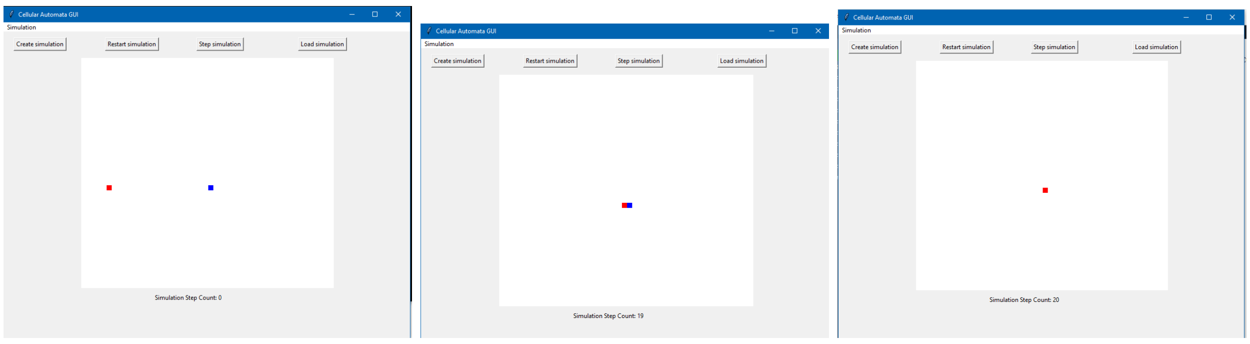


Figure 4: Pedestrian walking in a straight line

Report on task 3, Interaction of pedestrians

In this simulation task, the objective was to place five pedestrians equally spaced on a circular path around a single target located at the center of the scenario (as depicted in 5). The primary goal was to demonstrate that all pedestrians could reach the target at the same time, while also introducing the concept of walking speed into the simulation.

Before addressing the notion of walking speed, it's worth noting that the initial movement of pedestrians involved selecting neighbor cells with the minimum distance to the target, without considering speed or obstacles. This formed the basis of the simulation.

The concept of walking speed was subsequently introduced, defined as the distance a pedestrian can travel in one tick. A tick represents a single step in the simulation. The introduction of walking speed meant that pedestrians could progress varying distances in each iteration. For instance, a pedestrian with a walking speed of 1 could reach (0, 2) in two ticks, while a pedestrian with a walking speed of 2 would cover the same distance in just one tick.

However, considering that both speed and distance can be real numbers, the relationship between speed and distance became more intricate. Pedestrians needed to keep track of the real distance traveled in one step, allowing for the remaining distance to be saved for the next step. This approach accommodated diagonal movements as well.

Now, turning to the simulation scenario, it had dimensions of 100x100, with the target placed at position (50, 50). The five pedestrians were strategically placed in locations such that their Euclidean distances from the target were equal. All pedestrians were assigned the same walking speed, ensuring they could collectively reach the target at approximately the same time.

5 illustrates various stages of the simulation, demonstrating how pedestrians advanced toward the target. The goal of this task was for all pedestrians to reach the target, which was configured as absorbing, symbolizing the successful completion of their mission 5.

After simulating the scenario as described, all pedestrians reached the target at roughly the same time. The equal initial distances from the target allowed for an equitable outcome, exemplifying the effectiveness of the simulation in achieving synchronized pedestrian arrivals.

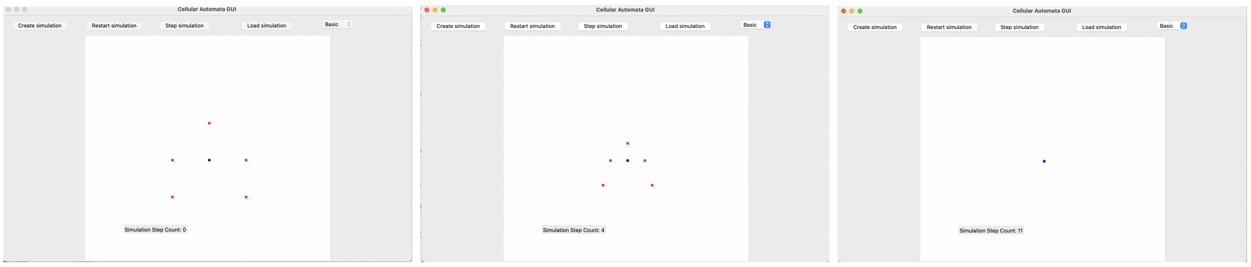


Figure 5: Pedestrians forming a circular arrangement

Report on task 4, Obstacle avoidance

In this task, our focus is on the implementation of various obstacle avoidance methods. In order to evaluate the effectiveness of these approaches, we observe the movement and behavior of pedestrians in two distinct scenarios, namely the "bottleneck" [1] and the "chicken test." In each of these scenarios, we place pedestrians in a uniform manner within their respective areas to simulate the environment. It is significant to note that, without any kind of obstacle avoidance implemented, pedestrians generally perceive other pedestrians and obstacles as empty spaces, leading to inevitable collisions between pedestrians and rendering obstacles as barriers.

The rudimentary obstacle avoidance implemented aims to prevent pedestrians from stepping on other pedestrians or obstacles, meaning that pedestrians strive to reach their target by moving in its direction. This effectively enables each pedestrian prior to deciding their next move, to construct a cost matrix, which has the same dimensions as the grid and where each element represents a cell. The value assigned to each cell corresponds to the Euclidean distance between the target and that particular cell. Subsequently, when planning their movements, pedestrians opt for the neighboring cell with the lowest associated cost.

As a result, basic obstacle avoidance involves pedestrians moving without any prior knowledge of obstacles, solely considering their path toward the nearest target. This is why the pedestrians in the "chicken test" scenario cannot successfully pass the obstacle, which can be observed in Figure 6., where the pedestrian is stuck in the trap at the end of the simulation.

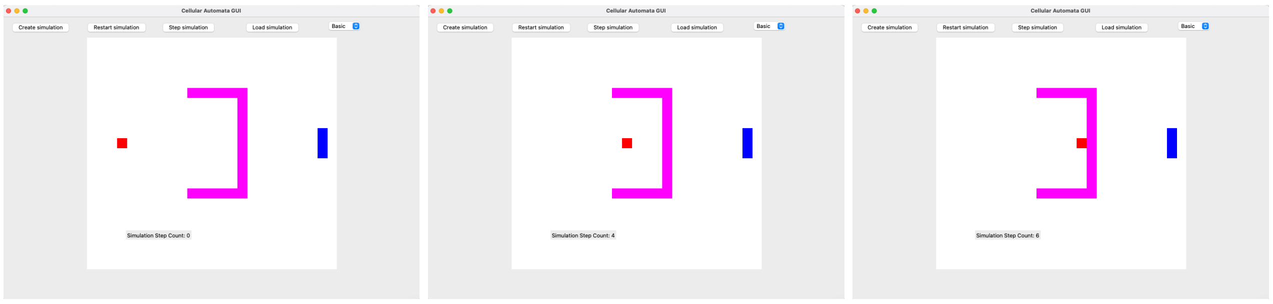


Figure 6: "Chicken test" with Rudimentary Obstacle Avoidance (a) setup of the chicken test; (b) pedestrian trying to reach the target; (c) pedestrian stuck in the trap.

On the other hand, they managed to successfully pass the bottleneck scenario (see Figure 7), as instead of moving straight towards the bottleneck for exiting the first square, they started moving to the right and slowly walk along the wall to the outlet.

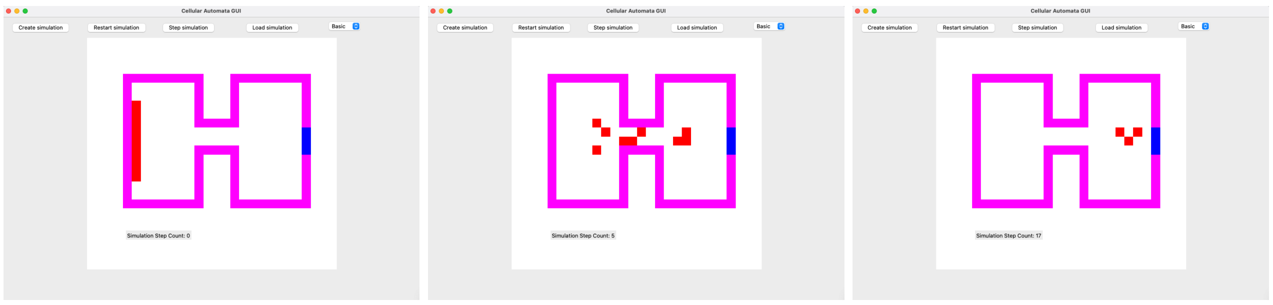


Figure 7: "Bottleneck test" with Rudimentary Obstacle Avoidance (a) setup of the bottleneck test; (b) pedestrian trying to pass the outlet; (c) pedestrian reaching the target.

To address the issues identified in the "chicken test," we employ Dijkstra's algorithm (and later also Fast Marching Method) as a feasible solution. Its purpose is to determine the shortest path between two nodes in a graph. In the context of obstacle avoidance, each cell can be treated as a vertex with undirected edges connecting it exclusively to its neighboring cells. Dijkstra's algorithms starts from a source cell, which is the pedestrian calculating the cost matrix, and tries to reach a destination cell, which is the target, by systematically visiting the neighboring cells of the current cell by updating the cumulative cost to reach that neighbor from the source. With Dijkstra's algorithm implemented, the "chicken test" is successfully passed, as illustrated in Figure 8, where the pedestrian manages to avoid the U-shaped obstacle.

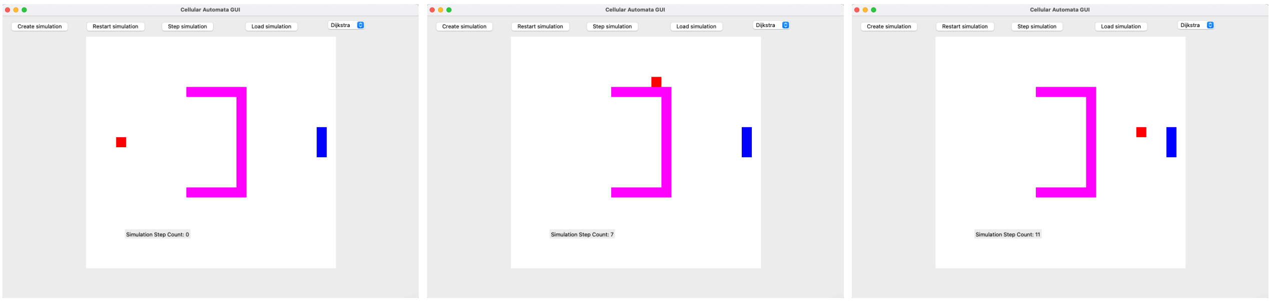


Figure 8: "Chicken test" with Dijkstra (a) setup of the chicken test; (b) pedestrian bypassing the obstacle; (c) pedestrian reaching the target.

It is significant to note that both the rudimentary obstacle avoidance and the Dijkstra implementation can help to reach the objective in the bottleneck scenario. They try to reach the bottleneck differently, one considering the Euclidean Distance, before going into the small corridor, the other one by finding the optimal path without moving the pedestrians to unnecessary directions. Figure 9. shows 3 different stages of the "bottleneck test" executed with Dijkstra's algorithm.

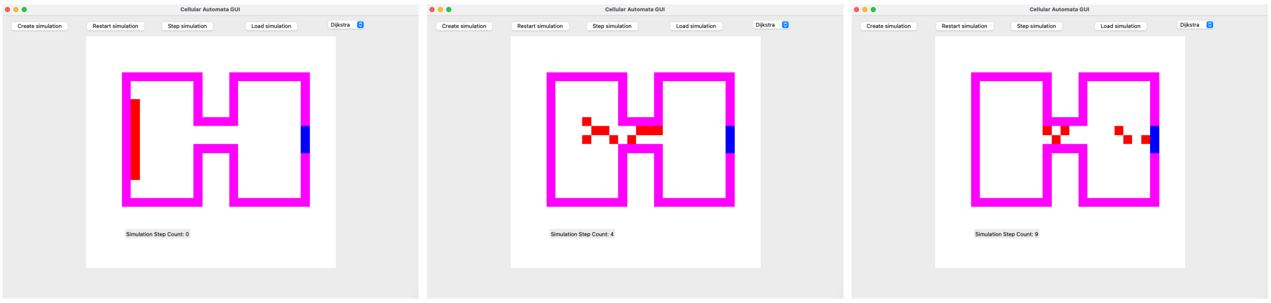


Figure 9: "Bottleneck test" with Dijkstra (a) setup of the bottleneck test; (b) pedestrian trying to pass the outlet; (c) pedestrian reaching the target.

In order to have a more accurate computation of the distance field, we also implemented the Fast Marching Method [2] (also ref. as FMM) in addition to Dijkstra's approach, using a third-party function provided in the library *scikit-fmm*. Figure 10. shows how the "bottleneck test" scenario is successfully passed using FMM algorithm.

As a result, it is possible to see that the rudimentary algorithm could not take any obstacles into consideration and failed in some scenarios, whereas Dijkstra and FMM were remarkably efficient.

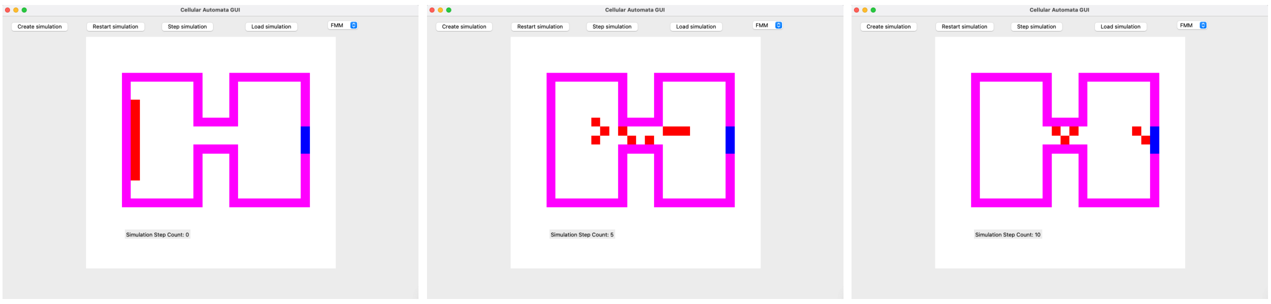


Figure 10: "Chicken test" with FMM (a) setup of the chicken test; (b) pedestrian bypassing the obstacle; (c) pedestrian reaching the target.

References

[RiMEA] [1] RiMEA Guideline for Microscopic Evacuation Analysis. RiMEA e.V., 3.0.0 edition, 2016. www.rimea.de.

[Fast Marching Method] [2] Fast Marching Method, Sethian, 1998 www.math.berkeley.edu/ sethian/2006/Papers/sethian.siam