Report for exercise 1 from group C

Tasks addressed:	5
Authors:	DANIEL MAYAU (03724644)
	AMELIE LASTNAME (MATRIKEL NUMBER)
	YALVAC LASTNAME (MATRIKEL NUMBER)
Last compiled:	2019–10–30
Source code:	https:// add link

The work on tasks was divided in the following way:

DANIEL MAYAU (03724644)	Task 1	0%
	Task 2	0%
	Task 3	50%
	Task 4	80%
	Task 5	30%
AMELIE LASTNAME (MATRIKEL NUMBER)	Task 1	75%
	Task 2	75%
	Task 3	50%
	Task 4	20%
	Task 5	10%
YALVAC LASTNAME (MATRIKEL NUMBER)	Task 1	25%
	Task 2	25%
	Task 3	0%
	Task 4	20%
	Task 5	60%

Report on task 1/5, Setting up the modeling environment

Amelie

Report on task 2/5, First step of a single pedestrian

§§§ Amelie
§§§

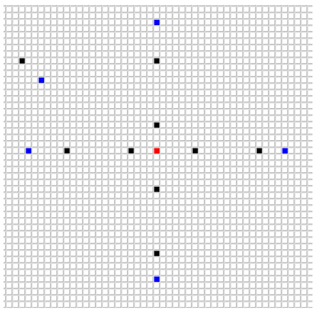
Report on task 3/5, Interaction of pedestrians

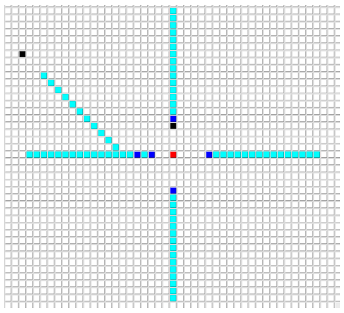
REPORT TEXT.

Report on task 4/5, Obstacle avoidance

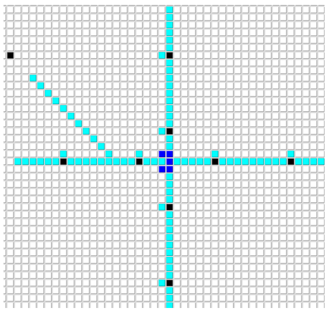
.

After pedestrian interracting with each others, the next step is to be able to avoid objects on their way. As in the following figures, the object are just runned through

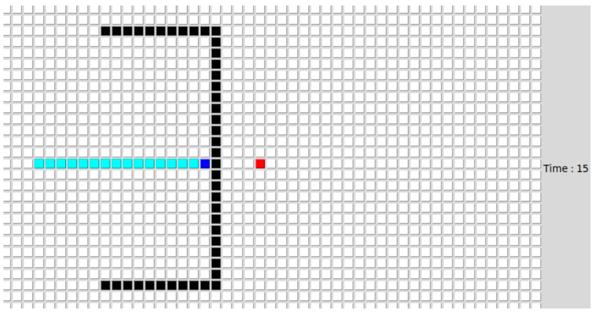


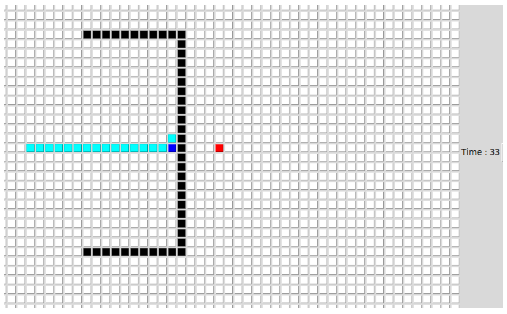


The first step is simply to consider obstacles to a distance infinity to the target, like it was done for the fellow pedestrians. Doing so allows us to avoid the problem of walking through the obstacles.



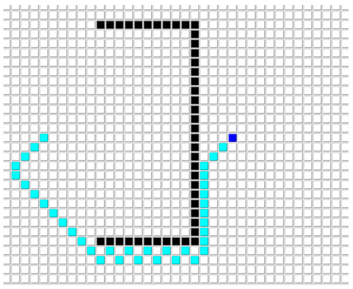
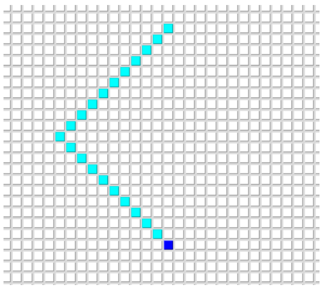
Nevertheless this implimentation is unsuffiscient to reach the target at any time. Indeed the implementation of the Chicken Test (following figures) shows that even though the pedestrian does not go through the obstacle, it gets stuck at the obstacle, for 15 moves in the figure but potentially forever. Therefore it is important to find another way to guide the pedestrian to its goal.



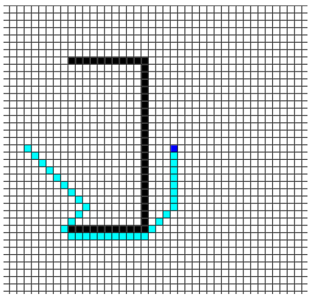
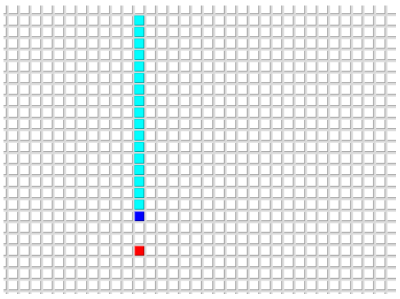


To achieve this goal, the Dijkstra algorithm seems to be correct. The main idea is to initialize all cells with distance infinity (so here -1) and then assigne distances by neighboors. In other terms, the distance is now the number of moves necessary to reach the target. One by one the neighboors to the cells distance x that are not obstacles take distance x+1.

Therefore we started computing a distance matrix in a distance matrix function, that we would use to update one step function. The first implementation of the the algorithm worked good to pass the chicken test and reach the target every time. Nevertheless the problem was about the itinary. Our computation had the pedestrian use the minimum number of cells but not the direct line

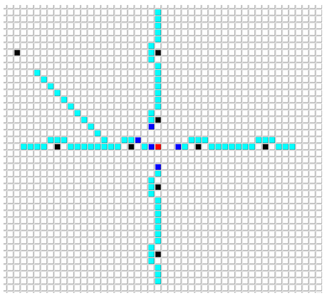
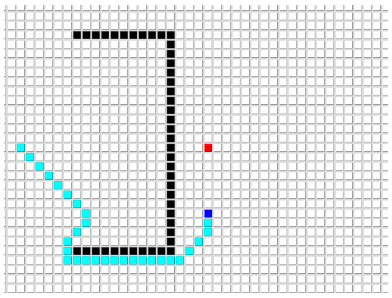



To solve this problem we decided to try a mix between the euclidean distance and the Dijkstra distance. We added to our distance matrix function the computation of the euclidian matrix as a decimal (we divided the euclidean distance by twice the maximum possible euclidean distance between 2 cells and then sumed it to the Dijkstra distance). This way between 2 cells that are at the same distance from the target with Dijkstra, the euclidean distance would make the difference. This mostly solved the previous problems:

Even though the Chicken test presented a slight deviation from the shortest path, because choosing between 2 cells it would take the closest to the target despite the fact that it might deviate a little from the obstacle, we decided that it was sufficient for now, and that we would implement a change if it would become necessary in the tests (which it did not). One way of solving this would be a reconstruction of distance considering the effort of moving to go diagonal compared to straight, or maybe try an other algorithm like the Fast Marching one.

The next challenge was to deal with the corners issue. indeed the shortest way arround a corner is diagonal but that would mean scratching or even goign through the obstacle. It was then important to make sure the pedestrian would walk arround the corner. To do so, the main idea was to recompute the neighboors, such that when an obstacle was adjacent to a pedestrian, the diagonal next to the obstacle would not be considered as a neighboor anymore. To do so, we totally rebuilt the neighboors computation function, with checking every direct (non diagonal) neighboor and deleting the whole line if the cell was either out of boundaries or an obstacle. This solved the corner issue :

Report on task 5/5, Tests

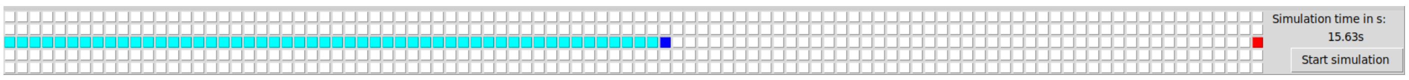
.

After implementation of the first tasks in a pretty free environnment, now was time to test it on real conditions following a guideline of Rimea.

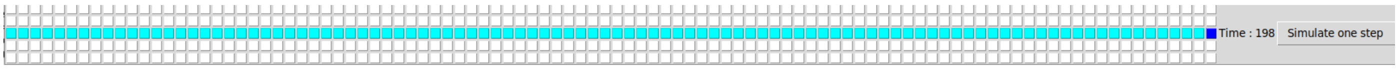
For the first test, we are set in a corridor of 40 meters long and 2 meters wide, with a speed of 1,33 m/s for one pedestrian of size 40cm*40cm, amd the test will play on the time to go from one end to the other of the corridor.
We decided to stick with our previous model, setting the size of one cell 40cm*40cm. This way for this test the size of the grid would be 5*100 cells corresponding to the size of the corridor.

The challenge for this test was to implement the speed of a pedestrian, which we had not compute yet. In order to do so, 2 methods were tried: one step by step, keeping the the manual click corresponding to a certain time, the other one by adding a timer and a duration to have a real time simulation.

The second one used a real time simulation. Let's assume here that our pedestrian has a speed of 1,33 m/s, this means 3,33 cell/s. The idea of this implementation was to have - in this very example - our pedestrian move one cell every 0,3 second. To do so, a while loop was implemented comparing the current time with the starting time. Every time the right amount of time was spent, the pedestrian would walk one step.

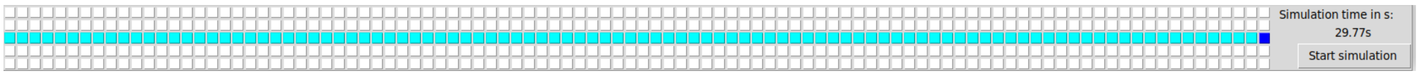


The test was implemented for the test step by step with a time per click of 0,2s and thus a speed of 0,66 cell/click. The result was as followed:



The time necessary is thus 198*0,2 = 39,6s to cross the corridor, so more than 34s. The test was failed.

With the real time simulation with a duration of 40s, the result was as followed:



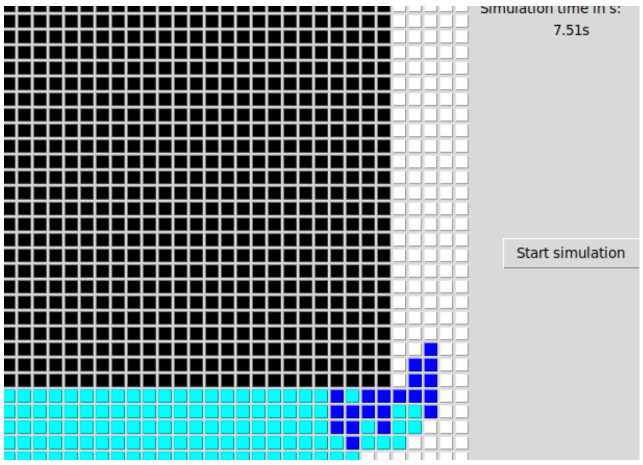
The time necessary is thus 29,77s, which is correctly between 26s and 34s. The test was a success.

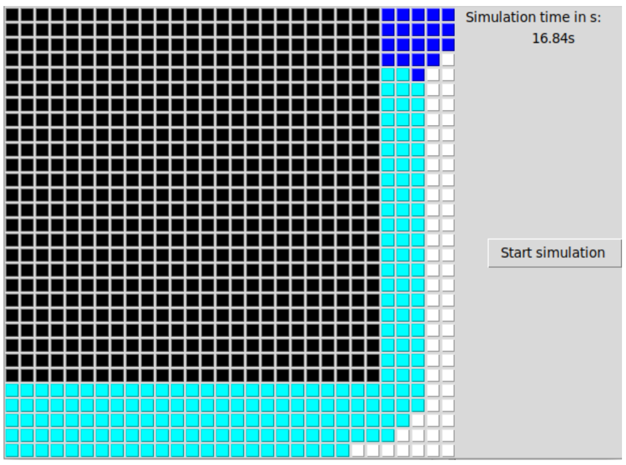
It happened the step by step methods had some flaws, but considering that we had a functional simulation, we decided to keep the second simulation, in real time, for the next tests.

The test 1 anyway was passed this way.

The test 2 ....

For the third test, the implementation required to put 20 people in a corridor and have them pass a corner. We kept a speed of 1,33m/s for each pedestrian and put them in the entrance of the corridor.





It is clear on the first image that the pedestrians do not cut the edges of the corner, and in the second picture we see that all of them did cross the corner in a normal time.

The test 3 is passed

The test 4 ....