Report for exercise 1 from group B

Tasks addressed:	5
Authors:	HORIA TURCUMAN (03752553) GEORGI HRUSANOV (03714895) UPPILI SRINIVASAN (03734253)
Last compiled:	2022–11–03
Source code:	https://gitlab.com/turcumanhoria/crowd-modeling/-/tree/main/1

The work on tasks was divided in the following way:

HORIA TURCUMAN (03752553)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
GEORGI HRUSANOV (03714895)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%
UPPILI SRINIVASAN (03734253)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	33.3%

Report on task TASK 1

The aim of the task was to create a basic simulation and scenario creation program. The simulation GUI is the one offered with the exercise. The contribution are:

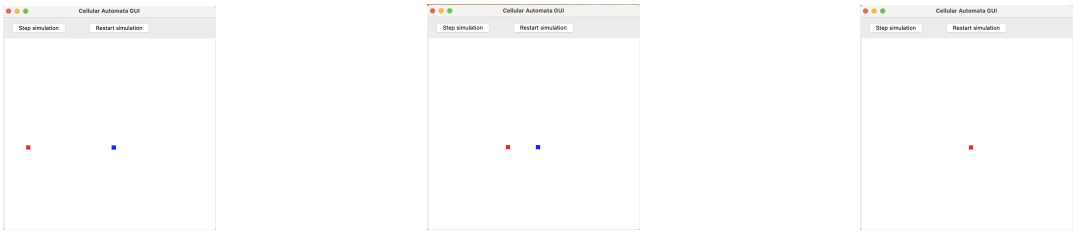
- Implementation of the simulation restart capability. This was accomplished by saving the initial state of the simulation, creating a new simulation with this initial state and wiring the step button to this new simulation. The initial state is saved as a dictionary in the same format as the scenarios are saved in json files.
- Implementation of a scenario creation CLI command which asks iteratively for data. The input is not checked for all possible errors (as negative numbers) since this would clutter the code without adding any benefit for our non-real-life project. Created scenarios can be later edited in the created json file under the scenarios folder. A creation process is shown in the image bellow.
- Implementation of a scenario loading and saving mechanism using json files as storage format. The structure is as follows:

```
{
    "width": 150,
    "height": 100,
    "pedestrians":[
        {
            "position": [23, 32],
            "desired_distance": 2.3
        },
        {
            "position": [50, 87],
            "desired_distance": 1.9
        },
        {
            "position": [80, 87],
            "desired_distance": 2.0
        }
    ],
    "targets":[
        [75, 75],
        [75, 33]
    ],
    "obstacles":[
    ]
}
```

Report on task TASK 2

At this point in the implementation, the pedestrians are allowed to step on the target. The saved scenario data which can be found in "scenarios/task 2" is:

```
{
    "width": 50,
    "height": 50,
    "pedestrians":[
        {
            "position": [5, 25],
            "desired_distance": 1.0
        }
    ],
    "targets":[
        [25, 25]
    ],
    "obstacles":[
    ]
}
```



(a) Starting state (b) State after 10 steps (c) Final state (after 25 steps)

Figure 1: Steps in simulation of task 2

It can be seen that the pedestrian travels on a straight line (shortest path) towards the target and waits there.

Report on task TASK 3

Placing the pedestrian on a circle was realize through in a short jupyter notebook that by placing a point at angle 0 on a radius=50 ball and moving $2 \cdot \pi/5$ in angle around the circle.

Task 3

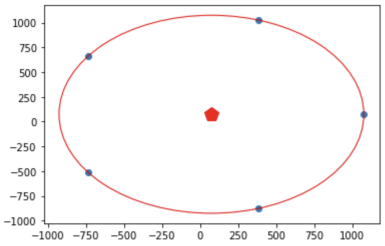
Generate points in a circle around (75, 75)

In [10]:
```
import math
import matplotlib.pyplot as plt
```

In [14]:
```
x = []
y = []

for i in range(0, 5):
    x.append(75 + math.cos(2 * math.pi * i / 5) * 1000)
    y.append(75 - math.sin(2 * math.pi * i / 5) * 1000)

print(x, y)
```

```
[1075.0, 384.01699437494744, -734.0169943749473, -734.0169943749474, 384.0169943749472] [75.0, -876.0565162951535, -512.7852522924733, 662.785252924733, 1026.0565162951536]
```

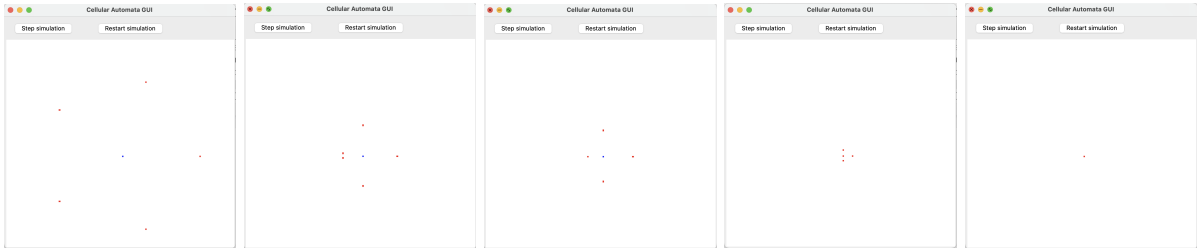
In [19]:
```
plt.scatter(x, y)
plt.plot(75, 75, 'rp', markersize=14)

circle = plt.Circle((75, 75), 1000, color='r', fill=False)
plt.gca().add_patch(circle)

plt.show()
```



The scenario defining file is:

```json
{
    "width": 150,
    "height": 150,
    "pedestrians": [
        {
            "position": [125, 75],
            "desired_distance": 3.0
        },
        {
            "position": [90, 27],
            "desired_distance": 3.0
        },
        {
            "position": [34, 45],
            "desired_distance": 3.0
        },
        {
            "position": [34, 104],
            "desired_distance": 3.0
        },
        {
            "position": [90, 122],
            "desired_distance": 3.0
        }
    ],
    "targets": [
        [75, 75]
    ],
    "obstacles": []
}
```

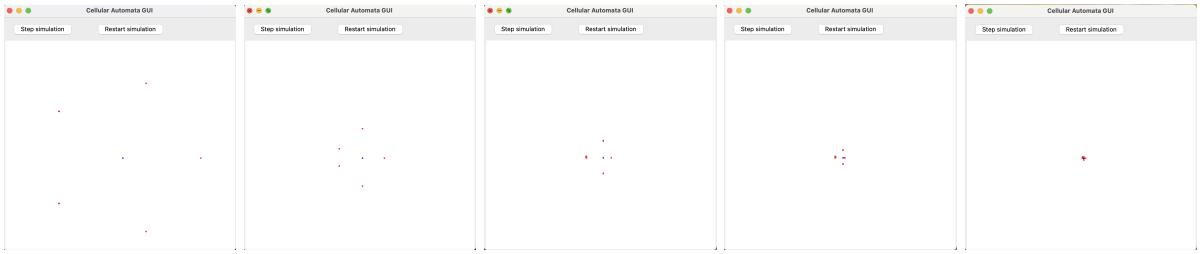
If one sets the target on 'absorbing' mode and allows for pedestrians to simultaneously step on the same cell, the simulation goes as in figure 2.



(a) (b) (c) (d) (e)

Figure 2: Steps in simulation of task 3

By this point avoidance is not implemented so pedestrians can be on the same cell with one another and the target. It can be seen that not all pedestrians reach the target at the same time. This is because even though they are located at the same (Euclidian) distance from the target, they do not move on straight paths towards it. In the current implementation the pedestrian can move to one adjacent cell including diagonal cells. On one hand, some pedestrians might require to go through more cells than other taking them more time. On the other hand, through a diagonal move the distance to the target becomes shorter than through a lateral move, reducing the required time.



(a) (b) (c) (d) (e)

Figure 3: Steps in simulation of task 3 for straight path movement

The implemented solution for this problem uses the option to specify, for each pedestrian, an Euclidian distance they are able to traverse during a time step. In one step, pedestrians are allowed to move through multiple cells as long as they have not moved a larger Euclidian distance than the distance per time step. This is implemented by setting each new time step a still-allowed moving distance and substracting from it each time the pedestrian moves. Diagonal moves of course are more expensive than lateral moves. In this way we make sure that pedestrians can move (if no obstacles) roughly the same distance during a time step. The simulation shows that now they all arrive at roughly the same time. Also, avoidance is now implemented and the pedestrians are no longer allowed to step on the targets.

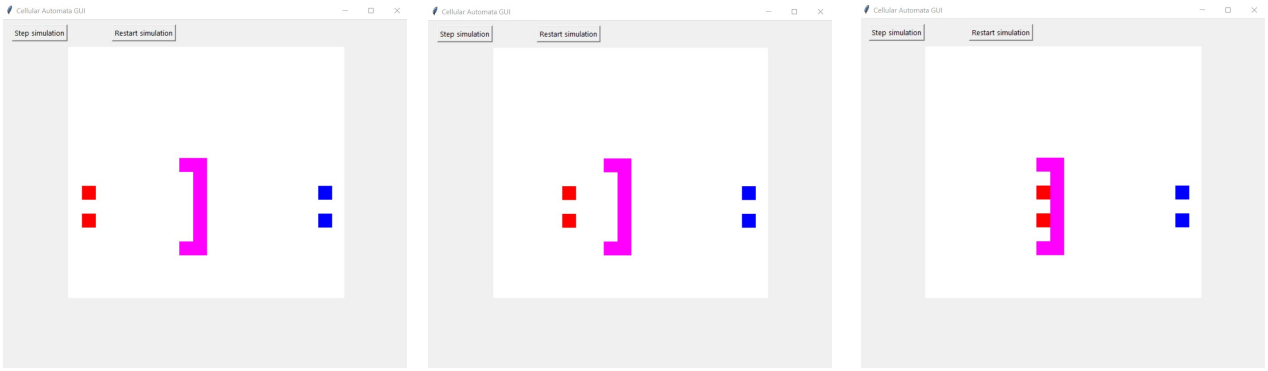
Report on task 4, Obstacle avoidance

No Obstacle Avoidance It is unavoidable for pedestrians to trip over one another and for obstacles to fail in their function as barriers if there is no obstacle avoidance system in place. Because walkers will see both other pedestrians and obstructions as "empty cells" in the absence of an obstacle avoidance system, For instance, in the scenario shown in Figure 10 from the RIMEA Guidelines [2], which depicts a bottleneck situation, the pedestrians would simply step on the obstacle and other pedestrians as if they were not there at all, which would cause them to avoid going through the bottleneck and instead proceed directly toward the target.

Rudimentary Obstacle Avoidance The basic obstacle avoidance system in this project (also called the *Rudimentary Obstacle Avoidance*) simply instructs walkers not to step on other pedestrians or impediments. The mechanism was built as part of this project. The usage of a cost matrix aids in the achievement of this goal. In the first, simpler design, pedestrians simply try to approach the target by walking in the direction it is facing. Pedestrians make their own cost matrices, which consist of a grid the same size as the matrix itself, with each object standing in for a cell in the matrix. The cost allocated to each cell is just the euclidean distance between the cell in the object and the cell in the target. When it comes time to plan the next move, the agent will select the nearby cell with the lowest cost. However, before doing so, the cost matrix must be post-processed. The expenses of any cells that are currently occupied (or will soon be occupied) by other pedestrians or barriers will be increased to an unlimited value during this step. In the case of the chicken test, the pedestrian (shown in red) will begin walking in the direction of the objective (shown in blue) until it is stopped by the obstacles (shown in pink), at which point it will remain in this position until the simulation is completed. Figure 4a shows the initial stages of setting up the chicken test, while Figure 4c shows how the simulation concludes with the pedestrian being trapped in the trap.

Dijkstra's algorithm The next significant task that we had to tackle in this task was the Dijkstra's algorithm, which is one of the most famous and used algorithms to find the shortest path between nodes [1]. The usage of Dijkstra's algorithm may show to be a viable approach for tackling the concerns raised by the chicken test.

The implementation of the Dijkstra's algorithm was done as followed. We would first initialize the Dijkstra, then define the Distance Matrix, then set the Distance Matrix to a high value, and finally set all unvisited states to 1. As a result, the distances between targets will be reset to zero, as will the values of the unvisited statuses of both targets and obstacles. We don't need to compute or alter those distances, so we skip this step. We build unvisited nodes (queue), which are nodes with an unvisited status of 1. These are the nodes we will look into deeper in order to identify the shortest distance between this node and the aim. The next stage would be to calculate the shortest distance between each of the nodes that have yet to be visited (queue). When the



(a) Starting state (b) The middle state (c) Final state

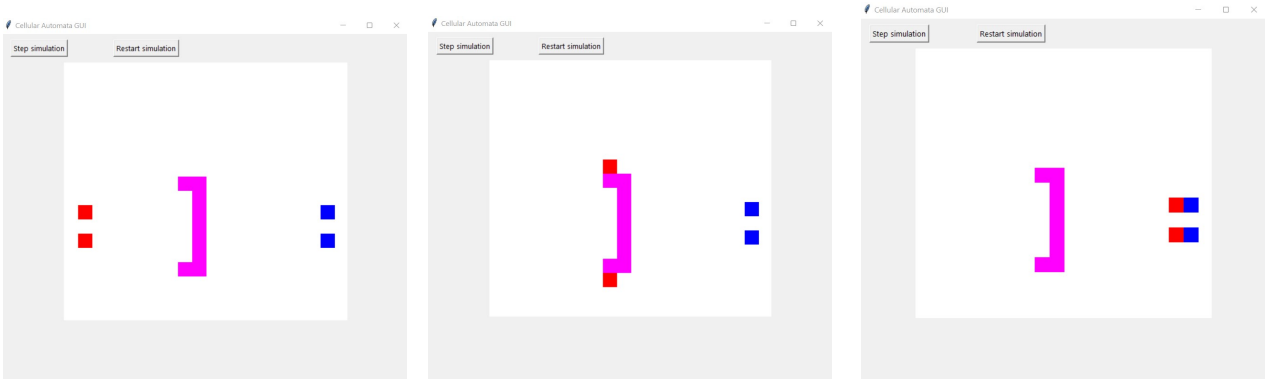
Figure 4: Steps in the Chicken Test Scenario with the Rudimentary Obstacle Avoidance that uses only the Euclidean Distance

Figure 5: The final State in sixth scenario from the RIMEA Guideline documenation [2], the Left Turn Scenario with the use of the Euclidean Distance

maximum distance to a certain node is reduced, the node's unvisited state is transformed to the visited state and reported as such. That node, however, has not yet been removed from the queue. To evaluate if the node should be removed from the queue, we first check to see if all of the node's neighbors have been visited. As a result, we make it a point to make sure that the neighbor who is the furthest away is not considered or chosen as the next step for pedestrians. The target grid needs to be updated, which is the very last thing on our to-do list. The distance matrix that is used to determine the next step can be obtained by using this function. It takes the user-defined boolean value from the JSON file and utilizes that for the dijkstra algorithm.

The purpose of this technique, as mentioned, is to find the shortest and most efficient path between two nodes in a network. It's possible to conceive of each cell in this case as a vertex, with undirected archs connecting it to only the cells close to it. This interpretation applies to the current circumstances. Dijkstra begins with a source cell, also known as the pedestrian calculating the cost matrix, and attempts to reach a destination cell, also known as the target, by iteratively visiting the neighbors of the current cell in the object and updating the overall cost to reach that neighbor from the source passing from a specific cell as the second-last one in the path. This method is repeated until Dijkstra arrives at the desired cell. The "chicken test" was successfully completed in this case. As shown in Figure 6, the red pedestrians are both able to make their way past the U-shaped barrier and arrive at their destination.

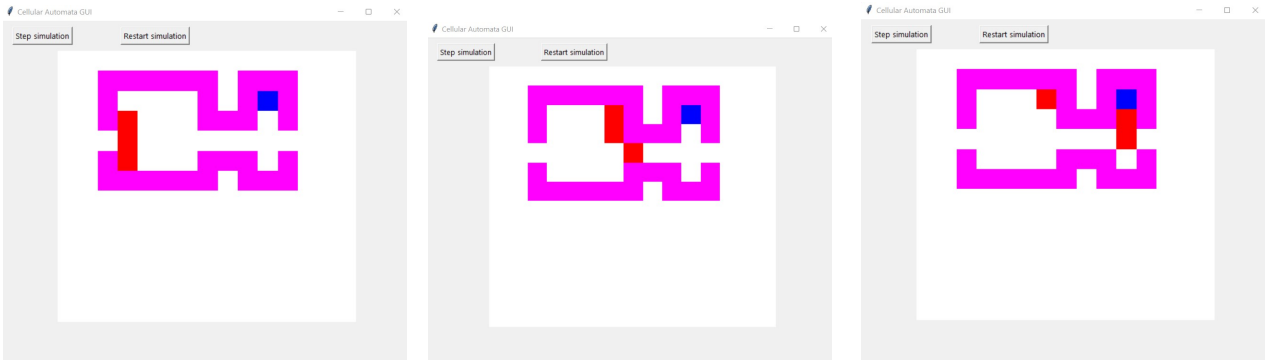
To perform a specific simulation using Dijkstra's technique to find the optimum path, you have to specify that in the json scenario file that you want to run. In each *json* file there is and object called ***use_dijkstra*** and the scenario is initialized with it. If this value is set to be false, the simulation will be run with the fundamental



(a) Starting state (b) The middle state (c) Final state

Figure 6: Steps in the Chicken Test Scenario with the utilization of the Dijkstra shortest path algorithm

obstacle avoidance algorithm that uses only the euclidean distance (described in the preceding paragraph), which selects the next cell along the path

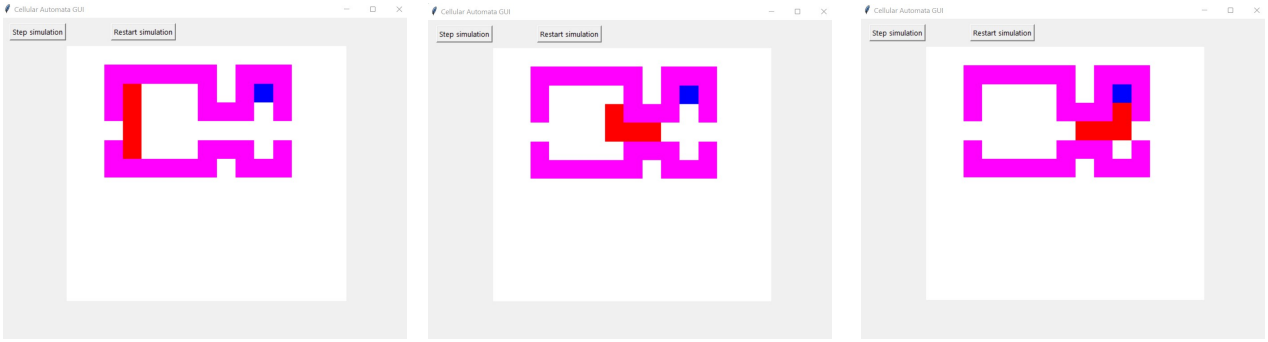


(a) Starting state (b) The middle state (c) Final state

Figure 7: Steps in the explicitly conducted Bottleneck Scenario with the Rudimentary Obstacle Avoidance that uses only the Euclidean Distance

Another noteworthy experiment that we thought of and might be performed is the bottleneck test. Both the simple obstacle avoidance method and the more complicated Dijkstra implementation are feasible options for reaching the goal. The method by which they achieve their goal is the source of the intrigue. In fact, by looking at Figure 7, one can observe how all of the pedestrians make an effort to get to the upper right corner of the bottleneck's leftmost section. This is because, before entering the narrow corridor, this cell is the closest to the target in terms of Euclidean distance. On the other hand, as illustrated in Figure 8, the algorithm ensures that the optimum path is promptly determined. This causes all pedestrians to form an organized queue without visiting any non-essential cells.

Fast Marching Method Implementing and setting up the *Fast Marching Method* was a bonus exercise within this task. J.A. Sethian is the creator of the fast marching technique algorithm, which was developed in the 1990s (1996a) [3]. The process is similar to Dijkstra's algorithm, which is used to identify the optimum routes through a graph (Dijkstra, 1959) [1]. The fast marching approach seeks a solution to a discretized variation of the Eikonal equation on a spatial grid with all dimensions the same. This solution takes the form of a fictitious arrival time for each grid point, with the origination point determined in advance. To accomplish this purpose, a speed function is built for each of the grid's points. This speed function is used to compute the arrival times of the remaining locations in the grid sequentially, point by point, starting from the initial point. Unfortunately however we did not have enough time to implement and try this approach in our project.

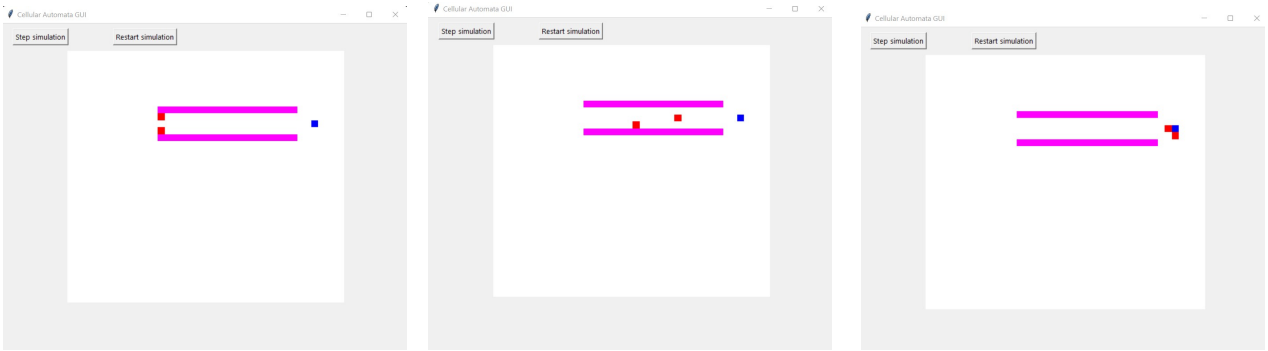


(a) Starting state (b) The middle state (c) Final state

Figure 8: Steps in the explicitly conducted Bottleneck Scenario with the utilization of the Dijkstra shortest path algorithm

Report on task 5, Tests

TEST1: RiMEA scenario 1 (straight line, ignore premovement time)
This is the first scenario mentioned in the RiMEA guideline[2] which is a guideline for microscopic evacuation analysis. There is a pedestrian at one end of the corridor (which is bounded on all sides by obstacles), and a target at the other. This could be seen both in Figure 9 and 10. As both the pictures show, both approaches are able to conduct successfully the test and the target cell is reached. In both cases A pedestrian is exactly one cell in size, and their walking speed is one meter per second. Furthermore, each cell is 1 m2 in size, which means that the cell width is equal to 1. Because the corridor is only 2 meters wide and 40 meters long, the pedestrian must take 39 steps to reach his or her destination. The results of the trials show that the simulation lasts exactly 39.0 seconds and that the pedestrian moves at exactly 1.0 meters per second.[2] -

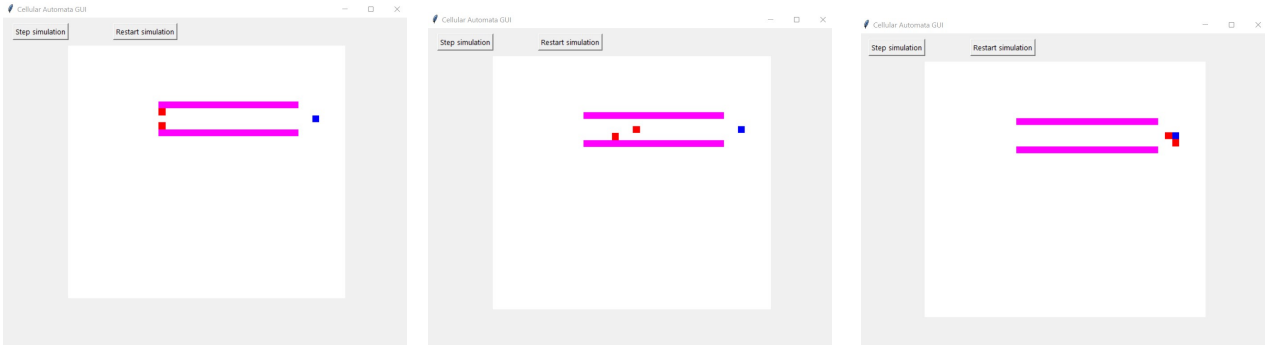


(a) Starting state (b) The middle state (c) Final state

Figure 9: Steps in first scenario from the RIMEA Guideline documenation [2], the Corridor Scenario with the use of the Euclidean Distance

TEST2: RiMEA scenario 4 (fundamental diagram, be careful with periodic boundary conditions).
Modelling and setting up this subtask and this test proved to be the most difficult to deal with, resulting in changes to the code that was suggested as well as the decision to accept the request. The first task requires a corridor that is 1000 meters long and 10 meters wide, with three unique measuring zones that are each 2 x 2 meters. Unfortunately, the implementation described does not meet all of the criteria. In fact, because there are so many cells and people, running the simulation is unfeasible because the program would either freeze or move at a glacial speed.

As a result, a few assumptions or approximations have to be used in order to maintain the viability of the simulation while keeping honoring the objectives of the scenario to the greatest extent feasible. The



(a) Starting state (b) The middle state (c) Final state

Figure 10: Steps in first scenario from the RIMEA Guideline documenation [2], the Corridor Scenario with the utilization of the Dijkstra shortest path algorithm

implemented scenario depicts a grid with 60x60 components, with a corridor that has 3x6 cells. The right side of the corridor is where the targets are located, while the left side has a density of 0.50, 1, 2, 3. The formula for calculating the density is $density = \frac{Pedestrians}{SquareMeters}$.

(a) Starting state (b) The middle state

Figure 11: Comparison of the corridor for the fourth RIMEA Scenario [2] with density 0.5 the starting and the ending state

(a) Density of 1 (b) Density of 2

Figure 12: Comparison of the corridor for the fourth RIMEA Scenario [2] with density 1 and 2 respectively

In order to achieve the density that we need for conducting several experiments according to the *RIMEA Guidelines* [2], we decided to have an area that is equal to 3x6 cells in form of a corridor. This will deliver us with an area of total $6 * 3 = 18m^2$ square meters surface. In order to achieve the density we want, we would have to have $9, 18, 36, 54$ pedestrians for the different testing scenarios. The speed is fixed and set to be equal to 2, so this will give us always an average speed of 2. Because using the primary obstacle avoidance technique causes a pedestrian to remain stationary only when surrounded by barriers or other pedestrians, the number of times a person stands still in this scenario is modest. In Figures 11 we can see the performance of the algorithm when the number of pedestrians is 9, hence the density is equal to 0.5. On the other hand we also have Figure 12, where we compare the behaviour when the density is 1 and 2

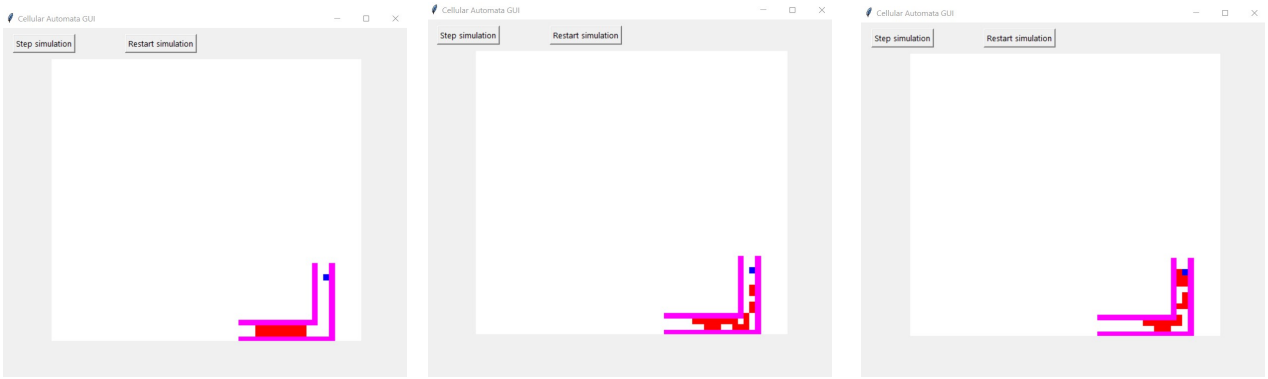
respectively. It is safe to say that for both cases when the density is 2 or 3, we will have the same result, because of the implementation we decided for reaching the goal state.

Unfortunately, we were unable to adapt our code and integrate the necessary functionality for the measurement zones. However, we thought about how we could do it, and one of the ideas that seemed to make the most sense was to develop a class that could record both the population density and the average speed of the pedestrians at each time step. We will be able to compute the average value for each of the three measurement zones once all of the pedestrians have arrived at their destination.

TEST3: RiMEA scenario 6 (movement around a corner).
The RiMEA scenario number 6 [2] has twenty persons making their way around a left bend to reach a target at the end of the corridor. The goal is to successfully travel the course without encountering any of the obstacles. This can be accomplished by employing either the "rudimentary obstacle avoidance mechanism" or Dijkstra's algorithm. Both are capable of accomplishing this task, as we will show later The first situation, depicted in Figure 13, depicts pedestrians making less accurate and "short-sighted" motions. It's understandable considering that the only criterion they use to decide which direction to take is the euclidean distance between the prospective destination cell and the target cell (of course the candidate destination cell must be empty).

(a) Starting state (b) The middle state (c) Final state

Figure 13: Steps in sixth scenario from the RIMEA Guideline documenation [2], the Left Turn Scenario with the use of the Euclidean Distance



(a) Starting state (b) Moving state (c) Moving state

Figure 14: Steps in sixth scenario from the RIMEA Guideline documenation [2], the Left Turn Scenario with the utilization of the Dijkstra shortest path algorithm

Instead, as illustrated in Figure 14, Dijkstra's algorithm guides walkers' attention to the most advantageous route. Again, this is expected behavior given that each pedestrian determines the best route for themselves and makes every attempt to follow it as closely as possible, with certain modifications allowed if specific cells are occupied by other pedestrians at a given time. It is possible that the persons on the outside of

Figure 15: The final State in sixth scenario from the RIMEA Guideline documenation [2], the Left Turn Scenario with the use of the Euclidean Distance

the optimal path who are the furthest away from the best route would try to go around the crowd in Figure 14c, but this would be true in a real-world scenario. The reason that pedestrians do not appear to take "different" paths may be explained by the fact that, in the scenario presented, each walker selects its own optimal path independently of the others, taking just the barriers and the goals into account. They find the best route, which is often hindered by other pedestrians; hence, they wait.

In the end as we can derive from both Figure 15 and from Figure 19 both approaches could be used in order to fulfill this testing scenario and they both deliver satisfiable results.

TEST4: RiMEA scenario 7

The RiMEA scenario number 7 [2] emphasizes the concept of altering pedestrian walking speeds based on their demographic factors. The various speeds are sampled from a predetermined interval based on the subject's age, with the youngest subject being three or four years old and the oldest subject being eighty years old. The following is a list of the lowest and highest speeds appropriate for each age group:

- For people aged 21 to 50, the speed ranges between (1.4 and 1.6) meters per second.
- 11 to 20 years old: (1.2 to 1.6) meters per second
- For people 51-70 years old, speed ranges between (1.1 and 1.4) meters per second.
- For children aged 3 to 10, the pace ranges between 0.6 and 1.2 meters per second
- Age 71-80: speed between (0.7 and 1.1) meters per second

A heuristic piecewise function approximation of Figure 2 from the RiMEA[2] benchmark yields the acceptable ranges.

The scenario's execution requires a method for sampling the speed of fifty pedestrians, as well as the initialization of a grid of sixty meters by sixty meters, with all pedestrians on the far left and targets on the far right. For better visibility and readability, every ten pedestrians, there are two cells that separates them. The goal is to measure the injected velocity of the walkers in order to observe and respect their speed. In terms of both visual appearance and statistical relevance, the proposed implementation makes it simple to achieve this goal; in reality, all that is required is the amount of time spent traveling to the destination and the distance traveled. Figures 17 and 18 depicts the executions visually, which may be understood by observing the varying speeds of the red dots. The slower red dots are most likely elderly people or tiny children.

It is critical to note that the execution of cellular automaton occurs sequentially rather than in concurrently. A graphical update occurs after all of the Pedestrians have had the opportunity to plan and execute a move (the order in which the Pedestrians choose to plan and execute their moves is randomized to maintain equality), and a call to sleep is made in between each of the graphical updates to simulate the passage of

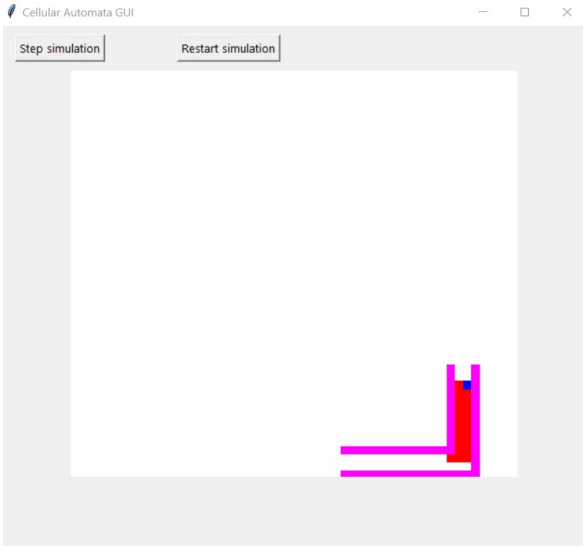
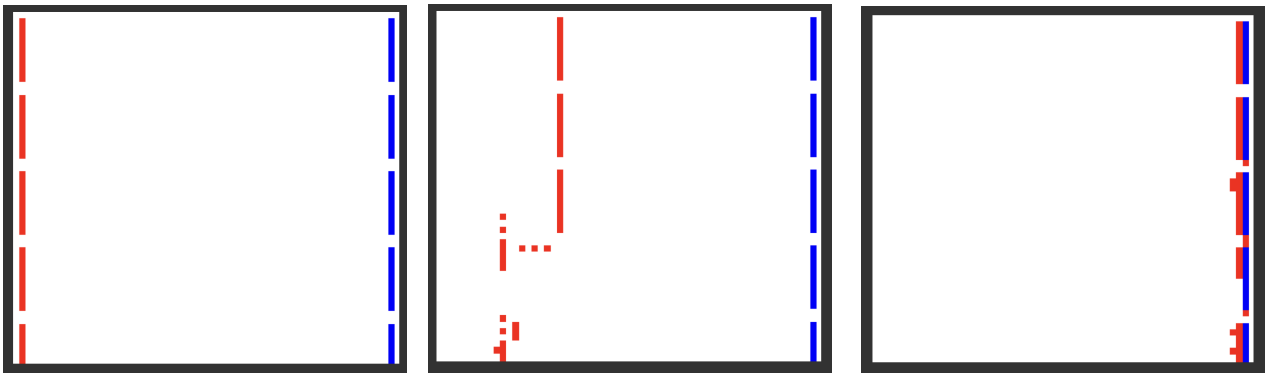


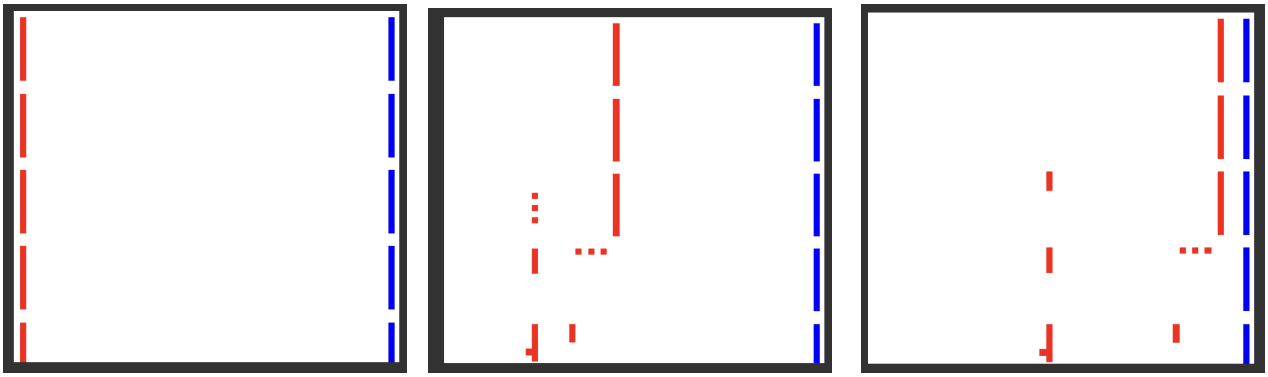
Figure 16: Steps in sixth scenario from the RIMEA Guideline documenation [2], the Left Turn Scenario with the utilization of the Dijkstra shortest path algorithm



(a) Starting state (b) The middle state (c) Final state

Figure 17: Steps in seventh scenario from the RIMEA Guideline documenation [2], the Demographic Scenario with the use of the Euclidean Distance

a time step. This prologue is necessary to understand in order to completely appreciate the discretization of time, which results in minute differences between a pedestrian's projected speed and their actual speed when measured. Regardless, the fact that the gap is related to the time step chosen is an intriguing finding



(a) Starting state (b) Moving state (c) Moving state

Figure 18: Steps in seventh scenario from the RIMEA Guideline documentation [2], the Demographic Scenario with the utilization of the Dijkstra shortest path algorithm

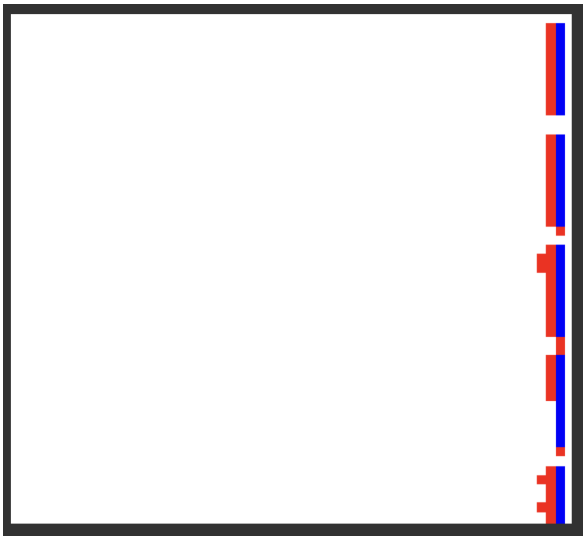


Figure 19: Final step in the seventh scenario from the RIMEA Guideline documentation [2], the Demographic Scenario with the utilization of the Dijkstra shortest path algorithm

References

[1] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[2] von RiMEA, Von Angelikakneidl, and Von Anwinkens. Rimea e.v., Oct 2022.

[3] James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93 4:1591–5, 1996.