Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Modeling of a passenger ship evacuation

Manuela Eugster, Andreas Reber, Raphael Brechbuehler, Fabian Schmid

Zurich
December 14, 2012

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Manuela Eugster   Andreas Reber

Raphael Brechbuehler   Fabian Schmid

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Modeling of a ship evacuation

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Eugster	Manuela
Brechbuehler	Raphael
Reber	Andreas
Schmid	Fabian

Supervising lecturer

Last name	First name
Balietti	Stefano
Donnay	Karsten

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

R. Brechbühler
A. Reber
F. Schmid
M. Eugster

Zurich, 20.11.2012
Place and date

Signature

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Contents

1 Abstract **6**

2 Individual contributions **6**

3 Introduction and Motivations **7**
3.1 Introduction . . . 7
3.2 Motivation . . . 7
3.3 Fundamental Questions . . . 7
3.4 Expected Results . . . 8

4 Description of the model **10**
4.1 Social force model . . . 10
4.2 Ship structure . . . 11

5 Implementation **12**
5.1 Basic code . . . 12
5.2 Code adjustments . . . 12
5.2.1 Exits . . . 13
5.2.2 Different exits . . . 13
5.2.3 Closing exits during simulation . . . 14
5.3 Crew command simulation . . . 16
5.4 Video creation . . . 16
5.5 System output . . . 17
5.6 Postprocessing . . . 17
5.7 Parameter input and config file . . . 18
5.8 Ship Decks . . . 18
5.8.1 Conversion . . . 18
5.8.2 Similarities and reduction . . . 20

6 Results and Discussion **22**
6.1 Simulation issues . . . 22
6.2 Simulation Results . . . 22
6.2.1 Standard ship . . . 22
6.2.2 Modified room disposition . . . 22
6.2.3 Modified rescueboat size . . . 24
6.2.4 Crew command . . . 24
6.3 Comparison . . . 25
6.3.1 Standard - Modified room disposition . . . 25

6.3.2 Standard - Modified rescueboat size 25
6.3.3 Standard - Crew command 26
6.4 General discussion 26

7 Outlook **26**

8 References **27**

9 Appendix **28**
9.1 Code 28
9.1.1 *standard* code 28
9.1.2 *C* code 53
9.1.3 *crew command* code 75

1 Abstract

The evacuation of a passenger liner is problematic due to a big mass of passengers wanting to reach the rescue boats. Real data can just be gained on accidents. Another way is to simulate such scenarios using computer models. Our approach was to take a common ship shape with real floor plans and run simulations on it with a continuous space social force model as introduced by Helbling et al [1]. A implementation of this has already been done by a former MSSSM group [2] on building structures. We took this C based code with a MATLAB interface and changed it to our needs.

Several scenarios were simulated. As a standard we took a simplified floor plan of the passenger liner Costa Serena, a 290 meter long ship with up to more than 4000 passengers. We compared it to modifications in the stair placement as well as rescue boat capacity adjustments. Another idea was to implement the leading and controlling influence of staff on the passengers.

Our simulations clarified that there is a potential to decrease the overall evacuation time. The most weightily influence had the stair displacement with up to 20% faster evacuation. on the other hand, the crew influence model seemed to be to simplified. We did not see a lot of improvement in this scenario. However, a more detailed model would probably lead to better results.

In a future work one could merge our modifications to get an even safer evacuation. The potential exists.

2 Individual contributions

The whole project was completed as a team. For sure we took into consideration all the personal backgrounds and knowledge. That is the reason why Raphael and Manuela focused on implementing the computer code. Whereas Andreas and Fabian concentrated on providing background information, compared the results with the reality and doing its verification. A more detailed breakdown can be seen by looking at the GitHub time line of our project.

3 Introduction and Motivations

3.1 Introduction

The evacuation of a passenger liner due to fire, sinking or other issues leads to several problems. A large amount of passengers try to safe their lives and get to a rescue boat. Narrow and branched floors, smoke, inflowing water, the absence of illumination, rude passengers and so forth can make the evacuation difficult and reduce the number of survivors. There are a lot of norms how to minimize the harm of such an evacuation. For example there are rules on the number of rescue boats dependent on the amount of passengers [3]. With dry runs the staff is prepared for the case of emergency et cetera. In real life ship corridor reproductions, the behavior of distressed people is studied. Another approach is to model such ship evacuations numerically on the computer. As an example the software maritimeEXODUS by a development team from the University of Greenwich is a computer based evacuation and pedestrian dynamics model that is capable of simulating individual people, behavior and vessel details. The model includes aspects of people-people, people-structure and people-environment interaction. It is capable of simulating thousands of people in very large ship geometries and can incorporate interaction with fire hazard data such as smoke, heat, toxic gases and angle of heel [6]. Our approach is similarly to model a passenger ship with a common geometrical outline and ground view. In an optimization process we will thereafter look for an ideal ground view, rescue boat distribution and their size to minimize the time needed for evacuation. Finally we will make a statement on possible improvements.

3.2 Motivation

Even though modern ocean liners are considered to be safe, the latest occasions attested that there is still potential for evacuation and safety improvements [7]. Certainly we know that this science is very advanced and practiced since the sinking of the Titanic. Nevertheless knowing that there are still bottlenecks on the ships we are very motivated to detect and eliminate them with our mathematical models.

3.3 Fundamental Questions

To find these bottlenecks we run a mathematical model of a ship structure with several decks and its passengers [5]. After we localized these places we are interested in the answers of the following questions:

How much time can be saved by varying the dependent variables mentioned below:

- How much people can be saved by changing the disposition of the specific room types?
- Where are the bottlenecks during the evacuation? How can they be avoided?

What is the influence of the rescue boats?

- Are small or bigger boats better?
- Where do they have to be positioned?

We analyze the difference between uncontrolled and controlled passenger flow:

- Is the crew able to prevent chaos in the evacuation process?
- What is the best way to lead the passengers out of the ship?

In addition we are keen to know if our model is a good abstraction of the reality.

3.4 Expected Results

Before making computer simulations we discussed, what we expect as results out of the simulation:

- Even though modern ships are quite optimized in regard to evacuation time, they are always a compromise between safety and luxury. Therefore we are convinced to find a superior adjustment of the decks geometries to increase the survival rate.
- Since the rescue boats can not be averaged but are rather concentrated over one or two decks, we consider the staircases as the bottlenecks.
- In aligning this variables we are persuaded of a reduction of the overall evacuation time.
- We suppose that smaller and evenly spread rescue boats combined with a higher quantity will scale the evacuation time down. Certainly there is going to be an optimum in size which we are willing to find.
- By controlling the rescue we assume to detect a huge decrease in evacuation time. Further we have the hypothesis that disorder can be minimized. The crew who is familiar with the decks and the emergency exits is able to guide the passengers in minimum time to the rescue boats.

- There are many parameters we do not model in our simulation. For example fire and smoke, the tilt of the ship or handicapped and petrified passengers are disregarded. By leaving out this details we get a very simplified model. However, by starting the optimization process by data of a nowadays passenger liner we hope to see some real evacuation dynamics in this system and therefore make conclusion on the fundamental questions.

4 Description of the model

We base our model on the work done by a group of former MSSSM students, by name Hans Hardmeier, Andrin Jenal, Beat Kueng and Felix Thaler [2]. In their work "Modeling Situations of Evacuation in a Multi-level Building" they wrote a computer program in in the language C with a MATLAB interface to rapidly simulate the evacuation of multi-level buildings.

4.1 Social force model

Our approach is the social force model described by Helbling, Farkas and Vicsek [1], as summarised below. In order to observe an escape panic situation and especially bottlenecks, they built up a continuous-space model. In mathematical terms they took Newtons second law and introduced a mix of socio psychological and physical forces for each so called pedestrian.

$$m_i \frac{\mathsf{d}\mathbf{v}_i}{\mathsf{d}t} = m_i f_D + \sum_{j(\neq i)} f_{ij} + \sum_{W} f_{iW} \tag{1}$$

Each pedestrian has his own mass and a direction e_i^0 in which he wants to move with a certain velocity v_i^0. He tries to adapt his own velocity to the wanted velocity with a given characteristic time τ_i.

$$f_D = \frac{v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t)}{\tau_i} \tag{2}$$

There are additionally interaction forces f_{ij} between the pedestrian and other people including an exponential part for the tendency of two pedestrians to stay away from each other. The second and third term are zero if the two pedestrians are not in touch. Elsewise there is a force in tangential direction t and in radial direction away from each other.

$$f_{ij} = \left\{A_i exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\right\} \mathbf{n}_{ij} + \kappa g(r_{ij} - d_{ij}) \triangle v_{ji}^t \mathbf{t}_{ij} \tag{3}$$

To be able to handle with walls there is another forces f_{iW} introduced. Its direction is away from the surrounding walls and the structure is similar to the one for the interaction between pedestrians.

$$f_{iW} = \left\{A_i exp[(r_i - d_{iW})/B_i] + kg(r_i - d_{iW})\right\} \mathbf{n}_{iW} - \kappa g(r_i - d_{iW})(\mathbf{v}_i \cdot \mathbf{t}_{iW})\mathbf{t}_{iW} \tag{4}$$

4.2 Ship structure

To apply the force model on a realistic situation a ship has to be implemented as well. Floor plans from the Costa Serena were taken [5]. In order to make strong conclusions we want to keep the following variables independent:

- Number of passengers (4400 agents)
- Overall capacity of the rescue boats (4680 seats)
- Ship size (290 meters long) and outer shape

In order to optimize the evacuation time, we change the following dependent variables:

- Stairs and their positions
- Rescue boat size, number and position
- Control of the passenger flow by crew members (e.g. is there staff to lead the passengers and how are they doing it?)

5 Implementation

As mentioned above our code is based on the work of a previous project. Working on a different problem statement we had to make several adjustments and adapt it to our simulation model. In this chapter we will explain the most significant modifications we made. For the exact understanding of the original code and the process of optimization, please refer to the documentation of the previous project group [2], especially chapter five and nine or our code in the Appendix.

5.1 Basic code

The builders of the code we base on created a flexible model to simulate building structures. The floors can be feed as graphic data into the system where different colors stand for different areas. Black are walls, red stairs up, blue stairs down, green exits and purple agent spawning areas. The data is evaluated in MATLAB in matrices. Information on parameters such as timestep, simulation time, force characteristics et cetera can be read in using a config file which is evaluated on simulation start.
With a fast sweeping algorithm in C a vector field is created for every floor pointing in direction of the shortest way towards stairs and exits. In a loop over all passengers the acting forces on them are calculated and via forward Euler converted into velocities.

$$\mathbf{v}_{i,new} = \mathbf{v}_{i,previous} + \frac{\sum f}{m_i} dt \tag{5}$$

Again with the forward Euler scheme positions are calculated that were reached in a finite timestep.

$$\mathbf{r}_{i,new} = \mathbf{r}_{i,previous} + \mathbf{v}_{i,new} dt \tag{6}$$

5.2 Code adjustments

This subsection is intended to illustrate the modifications in the code. We explain the difference between our model and the one used in [2] and the reason why a modification was necessary. Further we define the assumptions we made to minimize the modifications. Finally we specify the modifications and where they take place.
Those parts are however not essential for the results of our research. They are meant to be an assistance for following projects.

5.2.1 Exits

Reason:
The first change which was necessary was due to the fact that the exits in our simulation are not simply on the lowest floor.
Assumption:
In case of an emergency all agents are keen to leave the ship as fast as possible. Therefore in our simulation all passengers above the exits floor are only enabled to move down and passengers lower than the exit floor move upstairs.
Function:
applyForcesAndMove.m
New variables:
To define the floor in which the exits are we introduced a new variable *floor_exit* in the config file.
Modifications:
Since we defined our exit, we are able to easily modify the code by splitting the loop, in which the force-calculation and the movement of the agents take place, in two parts. First we loop over all floors higher than the exit floor. In those the agents are only allowed to move down. Secondly we do the same for all people in the floors lower than the exit floor where the passengers only can move up. The only floor left is the exit floor where the agents only move towards exits.
Because of this modification it is not necessary to loop twice over all floors. Consequently our code remains fast and efficient. We also kept the simple concept of vectors out of booleans. This means for each agent there is one number set:

- If the agent reaches a staircase and therefore changes the floor 1 is stored in the logical array *floorchange*
- otherwise a 0 is stored

The assumption is also a very helpful simplification for the pictures since we do not have to mess with the problem of stairs overlapping.

5.2.2 Different exits

Reason:
In our model the exits are the rescue boats, which can only hold a limited number of agents. Therefore we had to find a way how to differ the exits from each other and to assign a specific number to every exit. This number defines how many agents can stay on that rescueboat.
Function:
loadConfig.m

New variables:
- *exit_count*, to define the number of exits.
- For each exit k a varialbe *exit_k_nr*, to define the number of agents it can hold.
- To store how many agents can exit in one specific exit, we introduced a matrix *exit_nr matrix*, where the number of agents that can exit is indicated for each pixel.

Modifications:
Some changes in the decoding of the pictures were necessary. The aim was to change as little as possible to the original code. It was clear that we are going to need as many different colors as we have different exits, to be able to distinguish them during the simulation.
We define every pixel which has red to value=0, blue value=0 and green value unequal to zero a "exit-pixel". Now we can specify a lot of different colored exits by using green values between 256 and 256-*exit_count*.
We implemented the matrix *exit_nr* similar to the already existing one *img_exit*, in which we store a count to number the different exits. The number is defined by the green value of the pixel that belongs to this exit.

```
%make a zeros matrix as big as img_exit
config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));
% build the exit_nr matrix
config.exit_nr = config.exit_nr + e*( img_build(:, :, 1) == 0 & img_build(:, :, 2) ==
(256-e) & img_build(:, :, 3) == 0 ) ;
```

Figure 1: Implementation of the exit_nr matrix

5.2.3 Closing exits during simulation

Reason:
To close an exit as soon as it let a specific number of agents in, we have to keep track of the number of agents that already used this exit.

Function:
loadConfig.m and *applyForcesAndMove.m*

New variables:
exit_left

Modifications:
For this purpose, we defined the matrix *exit_left*, in which we store the number of agents who can exit for every exit.

```
%make a zeros vector as long as exit_count
config.exit_left = zeros(1,config.exit_count);
%loop over all exits
for e=1:config.exit_count
%build the exit_nr matrix
config.exit_nr = config.exit_nr + e*( img_build(:, :, 1) == 0 & img_build(:, :, 2) ==
(256-e) & img_build(:, :, 3) == 0 ) ;
%build the exit_left matrix and save the number of agents the exit can hold
config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));
end
```

Figure 2: Implementation of the exit_left matrix

In the loop where the forces and velocities are updated, (*applyForcesandMove.m*) we added a piece of code, which updates the exit_left matrix at every timestep. First, we get the number of the current exit.

```
%save current exit nr
data.current_exit = data.exit_nr(round(newp(1)), round(newp(2)));
```

Figure 3: Implementation to get the current exit number

Then we update the *exit_left* matrix by counting down the number of agents allowed to exit by 1. If the allowed number of agents exited the number is 0. Now we have

```
%update exit_left
data.exit_left(1,data.current_exit) = data.exit_left(1,data.exit_nr(round(newp(1)),
round(newp(2)))) - 1;
```

Figure 4: Implementation to update the exit_left

to close the current exit, by changing it into a wall. Therefore we have to update the *img_wall* matrix.

```
%close exit if there is no more free space
if data.exit_ left(1,data.current_exit) < 1
%change current exit to wall
data.floor(data.floor_exit).img_wall = data.floor(data.floor_exit).img_wall == 1 ...
— (data.exit_nr == (data.current_exit));
data.floor(data.floor_exit).img_exit = data.floor(data.floor_exit).img_exit == 1 ...
& (data.exit_nr  = (data.current_exit));
```

Figure 5: Implementation to close filled boats

5.3 Crew command simulation

Reason:
In reality an evacuation is coordinated. Loudspeaker announcements and crew members lead the people to the exit floor. In some cases there are even assembly points on the deck where people meet. Afterward they go with crew members to the rescue boats. In our first simulations we saw, that the last passengers escaping the ship have to walk all the way along the ship to get into a rescue boat in the middle of the deck. Those boats are the only ones that are not fully loaded at this time. To counteract, we tried to implement somehow the interaction of the passengers and the crew members.
Assumption:
In order not to change to much the structure of our standard evacuation code, we tried to implement the scenario in two steps. First just some rescue boats in the middle of the ship are available what can be interpreted as a crew leading people to those boats because they know that the boats near to the stairs should be left unalloyed for the last passengers to be evacuated rapidly. In a second step, all rescue boats are opened when a certain number of passengers already left the ship.
Function:
The whole basic code has been copied and adjusted in several positions, mainly in loadConfig.m and applyForcesAndMove.m.

5.4 Video creation

Reason:
In the basic code there was a graphical output into eps-formated files possible on every timestep. The images then had to be converted into a video file in a postprocessing part. This is not an optimal implementation for a fast system. Furthermore

it turned out that we could increase the simulation speed by factor two just by saving not on every single timestep but only on every 10^{th} or 100^{th}. Another issue was the conversion from eps to a video file. To simplify the handling, we used a MATLAB function to create directly a video out of the figures instead of making a detour via images.

Function:
simulate.m, initialize.m
New variables:
save_frame

5.5 System output

Reason:
Storing the simulation output is important to be able to analyze and optimize the system. The output of the basic code was just a plot with agents left the building over time and the above mentioned graphical output of the floor plans over time. We extended the output and created a dump struct where all the important variables are listed over time. That struct is the system output at simulation end and it can be used in a postprocessing part to generate meaningful graphs and data.
Function:
simulate.m
New variables:
output

5.6 Postprocessing

Reason:
To analyze the gathered data in a systematic way and to create consistent plots we created a postprocessing file. The output file can be loaded in MATLAB using the load command and is thereafter automatically evaluated. Plots with agents per floor over time, left space in rescue boats over time and agents that left the ship over time are created with proper axis label and so on. Additionally there is an output in the MATLAB command window with timestep, number of steps, total simulation time, agents on ship on start, agents on ship on simulation end, agents deleted due to Not-a-Number-positions and the characteristic t_{10}, t_{50}, t_{90} and t_{95} times.
Function:
postprocessing.m
New variables:

none of relevance
Modification:
We created the whole script ourself. It can be found in the annex.

5.7 Parameter input and config file

The social force model can be adjusted using different parameters e.g. to weight the importance of different forces among each other or to define the decay rate of these forces over the distance. Also the mass and radii of the passengers as well as their maximum velocity can be set. The parameters are written down in the config files that are passed to the MATLAB interface on simulation start. For sake of simplicity we adopted the force model parameters from the previous group. The parameters concerning the ship shape were adjusted so that they represent reality as far as possible.
In the annex a typical simulation input of the config file and some explanations can be found.

5.8 Ship Decks

The Costa Concordia has 14 decks which are all different from each other. They are connected by stairs and elevators in different configurations and they fulfil different purposes. There are decks for entertainment, eating, shopping, sports and so on. Due to the big differences of each deck, it is enormous time consuming to implement all these decks with all details in a model. So it is necessary to simplify the decks in a reasonable manner. Further the picture source is not perfect in size and data type so there is a manual conversion needed. In this chapter it is shown what assumptions were made and with which conversion techniques the decks were implemented into the model.

5.8.1 Conversion

First of all the decks have to match each other in pixel size and position to allow a flawless connection of the decks. So the size of some decks has to be adjusted and the stair overlay has to be matched as good as possible. Secondly doors, numbers, names and symbols are removed to have one connected surface without unreal obstacles.
Now that the surfaces are clean and connected, the colors have to be replaced by predefined colors of the code (purple for spawn of agents, black for walls, red for upstairs, blue for downstairs and different green type for each rescue boat).
In figures 6 and 7 a small portion of a deck with stairs, rooms, doors and elevators is

shown before and after conversion. In the converted picture is a blue block included for the stairs, which conversion will just be explained in the next part.

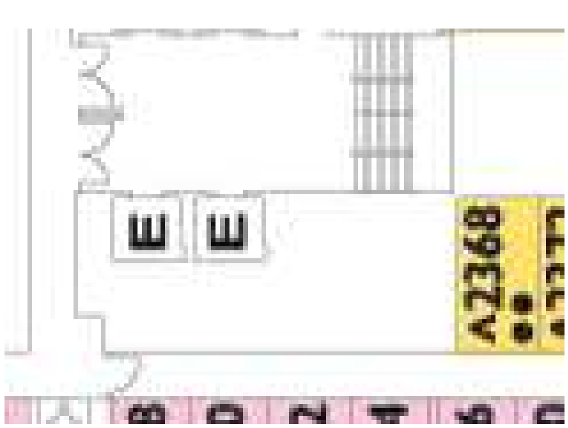

Figure 6: Deck before conversion

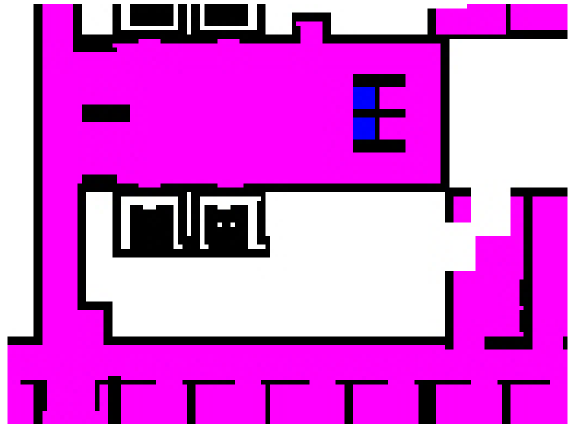

Figure 7: Deck after conversion

In most of the cases stairs from one to another floor are designed in a spiral way to minimize the consumed space. That leads to a problem by implementing the stairs into the model. It is not possible to make a clean transition from one floor to another without using some tricks.
There are special additional floors to overcome the mentioned problem in the work "Modeling Situations of Evacuation in a Multi-level Building"[2]. These additional floors which represents the stairs, are a fine method to create an infinite amount of floors and connections without any trouble. But it needs an additional floor for each connection.
While the agents on the ship march only in the direction of the rescue floor the connection can be simplified. So there is a new technique used to get rid of these additional floors.
In figure 8 a spiraling stairs connection between floor 01 and floor 02 is shown as example. The arrows in light blue are used to clarify the technique.
The agents arrive from the left on floor 11 and go into the direction of the arrow until they reach the blue box. There they are skipped down on the floor 10 and try to go to the blue box again. So they have to go around to be skipped down again on floor 09 and go around in the other direction. That continues until they reach floor 04 which is the floor with the rescue boats.

With that technique no additional floors are needed and we get very close to the real spiraling stairs. For the ship in that project this is sufficient, because the modeling fault will be in a lower degree than the one of other assumptions.

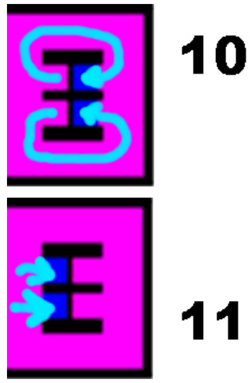

Figure 8: Stairs connection between floor 10 and floor 11

5.8.2 Similarities and reduction

The most important floor for the simulation is of course floor 04 because it holds all the rescue boats as seen in figure 10. There all agents have to pass and it is reasonable to have it very detailed. Floor 05 and 03 are the ones just above and beyond and will also have a non negligible influence on the agents movement.
The further we move the less important the deck configuration gets, because it is assumed that the stairs will be the bottlenecks. As long as the stair setup corresponds good to reality, the further decks can differ more. So we decided to convert deck 02, 03, 04 and 05 in detail and make a copy with adjusted stairs for each other floor. Floor 02 is used for the copy because it gets the closest to floor 01, 06 and 07 and is by that still a good approximation.
Floor 08 until 14 differ more from floor 02 but as long as they are far away from floor 04 it will not have a big influence. These last floors are also smaller than floor 02 and therefore the amount of floors is reduced to only 11 with approximately the same area as the original 14 floors.

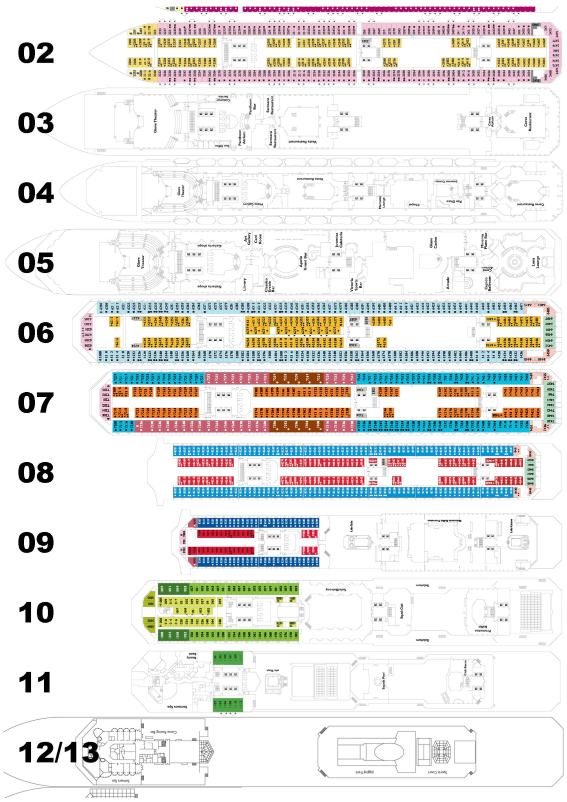

Figure 9: Originaldecks before conversion

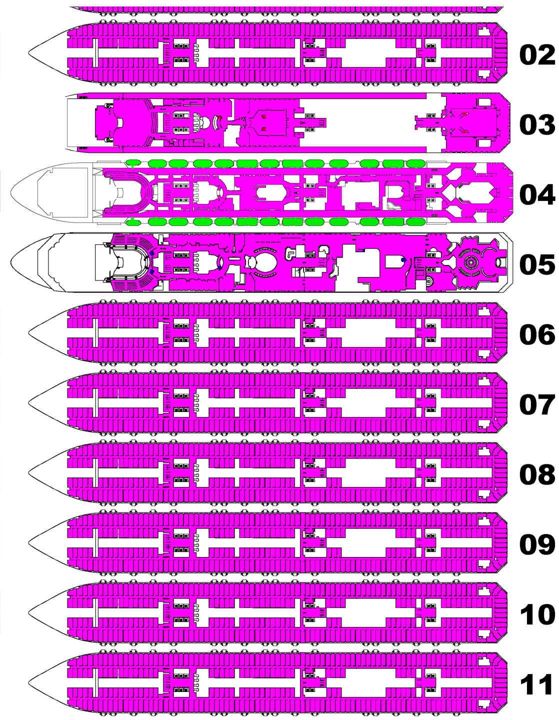

Figure 10: Deck approximated and converted

6 Results and Discussion

6.1 Simulation issues

Some agents always stuck in the walls near the stairways. Those agents did not reach the exits. This is why we could not get a t_{100} time but just saved about 98% of all passengers. The frequency of this issue was especially high in small areas and with a high total passenger amount. However, there was a trade off between realistic clogging in stairways and good ratio of rescued people out of all passengers. We decided not to decrease the number of passengers or expand the stairs because we did not want to eliminate the realistic clogging behavior which is important for the simulation.

6.2 Simulation Results

6.2.1 Standard ship

As listed above we were interested in the time in which a certain percentage of all agents was evacuated:

percentage of agents	10%	50%	90%	95%
evacuation time run1	16s	69s	167s	214s
evacuation time run2	16s	69s	164s	238s
averaged evacuation time	16s	69s	166s	226s

Table 1: Standard simulation: Needed time to evacuate a certain percentage of all agents.

Further our standard ship simulation showed the following performance:

6.2.2 Modified room disposition

As we expected the standard simulation revealed that hold-up problems occur because of the staircases. As the flow everywhere else was quite dynamic we abstained from adjusting the room disposition but instead we inserted an additional staircase. This simulation yielded the following results:

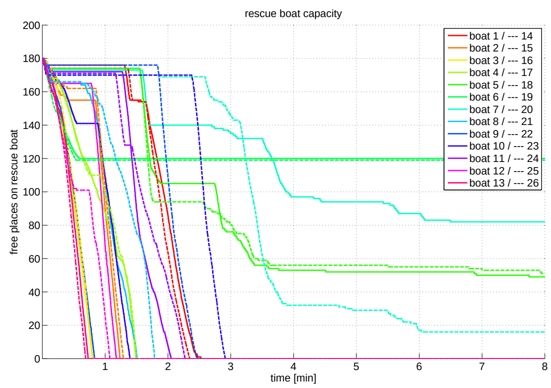

Figure 11: Standard simulation: Boat capacities during first simulation

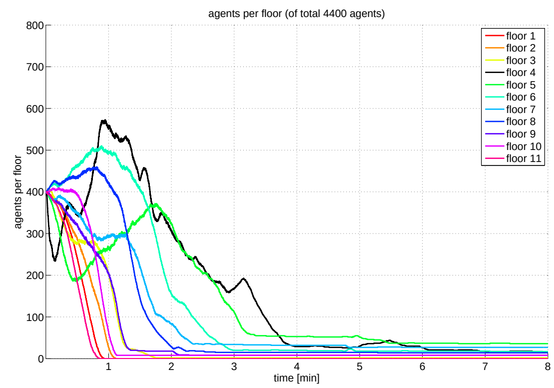

Figure 12: Standard simulation: Number of agents per floor in first simulation

percentage of agents	10%	50%	90%	95%
evacuation time	16s	67s	145s	182s

Table 2: Added stairs simulation: Needed time to evacuate a certain percentage of all agents.

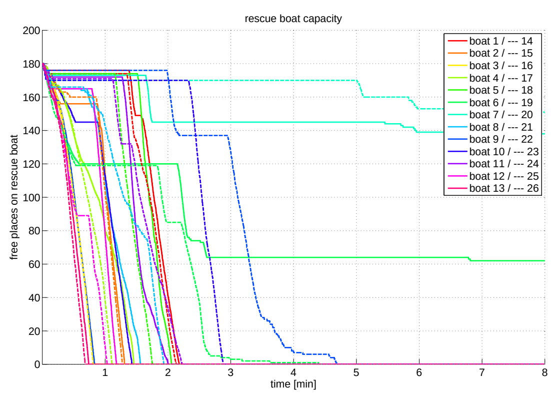

Figure 13: Added stairs simulation: Boat capacities during simulation

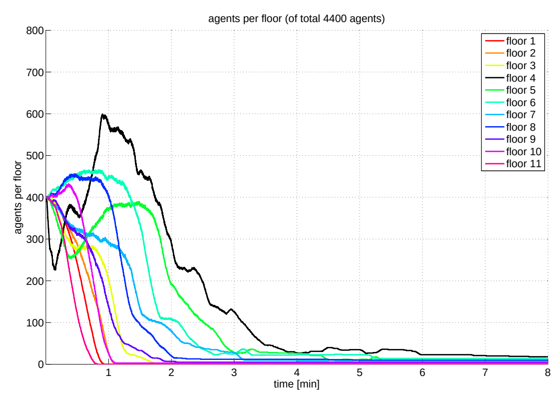

Figure 14: Added stairs simulation: Number of agents per floor

6.2.3 Modified rescueboat size

As we could see in the simulations with the standard ship, the rescue boats which are nearest to the stairs are filled first. This behavior is very intuitive. As soon as the nearest lifeboats are full, the agents continue to fill the other lifeboats. The greatest extension of the evacuation time occurs as follows: Towards the end of the simulation, not all lifeboats are still open. Therefore leftover agents, which walked in the direction of a lifeboat that closed in the meantime, have to cross a big distance to reach a lifeboat with free space.
We tried to avoid this delay by increasing the capacity of lifeboats near the stairs and removed some, which are the farthest away for the agents left over towards the end of the evacuation.

percentage of agents	10%	50%	90%	95%
evacuation time	17s	70s	154s	203s

Table 3: Varied boatsize simulation: Needed time to evacuate a certain percentage of all agents.

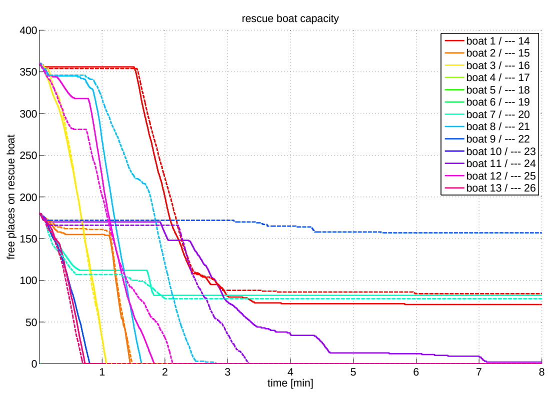

Figure 15: Varied boatsize simulation: Boat capacities during simulation

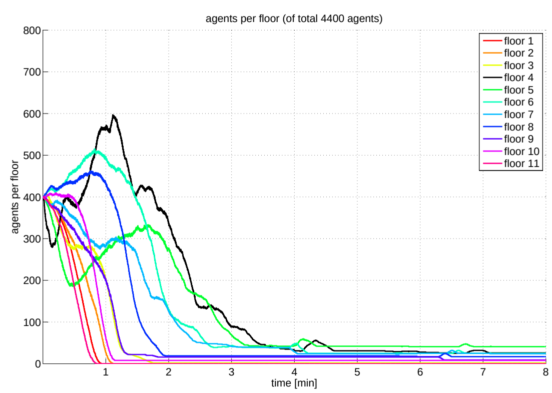

Figure 16: Varied boatsize simulation: Number of agents per floor

6.2.4 Crew command

As already mentioned in the implementation part, we changed our code in order to simulate staff controlling the rescue situation. The results of that extension can be seen below.

percentage of agents	10%	50%	90%	95%
evacuation time	20s	78s	162s	219s

Table 4: Crew command implementation: Needed time to evacuate a certain percentage of all agents.

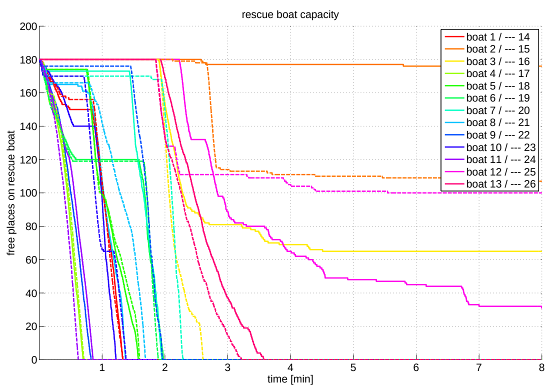

Figure 17: Crew command simulation: Boat capacities during simulation

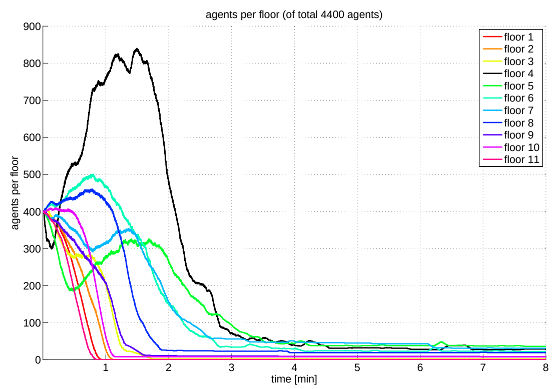

Figure 18: Crew command simulation: Number of agents per floor

6.3 Comparison

6.3.1 Standard - Modified room disposition

The Analysis of the simulation results showed that there is a huge potential in saving evacuation time. By adding just one additional staircase we were able to reduce the overall evacuation time by 42 seconds. Further 50% of all agents entered the exits approximately 2 seconds earlier compared to the standard model. In contrast to that the rescueboat capacity utilization remains basically the same.

6.3.2 Standard - Modified rescueboat size

As the results above show, there is no significant acceleration in the first part of the evacuation achieved by changing the distribution of the lifeboats . But, there is clearly a difference in the second half of the evacuation. The last 5% of agents are evacuated 23 seconds faster what is a huge speed up. This decrease in evacuation time is an effect of the reduced time the last agents need to leave the exit-deck.

6.3.3 Standard - Crew command

The results of the crew command extension are rather disappointing. On the one hand it was clear, that the first part of the simulation would become slower because the agents do not use the nearest boat. But on the other hand it was mentioned that the last part of the simulation has a significant speed up what is not the case. By looking at the video output the reason can be seen: The opening of the rescue boats near the stairs is useless at the point when only a few passengers are left because then they are already on the exit floor and walked towards the middle of the ship.
However, the idea has not to be thrown away. An implementation with a continuous opening of the next outer boat when the midmost ones are filled would eliminate our problem and a faster total evacuation time could be reached.

6.4 General discussion

During our project we revealed a lot of adjustments which could make cruise ships safer. Definitely we are aware of the simplicity of our model. Nevertheless we achieved to decrease the evacuation time with our improvements as shown above. Further we were able to attest that the staircases are the bottlenecks and it is possible to save time by adapting the ship geometry. Due to lack of time and simulation problems we unfortunately did not manage to find the optimal rescueboat size. However we proved that there is a capability to save further time by varying its size and position.

7 Outlook

During our work, we found a lot of possibilities to improve the model. The target of an ongoing project could be to make the model more realistic. There are a lot of possibilities to achieve this goal.
Some suggestions:

- To take into account that not all people automatically know where exactly the nearest exit is, the initialization of the escape routes should be modified.
- In our model the agents are evenly distributed over the floors and decks at the beginning of the simulation. This is however a very unrealistic scenario. In reality the agents will be unevenly spread. The evacuation time would depend very much on the form of this distribution.

- Different scenarios like night, dinner-time etc. that would change the distribution of the agents significantly could be compared.
- In reality there are a lot of effects that could occur like fire, tilt of the ship, flooded areas, power failure, mass panic etc.

There are a lot of other interesting effects that could be looked at as well.
Some ideas:

- In the ideal case, the evacuations on a ship are planned well. A good idea would be to define specific control points at which agents gather first. From those points the agents would be led in small groups to the rescue boats by a crew member. It would be interesting to analyze the effect of such a efficient control.
- The optimal combination of the different modifications we analyzed in this project could be found and therefore a minimal evacuation time.

For sure further interesting outcomes could be made by simply merging our results, means for example varying rescueboat size and geometries.

8 References

References

[1] Helbling, Dirk et al (2000): Simulating dynamical features of escape panic.

[2] Hardmeier,Jenal,Kueng,Thaler (2012): Modelling Situations of Evacuation in a Multi-level Building.

[3] SOLAS (1974): International Convention for the Safety of Life at Sea. http://www.imo.org/about/conventions/listofconventions/pages/international-convention-for-the-safety-of-life-at-sea-(solas),-1974.aspx

[4] SHIP EVACUATION: http://www.shipevacuation.com/

[5] CRUISE DECK PLANS PUBLIC SITE: http://www.cruisedeckplans.com/DP/Main/decks.php?ship=CostaSerena

[6] University of Greenwich (2011): maritimeEXODUS. http://fseg.gre.ac.uk/fire/marine_evac_model.html

[7] Haverie of Costa Concordia (2012): http://de.wikipedia.org/wiki/Costa_Concordia#Havarie_2012

9 Appendix

9.1 Code

9.1.1 *standard* code

```
function data = addAgentRepulsiveForce(data)
%ADDAGENTREPULSIVEFORCE Summary of this function goes here
%   Detailed explanation goes here

% Obstruction effects in case of physical interaction

% get maximum agent distance for which we calculate force
r_max = data.r_influence;
tree = 0;

for fi = 1:data.floor_count
    pos = [arrayfun(@(a) a.p(1), data.floor(fi).agents);
           arrayfun(@(a) a.p(2), data.floor(fi).agents)];

    % update range tree of lower floor
    tree_lower = tree;

    agents_on_floor = length(data.floor(fi).agents);

    % init range tree of current floor
    if agents_on_floor > 0
        tree = createRangeTree(pos);
    end

    for ai = 1:agents_on_floor
        pi = data.floor(fi).agents(ai).p;
        vi = data.floor(fi).agents(ai).v;
        ri = data.floor(fi).agents(ai).r;

        % use range tree to get the indices of all agents near agent ai
        idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
                                       pi(2) - r_max, pi(2) + r_max)';

        % loop over agents near agent ai
        for aj = idx

            % if force has not been calculated yet...
            if aj > ai
                pj = data.floor(fi).agents(aj).p;
                vj = data.floor(fi).agents(aj).v;
                rj = data.floor(fi).agents(aj).r;

                % vector pointing from j to i
                nij = (pi - pj) * data.meter_per_pixel;
```

```
                % distance of agents
                d = norm(nij);

                % normalized vector pointing from j to i
                nij = nij / d;
                % tangential direction
                tij = [-nij(2), nij(1)];

                % sum of radii
                rij = (ri + rj);

                % repulsive interaction forces
                if d < rij
                   T1 = data.k*(rij - d);
                   T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
                else
                   T1 = 0;
                   T2 = 0;
                end

                F =  (data.A * exp((rij - d)/data.B) + T1)*nij + T2;

                data.floor(fi).agents(ai).f = ...
                    data.floor(fi).agents(ai).f + F;
                data.floor(fi).agents(aj).f = ...
                    data.floor(fi).agents(aj).f - F;
            end
        end

        % include agents on stairs!
        if fi > 1
            % use range tree to get the indices of all agents near agent ai
            if ~isempty(data.floor(fi-1).agents)
                idx = rangeQuery(tree_lower, pi(1) - r_max, ...
                        pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';

                % if there are any agents...
                if ~isempty(idx)
                    for aj = idx
                        pj = data.floor(fi-1).agents(aj).p;
                        if data.floor(fi-1).img_stairs_up(round(pj(1)),
                            round(pj(2)))

                            vj = data.floor(fi-1).agents(aj).v;
                            rj = data.floor(fi-1).agents(aj).r;

                            % vector pointing from j to i
                            nij = (pi - pj) * data.meter_per_pixel;

```

```
                                % distance of agents
                                d = norm(nij);

                                % normalized vector pointing from j to i
                                nij = nij / d;
                                % tangential direction
                                tij = [-nij(2), nij(1)];

                                % sum of radii
                                rij = (ri + rj);

                                % repulsive interaction forces
                                if d < rij
                                   T1 = data.k*(rij - d);
                                   T2 = data.kappa*(rij - d)*dot((vj -
                                       vi),tij)*tij;
                                else
                                   T1 = 0;
                                   T2 = 0;
                                end

                                F = (data.A * exp((rij - d)/data.B) + T1)*nij
                                    + T2;

                                data.floor(fi).agents(ai).f = ...
                                     data.floor(fi).agents(ai).f + F;
                                data.floor(fi-1).agents(aj).f = ...
                                     data.floor(fi-1).agents(aj).f - F;
                            end
                        end
                    end
                end
            end
    end
end
```

Listing 1: addAgentRepulsiveForce.m

```
function data = addDesiredForce(data)
%ADDDESIREDFORCE add 'desired' force contribution (towards nearest exit or
%staircase)

for fi = 1:data.floor_count

    for ai=1:length(data.floor(fi).agents)

        % get agent's data
        p = data.floor(fi).agents(ai).p;
        m = data.floor(fi).agents(ai).m;
        v0 = data.floor(fi).agents(ai).v0;
```

```
        v = data.floor(fi).agents(ai).v;


        % get direction towards nearest exit
        ex = lerp2(data.floor(fi).img_dir_x, p(1), p(2));
        ey = lerp2(data.floor(fi).img_dir_y, p(1), p(2));
        e = [ex ey];

        % get force
        Fi = m * (v0*e - v)/data.tau;

        % add force
        data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
    end
end
```

Listing 2: addDesiredForce.m

```
function data = addWallForce(data)
%ADDWALLFORCE adds wall's force contribution to each agent

for fi = 1:data.floor_count

    for ai=1:length(data.floor(fi).agents)
        % get agents data
        p = data.floor(fi).agents(ai).p;
        ri = data.floor(fi).agents(ai).r;
        vi = data.floor(fi).agents(ai).v;

        % get direction from nearest wall to agent
        nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
        ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));

        % get distance to nearest wall
        diW = lerp2(data.floor(fi).img_wall_dist, p(1), p(2));

        % get perpendicular and tangential unit vectors
        niW = [ nx ny];
        tiW = [-ny nx];


        % calculate force
        if diW < ri
            T1 = data.k * (ri - diW);
            T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
        else
            T1 = 0;
            T2 = 0;
        end
        Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;
```

```
        % add force to agent's current force
        data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
    end
end
```

Listing 3: addWallForce.m

```
function data = applyForcesAndMove(data)
%APPLYFORCESANDMOVE apply current forces to agents and move them using
%the timestep and current velocity

n_velocity_clamps = 0;

% loop over all floors higher than exit floor
for fi = data.floor_exit:data.floor_count

    % init logical arrays to indicate agents that change the floor or exit
    % the simulation
    floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);

    % loop over all agents
    for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
        v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f  / data.floor(fi).agents(ai).m;

        % clamp velocity
        if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
            n_velocity_clamps = n_velocity_clamps + 1;
        end

        % get agent's new position
        newp = data.floor(fi).agents(ai).p + ...
               v * data.dt / data.meter_per_pixel;

        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
        if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
                data.floor(fi).agents(ai).r

            % get agent's position
            p = data.floor(fi).agents(ai).p;

            % get wall distance gradient (which is off course perpendicular
            % to the nearest wall)
            nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
            ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
```

```
            n = [nx ny];

            % project out perpendicular component of velocity vector
            v = v - dot(n,v)/dot(n,n)*n;

            % get agent's new position
            newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
        end

        % check if agents position is ok
        % repositioning after 50 times clogging
        % deleting if agent has a NaN position
        if ~isnan(newp)
            if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
               newp = data.floor(fi).agents(ai).p;
               v = [0 0];
               data.floor(fi).agents(ai).clogged =
                   data.floor(fi).agents(ai).clogged + 1;
               fprintf('WARNING: clogging agent %i on floor %i (%i).
                   Position
                   (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
               if data.floor(fi).agents(ai).clogged >= 40
                    nx = rand(1)*2 - 1;
                    ny = rand(1)*2 - 1;
                    n = [nx ny];
                    v = n*data.v_max/2;
                    fprintf('WARNING: agent %i on floor %i velocity set
                        random to get out of wall. Position
                        (%f,%f).\n',ai,fi,newp(1),newp(2))

                   % get agent's new position
                   newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
                   if isnan(newp)
                        % get rid of disturbing agent
                        fprintf('WARNING: position of an agent is NaN!
                            Deleted this agent.\n')
                        exited(ai) = 1;
                        data.agents_exited = data.agents_exited +1;
                        data.output.deleted_agents=data.output.deleted_agents+1;
                        newp = [1 1];
                   end
               end
            end
        else
            % get rid of disturbing agent
            fprintf('WARNING: position of an agent is NaN! Deleted this
                agent.\n')
            exited(ai) = 1;
```

```
            data.agents_exited = data.agents_exited +1;
            data.output.deleted_agents=data.output.deleted_agents+1;
            newp = [1 1];
        end


        % update agent's velocity and position
        data.floor(fi).agents(ai).v = v;
        data.floor(fi).agents(ai).p = newp;

        % reset forces for next timestep
        data.floor(fi).agents(ai).f = [0 0];

        % check if agent reached a staircase down and indicate floor change
        if data.floor(fi).img_stairs_down(round(newp(1)), round(newp(2)))
            floorchange(ai) = 1;
        end

        % check if agent reached an exit
        if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;

%              fprintf('agent exited from upper loop\n');

            %save current exit nr
            data.current_exit = data.exit_nr(round(newp(1)),
                round(newp(2)));

%              fprintf(int2str(data.current_exit));

            %update exit_left
            data.exit_left(1,data.current_exit) =
                data.exit_left(1,data.exit_nr(round(newp(1)),
                round(newp(2)))) - 1;

            %close exit if there is no more free space
            if data.exit_left(1,data.current_exit) < 1

                %change current exit to wall
                data.floor(data.floor_exit).img_wall =
                    data.floor(data.floor_exit).img_wall == 1 ...
                                | (data.exit_nr == (data.current_exit));
                data.floor(data.floor_exit).img_exit =
                    data.floor(data.floor_exit).img_exit == 1 ...
                                & (data.exit_nr ~= (data.current_exit));

                %redo initEscapeRoutes and initWallForces with new exit
                    and wall parameters
                data = initEscapeRoutes(data);
```

```
                data = initWallForces(data);

%                 fprintf('new routes from upper loop\n');

            end
        end
    end

    % add appropriate agents to next lower floor
    if fi > data.floor_exit
        data.floor(fi-1).agents = [data.floor(fi-1).agents
            data.floor(fi).agents(floorchange)];
    end

    % delete these and exited agents
    data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
end





% loop over all floors lower than exit floor
for fi = 1:data.floor_exit

    % init logical arrays to indicate agents that change the floor or exit
    % the simulation
    floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);

    % loop over all agents
    for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
        v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f  / data.floor(fi).agents(ai).m;

        % clamp velocity
        if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
            n_velocity_clamps = n_velocity_clamps + 1;
        end

        % get agent's new position
        newp = data.floor(fi).agents(ai).p + ...
               v * data.dt / data.meter_per_pixel;

        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
        if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
                data.floor(fi).agents(ai).r
```

```
            % get agent's position
            p = data.floor(fi).agents(ai).p;

            % get wall distance gradient (which is of course perpendicular
            % to the nearest wall)
            nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
            ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
            n = [nx ny];

            % project out perpendicular component of velocity vector
            v = v - dot(n,v)/dot(n,n)*n;

            % get agent's new position
            newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
        end


        % check if agents position is ok
        % repositioning after 50 times clogging
        % deleting if agent has a NaN position
        if ~isnan(newp)
            if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
                newp = data.floor(fi).agents(ai).p;
                v = [0 0];
                data.floor(fi).agents(ai).clogged = data.floor(fi).agents(ai).clogged + 1;
                fprintf('WARNING: clogging agent %i on floor %i (%i). Position (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
                if data.floor(fi).agents(ai).clogged >= 40
                     nx = rand(1)*2 - 1;
                     ny = rand(1)*2 - 1;
                     n = [nx ny];
                     v = n*data.v_max/2;
                     fprintf('WARNING: agent %i on floor %i velocity set random to get out of wall. Position (%f,%f).\n',ai,fi,newp(1),newp(2))

                    % get agent's new position
                    newp = data.floor(fi).agents(ai).p + ...
                    v * data.dt / data.meter_per_pixel;
                    if isnan(newp)
                        % get rid of disturbing agent
                        fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
                        exited(ai) = 1;
                        data.agents_exited = data.agents_exited +1;
                        data.output.deleted_agents=data.output.deleted_agents+1;
```

```
                        newp = [1 1];
                    end
                end
            end
        else
            % get rid of disturbing agent
            fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;
            data.output.deleted_agents=data.output.deleted_agents+1;
            newp = [1 1];
        end

        % update agent's velocity and position
        data.floor(fi).agents(ai).v = v;
        data.floor(fi).agents(ai).p = newp;

        % reset forces for next timestep
        data.floor(fi).agents(ai).f = [0 0];

        % check if agent reached a staircase up and indicate floor change
        if data.floor(fi).img_stairs_up(round(newp(1)), round(newp(2)))
            floorchange(ai) = 1;
        end

        % check if agent reached an exit
        if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;

%             fprintf('agent exited from lower loop\n');

            %save current exit nr
            data.current_exit = data.exit_nr(round(newp(1)), round(newp(2)));

            %update exit_left
            data.exit_left(1,data.current_exit) = data.exit_left(1,data.exit_nr(round(newp(1)), round(newp(2)))) - 1;

                        %close exit if there is no more free space
            if data.exit_left(1,data.current_exit) < 1

                %change current exit to wall
                data.floor(data.floor_exit).img_wall = data.floor(data.floor_exit).img_wall == 1 ...
                                | (data.exit_nr == (data.current_exit));
```

```
                data.floor(data.floor_exit).img_exit =
                    data.floor(data.floor_exit).img_exit == 1 ...
                                & (data.exit_nr ~= (data.current_exit));

                %redo initEscapeRoutes and initWallForces with new exit
                    and wall parameters
                data = initEscapeRoutes(data);
                data = initWallForces(data);

%                 fprintf('new routes from lower loop\n');

            end

        end
    end

    % add appropriate agents to next lower floor
    if fi < data.floor_exit
        data.floor(fi+1).agents = [data.floor(fi+1).agents ...
                                   data.floor(fi).agents(floorchange)];
    end

    % delete these and exited agents
    data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
end

% if n_velocity_clamps > 0
%     fprintf(['WARNING: clamped velocity of %d agents, ' ...
%             'possible simulation instability.\n'], n_velocity_clamps);
% end
```

Listing 4: applyForcesAndMove.m

```
function val = checkForIntersection(data, floor_idx, agent_idx)
% check an agent for an intersection with another agent or a wall
% the check is kept as simple as possible
%
%  arguments:
%   data            global data structure
%   floor_idx       which floor to check
%   agent_idx       which agent on that floor
%   agent_new_pos   vector: [x,y], desired agent position to check
%
%  return:
%   0               for no intersection
%   1               has an intersection with wall
%   2                                    with another agent

val = 0;
```

```
p = data.floor(floor_idx).agents(agent_idx).p;
r = data.floor(floor_idx).agents(agent_idx).r;

% check for agent intersection
for i=1:length(data.floor(floor_idx).agents)
    if i~=agent_idx
        if norm(data.floor(floor_idx).agents(i).p-p)*data.meter_per_pixel
            ...
                <= r +  data.floor(floor_idx).agents(i).r
            val=2;
            return;
        end
    end
end


% check for wall intersection
if lerp2(data.floor(floor_idx).img_wall_dist, p(1), p(2)) < r
    val = 1;
end
```

Listing 5: checkForIntersection.m

```
mex 'fastSweeping.c'
mex 'getNormalizedGradient.c'
mex 'lerp2.c'
mex 'createRangeTree.c'
mex 'rangeQuery.c'
```

Listing 6: compileC.m

```
function data = initAgents(data)

% place agents randomly in desired spots, without overlapping



function radius = getAgentRadius()
    %radius of an agent in meters
    radius = data.r_min + (data.r_max-data.r_min)*rand();
end

data.agents_exited = 0; %how many agents have reached the exit
data.total_agent_count = 0;

floors_with_agents = 0;
agent_count = data.agents_per_floor;
for i=1:data.floor_count
    data.floor(i).agents = [];
    [y,x] = find(data.floor(i).img_spawn);
```

```
    if ~isempty(x)
        floors_with_agents = floors_with_agents + 1;
        for j=1:agent_count
            cur_agent = length(data.floor(i).agents) + 1;

            % init agent
            data.floor(i).agents(cur_agent).r = getAgentRadius();
            data.floor(i).agents(cur_agent).v = [0, 0];
            data.floor(i).agents(cur_agent).f = [0, 0];
            data.floor(i).agents(cur_agent).m = data.m;
            data.floor(i).agents(cur_agent).v0 = data.v0;
            data.floor(i).agents(cur_agent).clogged = 0; %to check if
                agent is hanging in the wall

            tries = 10;
            while tries > 0
                % randomly pick a spot and check if it's free
                idx = randi(length(x));
                data.floor(i).agents(cur_agent).p = [y(idx), x(idx)];
                if checkForIntersection(data, i, cur_agent) == 0
                    tries = -1; % leave the loop
                end
                tries = tries - 1;
            end
            if tries > -1
                %remove the last agent
                data.floor(i).agents = data.floor(i).agents(1:end-1);
            end
        end
        data.total_agent_count = data.total_agent_count +
            length(data.floor(i).agents);

        if length(data.floor(i).agents) ~= agent_count
            fprintf(['WARNING: could only place %d agents on floor %d ' ...
                'instead of the desired %d.\n'], ...
                length(data.floor(i).agents), i, agent_count);
        end
    end
end
if floors_with_agents==0
    error('no spots to place agents!');
end

end
```

Listing 7: initAgents.m

```
function data = initEscapeRoutes(data)
%INITESCAPEROUTES Summary of this function goes here
```

```
%   Detailed explanation goes here

for i=1:data.floor_count

    boundary_data = zeros(size(data.floor(i).img_wall));
    boundary_data(data.floor(i).img_wall) =  1;

if i<data.floor_exit
    boundary_data(data.floor(i).img_stairs_up) = -1;

elseif i>data.floor_exit
        boundary_data(data.floor(i).img_stairs_down) = -1;

    else
        boundary_data(data.floor(i).img_exit) = -1;

end
    exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    [data.floor(i).img_dir_x, data.floor(i).img_dir_y] = ...
        getNormalizedGradient(boundary_data, -exit_dist);
end
```

Listing 8: initEscapeRoutes.m

```
function data = initialize(config)
% initialize the internal data from the config data
%
%  arguments:
%   config      data structure from loadConfig()
%
%  return:
%   data        data structure: all internal data used for the main loop
%
%               all internal data is stored in pixels NOT in meters


data = config;

%for convenience
data.pixel_per_meter = 1/data.meter_per_pixel;

fprintf('Init escape routes...\n');
data = initEscapeRoutes(data);
fprintf('Init wall forces...\n');
data = initWallForces(data);
fprintf('Init agents...\n');
data = initAgents(data);

% maximum influence of agents on each other

```

```
data.r_influence = data.pixel_per_meter * ...
    fzero(@(r) data.A * exp((2*data.r_max-r)/data.B) - 1e-4, data.r_max);

fprintf('Init plots...\n');
%init the plots
%exit plot
data.figure_exit=figure;
hold on;
axis([0 data.duration 0 data.total_agent_count]);
title(sprintf('agents that reached the exit (total agents: %i)',
    data.total_agent_count));

% floors plot
data.figure_floors=figure;
% figure('units','normalized','outerposition',[0 0 1 1])
data.figure_floors_subplots_w = data.floor_count;
data.figure_floors_subplots_h = 4;
for i=1:config.floor_count
        data.floor(i).agents_on_floor_plot =
            subplot(data.figure_floors_subplots_h,
            data.figure_floors_subplots_w, 3*data.floor_count - i+1 +
            data.figure_floors_subplots_w);
        if i == config.floor_exit - 1
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w,
                [(2*config.floor_count+1):3*config.floor_count]);
        elseif i == config.floor_exit
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w,
                [(config.floor_count+1):2*config.floor_count]);
        elseif i == config.floor_exit + 1
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w, [1:config.floor_count]);
        end
end

% init output matrizes
data.output = struct;
data.output.config = config;
data.output.agents_per_floor =
    ones(data.floor_count,data.duration/data.dt).*(-1);
data.output.exit_left = zeros(data.exit_count,data.duration/data.dt);

% prepare output file name
data.output_file_name = ['output_' data.frame_basename];

% prepare video file name
```

```
data.video_file_name = ['video_' data.frame_basename '.avi'];

% set deleted_agents to zero
data.output.deleted_agents = 0;
```

Listing 9: initialize.m

```
function data = initWallForces(data)
%INITWALLFORCES init wall distance maps and gradient maps for each floor

for i=1:data.floor_count

    % init boundary data for fast sweeping method
    boundary_data = zeros(size(data.floor(i).img_wall));
    boundary_data(data.floor(i).img_wall) = -1;

    % get wall distance
    wall_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    data.floor(i).img_wall_dist = wall_dist;

    % get normalized wall distance gradient
    [data.floor(i).img_wall_dist_grad_x, ...
     data.floor(i).img_wall_dist_grad_y] = ...
     getNormalizedGradient(boundary_data, wall_dist-data.meter_per_pixel);
end
```

Listing 10: initWallForces.m

```
function config = loadConfig(config_file)
% load the configuration file
%
%  arguments:
%   config_file     string, which configuration file to load
%


% get the path from the config file -> to read the images
config_path = fileparts(config_file);
if strcmp(config_path, '') == 1
    config_path = '.';
end

fid = fopen(config_file);
input = textscan(fid, '%s=%s');
fclose(fid);

keynames = input{1};
values = input{2};

%convert numerical values from string to double
```

```
v = str2double(values);
idx = ~isnan(v);
values(idx) = num2cell(v(idx));

config = cell2struct(values, keynames);


% read the images
for i=1:config.floor_count

    %building structure
    file = config.(sprintf('floor_%d_build', i));
    file_name = [config_path '/' file];
    img_build = imread(file_name);

    % decode images
    config.floor(i).img_wall = (img_build(:, :, 1) ==   0 ...
                              & img_build(:, :, 2) ==   0 ...
                              & img_build(:, :, 3) ==   0);

    config.floor(i).img_spawn = (img_build(:, :, 1) == 255 ...
                               & img_build(:, :, 2) ==   0 ...
                               & img_build(:, :, 3) == 255);

%second possibility:
%pixel is exit if 1-->0, 3-->0, and if 2 is between 255 and 230 or if no
%red or blue

    config.floor(i).img_exit = (img_build(:, :, 1) ==   0 ...
                              & img_build(:, :, 2) ~=   0 ...
                              & img_build(:, :, 3) ==   0);


    config.floor(i).img_stairs_up = (img_build(:, :, 1) == 255 ...
                                   & img_build(:, :, 2) ==   0 ...
                                   & img_build(:, :, 3) ==   0);

    config.floor(i).img_stairs_down = (img_build(:, :, 1) ==   0 ...
                                     & img_build(:, :, 2) ==   0 ...
                                     & img_build(:, :, 3) == 255);


    if i == config.floor_exit

        %make the exit_nr matrix where the number of exit is indicated in
            each
        %pixel

        %make a zeroes matrix as big as img_exit
        config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));
```

```
        %make a zeros vector as long as floor_exit
        config.exit_left = zeros(1,config.exit_count);

        %loop over all exits
        for e=1:config.exit_count

            %build the exit_nr matrix
            config.exit_nr = config.exit_nr + e*( img_build(:, :, 1) == 0
                & img_build(:, :, 2) == (256-e) & img_build(:, :, 3) == 0 )
                ;

            %build the exit_left matrix
            config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));

        end
    end

    %init the plot image here, because this won't change
    config.floor(i).img_plot = 5*config.floor(i).img_wall ...
        + 4*config.floor(i).img_stairs_up ...
        + 3*config.floor(i).img_stairs_down ...
        + 2*config.floor(i).img_exit ...
        + 1*config.floor(i).img_spawn;
    config.color_map = [1 1 1; 0.9 0.9 0.9; 0 1 0; 0.4 0.4 1; 1 0.4 0.4; 0
       0 0];
end
```

Listing 11: loadConfig.m

```
function plotAgentsPerFloor(data, floor_idx)
%plot time vs agents on floor

h = subplot(data.floor(floor_idx).agents_on_floor_plot);

set(h, 'position',[0.05+(data.floor_count -
    floor_idx)/(data.figure_floors_subplots_w+0.2), ...
     0.05, 1/(data.figure_floors_subplots_w*1.2), 0.3-0.05 ]);

if floor_idx~=data.floor_count
    set(h,'ytick',[]) %hide y-axis label
end

axis([0 data.time+data.dt 0 data.agents_per_floor*2]);

%axis([0 data.duration 0 data.agents_per_floor*2]);

hold on;
plot(data.time, length(data.floor(floor_idx).agents), 'b-');
hold off;
```

```
title(sprintf('%i', floor_idx));
```

Listing 12: plotAgentsPerFloor.m

```
function plotExitedAgents(data)
%plot time vs exited agents

hold on;
plot(data.time, data.agents_exited, 'r-');
hold off;
```

Listing 13: plotExitedAgents.m

```
function plotFloor(data, floor_idx)

if floor_idx == data.floor_exit-1 || floor_idx == data.floor_exit ||
    floor_idx == data.floor_exit+1
     h=subplot(data.floor(floor_idx).building_plot);

set(h,
    'position',[0,0.35+0.65/3*(floor_idx-data.floor_exit+1),1,0.65/3-0.005]);

hold off;
% the building image
imagesc(data.floor(floor_idx).img_plot);
hold on;

%plot options
colormap(data.color_map);
axis equal;
axis manual; %do not change axis on window resize

set(h, 'Visible', 'off')
% title(sprintf('floor %i', floor_idx))

% plot agents
if ~isempty(data.floor(floor_idx).agents)
     ang = [linspace(0,2*pi, 10) nan]';
     rmul = [cos(ang) sin(ang)] * data.pixel_per_meter;
     draw = cell2mat(arrayfun(@(a) repmat(a.p,length(ang),1) + a.r*rmul, ...
            data.floor(floor_idx).agents, 'UniformOutput', false)');
     line(draw(:,2), draw(:,1), 'Color', 'r');
end

hold off;
end
end
```

Listing 14: plotFloor.m

```
% post processing of output.mat data from simulation
% to run, you need to load the output first:
% load('output_FILENAME');

% tabula rasa
clc

% read in data from output
agents_per_floor = output.agents_per_floor;
config = output.config;
exit_left = output.exit_left;
simulation_time_real = output.simulation_time;
dt = config.dt;
deleted_agents = output.deleted_agents;


% get users screen size
screen_size = get(0, 'ScreenSize');

% agents on boat
agents_on_boat = sum(agents_per_floor(:,1:1:length(agents_per_floor)));

% check if whole simulation was performed
steps=config.duration/dt-1;
for i=1:steps
    if agents_on_boat(i)<0
        steps=i-2;
        break
    end
end

simulation_time_sim = steps*dt;

% recalculate agents on boat
agents_on_boat = sum(agents_per_floor(:,1:1:steps));
agents_start = agents_on_boat(1);
agents_left = agents_start-agents_on_boat;

% find out t10, t50, t90, t100
t10=0;
for i=1:steps
    if agents_left(i)<agents_start/10
        t10=t10+dt;
    end
end
if t10~=0
    t10=t10+dt;
end
```

```
t50=0;
for i=1:steps
    if agents_left(i)<agents_start/2
        t50=t50+dt;
    end
end
if t50~=0
    t50=t50+dt;
end

t90=0;
for i=1:steps
    if agents_left(i)<agents_start*0.9
        t90=t90+dt;
    end
end
if t90~=0
    t90=t90+dt;
end

t95=0;
for i=1:steps
    if agents_left(i)<agents_start*0.95
        t95=t95+dt;
    end
end
if t95~=0
    t95=t95+dt;
end

t100=0;
if agents_left==agents_start
    for i=1:steps
        if agents_left(i)<agents_start
            t100=t100+dt;
        end
    end
end

% create time axis
if t100~=0
    time = [0:dt:t100];
else
    time = [0:dt:simulation_time_sim];
end
steps = length(time);

% recalculate agents on boat
agents_on_boat = sum(agents_per_floor(:,1:1:steps));
agents_start = agents_on_boat(1);
```

```
agents_left = agents_start-agents_on_boat;
agents_per_floor = agents_per_floor(:,1:1:steps);
exit_left = exit_left(:,1:1:steps);

% plot agents left over time
f1 = figure;
hold on
grid on
set(gca,'XTick',[1:1:8],'FontSize',16)
plot(time/60,agents_left/agents_start*100,'LineWidth', 2)
axis([0 8 0 100])
title(sprintf('rescued agents (of total %i agents)',agents_start));
xlabel('time [min]')
ylabel('rescued agents out of all agents [%]')

% plot agents_per_floor over time
f2 = figure;
hold on
grid on
set(gca,'XTick',[1:1:8],'FontSize',16)
list = cell(config.floor_count,1);
color = hsv(config.floor_count);
color(config.floor_exit,:) = [0 0 0];
for i=1:config.floor_count
    plot(time/60,agents_per_floor(i,:),'LineWidth', 2,'color',color(i,:))
    list{i} = [sprintf('floor %i',i)];
end
legend(list)

axis([0 8 0 800])
title(sprintf('agents per floor (of total %i agents)',agents_start));
xlabel('time [min]')
ylabel('agents per floor')

% plot free places in rescue boats over time
f3 = figure;
hold on
grid on
set(gca,'XTick',[1:1:8],'FontSize',16)
list = cell(config.exit_count/2,1);
color = hsv(config.exit_count/2);
for i=1:config.exit_count/2
    plot(time/60,exit_left(i,:),'LineWidth', 2,'color',color(i,:))
    list{i} = [sprintf('boat %i / --- %i',i,i+13)];
end
for i=config.exit_count/2+1:config.exit_count
    plot(time/60,exit_left(i,:),'--','LineWidth',
        2,'color',color(i-config.exit_count/2,:))
end
legend(list)
```

```
axis([0 8 0 200])
title('rescue boat capacity');
xlabel('time [min]')
ylabel('free places on rescue boat')

% scale plots up to screen size
set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );
set(f2, 'Position', [0 0 screen_size(3) screen_size(4) ] );
set(f3, 'Position', [0 0 screen_size(3) screen_size(4) ] );


% print out

fprintf('Timestep: %f s\n', dt)
fprintf('Steps simulated: %i\n', steps)
fprintf('Simulation time: %f min\n', simulation_time_sim/60)
fprintf('Agents on ship on start: %i\n', agents_start)
fprintf('Agents on ship on simulation end: %i\n', agents_on_boat(end))
fprintf('Agents deleted due to NaN-positions: %i\n', deleted_agents)

fprintf('t_10: %f\n', t10)
fprintf('t_50: %f\n', t50)
fprintf('t_90: %f\n', t90)
fprintf('t_95: %f\n', t95)
fprintf('t_100: %f\n', t100)
```

Listing 15: plotFloor.m

```
function simulate(config_file)
% run this to start the simulation

% start recording the matlab output window for debugging reasons
diary log

if nargin==0
    config_file='../data/config1.conf';
end

fprintf('Load config file...\n');
config = loadConfig(config_file);

data = initialize(config);

data.step = 1;
data.time = 0;
fprintf('Start simulation...\n');

% tic until simulation end
simstart = tic;
```

```
%make video while simulation
if data.save_frames==1
           vidObj=VideoWriter(data.video_file_name);
           open(vidObj);
        end

while (data.time < data.duration)
    % tic until timestep end
    tstart=tic;
    data = addDesiredForce(data);
    data = addWallForce(data);
    data = addAgentRepulsiveForce(data);
    data = applyForcesAndMove(data);

    % dump agents_per_floor to output
    for floor=1:data.floor_count
          data.output.agents_per_floor(floor,data.step) = length(data.floor(floor).agents);
    end

    % dump exit_left to output
    data.output.exit_left(:,data.step) = data.exit_left';

    if mod(data.step,data.save_step) == 0

        % do the plotting
        set(0,'CurrentFigure',data.figure_floors);
        for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
            plotFloor(data, floor);
        end

        if data.save_frames==1
%             print('-depsc2',sprintf('frames/%s_%04i.eps', ...
%                 data.frame_basename,data.step), data.figure_floors);

%           make video while simulate
            currFrame=getframe(data.figure_floors);
            writeVideo(vidObj,currFrame);

        end

        set(0,'CurrentFigure',data.figure_exit);
        plotExitedAgents(data);

        if data.agents_exited == data.total_agent_count
            fprintf('All agents are now saved (or are they?). Time: %.2f sec\n', data.time);
            fprintf('Total Agents: %i\n', data.total_agent_count);
```

```
            print('-depsc2',sprintf('frames/exited_agents_%s.eps', ...
                data.frame_basename), data.figure_floors);
            break;
        end

        % toc of timestep
        data.telapsed = toc(tstart);
        % toc of whole simulation
        data.output.simulation_time = toc(simstart);

        % save output
        output = data.output;
        save(data.output_file_name,'output')
        fprintf('Frame %i done (took %.3fs; %.3fs out of %.3gs
            simulated).\n', data.step, data.telapsed, data.time,
            data.duration);

    end

    % update step
    data.step = data.step+1;

    % update time
    if (data.time + data.dt > data.duration)
        data.dt = data.duration - data.time;
        data.time = data.duration;
    else
        data.time = data.time + data.dt;
    end

end

%make video while simulation
close(vidObj);

% toc of whole simulation
data.output.simulation_time = toc(simstart);

% save complete simulation
output = data.output;
save('output','output')
fprintf('Simulation done in %i seconds and saved data to output file.\n',
    data.output.simulation_time);

% save diary
diary
```

Listing 16: simulate.m

9.1.2 *C* code

```

#include <mex.h>
#include <string.h>

#include "tree_build.c"
#include "tree_query.c"
#include "tree_free.c"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
{
    point_t *points;
    tree_t *tree;
    int m, n;
    uchar *data;
    int *root_index;

    if (nlhs < 1)
        return;

    points = (point_t*) mxGetPr(prhs[0]);
    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);

    if (m != 2)
        mexErrMsgTxt("...");

    tree = build_tree(points, n);

    plhs[0] = mxCreateNumericMatrix(tree->first_free + sizeof(int), 1,
       mxUINT8_CLASS, mxREAL);
    data = (uchar*) mxGetPr(plhs[0]);

    root_index = (int*) data;
    *root_index = tree->root_index;
    memcpy(data + sizeof(int), tree->data, tree->first_free);

    free_tree(tree);
}
```

Listing 17: createRangeTree.c

```
#include "mex.h"

#include <math.h>

#if defined __GNUC__ && defined __FAST_MATH__ && !defined __STRICT_ANSI__
#define MIN(i, j) fmin(i, j)
```

```
#define MAX(i, j) fmax(i, j)
#define ABS(i)    fabs(i)
#else
#define MIN(i, j) ((i) < (j) ? (i) : (j))
#define MAX(i, j) ((i) > (j) ? (i) : (j))
#define ABS(i)    ((i) < 0.0 ? -(i) : (i))
#endif


#define SOLVE_AND_UPDATE    udiff = uxmin - uymin; \
                            if (ABS(udiff) >= 1.0) \
                            { \
                                up = MIN(uxmin, uymin) + 1.0; \
                            } \
                            else \
                            { \
                                up = (uxmin + uymin + sqrt(2.0 - udiff *
                                    udiff)) / 2.0; \
                                up = MIN(uij, up); \
                            } \
                            err_loc = MAX(ABS(uij - up), err_loc); \
                            u[ij] = up;

#define I_STEP(_uxmin, _uymin, _st) if (boundary[ij] == 0.0) \
                                    { \
                                        uij = un; \
                                        un = u[ij + _st]; \
                                        uxmin = _uxmin; \
                                        uymin = _uymin; \
                                        SOLVE_AND_UPDATE \
                                        ij += _st; \
                                    } \
                                    else \
                                    { \
                                        up = un; \
                                        un = u[ij + _st]; \
                                        ij += _st; \
                                    }




#define I_STEP_UP(_uxmin, _uymin)   I_STEP(_uxmin, _uymin, 1)
#define I_STEP_DOWN(_uxmin, _uymin) I_STEP(_uxmin, _uymin,-1)

#define UX_NEXT un
#define UX_PREV up
#define UX_BOTH MIN(UX_PREV, UX_NEXT)

#define UY_RIGHT u[ij + m]
```

```
#define UY_LEFT  u[ij - m]
#define UY_BOTH  MIN(UY_LEFT, UY_RIGHT)


static void iteration(double *u, double *boundary, int m, int n, double
    *err)
{
    int i, j, ij;
    int m2, n2;
    double up, un, uij, uxmin, uymin, udiff, err_loc;

    m2 = m - 2;
    n2 = n - 2;

    *err = 0.0;
    err_loc = 0.0;

    /* first sweep */
    /* i = 0, j = 0 */
    ij = 0;
    un = u[ij];
    I_STEP_UP(UX_NEXT, UY_RIGHT)

    /* i = 1->m2, j = 0 */
    for (i = 1; i <= m2; ++i)
        I_STEP_UP(UX_BOTH, UY_RIGHT)

    /* i = m-1, j = 0 */
    I_STEP_UP(UX_PREV, UY_RIGHT)

    /* i = 0->m-1, j = 1->n2 */
    for (j = 1; j <= n2; ++j)
    {
        I_STEP_UP(UX_NEXT, UY_BOTH)

        for (i = 1; i <= m2; ++i)
            I_STEP_UP(UX_BOTH, UY_BOTH)

        I_STEP_UP(UX_PREV, UY_BOTH)
    }

    /* i = 0, j = n-1 */
    I_STEP_UP(UX_NEXT, UY_LEFT)

    /* i = 1->m2, j = n-1 */
    for (i = 1; i <= m2; ++i)
        I_STEP_UP(UX_BOTH, UY_LEFT)

    /* i = m-1, j = n-1 */
```

```
    I_STEP_UP(UX_PREV, UY_LEFT)


    /* sweep 2 */
    /* i = 0, j = n-1 */
    ij = (n-1)*m;
    un = u[ij];
    I_STEP_UP(UX_NEXT, UY_LEFT)

    /* i = 1->m2, j = n-1 */
    for (i = 1; i <= m2; ++i)
        I_STEP_UP(UX_BOTH, UY_LEFT)

    /* i = m-1, j = n-1 */
    I_STEP_UP(UX_PREV, UY_LEFT)

    /* i = 0->m-1, j = n2->1 */
    for (j = n2; j >= 1; --j)
    {
        ij = j*m;
        un = u[ij];
        I_STEP_UP(UX_NEXT, UY_BOTH)

        for (i = 1; i <= m2; ++i)
            I_STEP_UP(UX_BOTH, UY_BOTH)

        I_STEP_UP(UX_PREV, UY_BOTH)
    }

    /* i = 0, j = 0 */
    ij = 0;
    un = u[ij];
    I_STEP_UP(UX_NEXT, UY_RIGHT)

    /* i = 1->m2, j = 0 */
    for (i = 1; i <= m2; ++i)
        I_STEP_UP(UX_BOTH, UY_RIGHT)

    /* i = m-1, j = 0 */
    I_STEP_UP(UX_PREV, UY_RIGHT)

    /* sweep 3 */
    /* i = m-1, j = n-1 */
    ij = m*n - 1;
    un = u[ij];
    I_STEP_DOWN(UX_NEXT, UY_LEFT)

    /* i = m2->1, j = n-1 */
    for (i = m2; i >= 1; --i)
        I_STEP_DOWN(UX_BOTH, UY_LEFT)
```

```
    /* i = 0, j = n-1 */
    I_STEP_DOWN(UX_PREV, UY_LEFT)

    /* i = m-1->0, j = n2->1 */
    for (j = n2; j >= 1; --j)
    {
        I_STEP_DOWN(UX_NEXT, UY_BOTH)

        for (i = m2; i >= 1; --i)
            I_STEP_DOWN(UX_BOTH, UY_BOTH)

        I_STEP_DOWN(UX_PREV, UY_BOTH)
    }

    /* i = m-1, j = 0 */
    I_STEP_DOWN(UX_NEXT, UY_RIGHT)

    /* i = m2->1, j = 0 */
    for (i = m2; i >= 1; --i)
        I_STEP_DOWN(UX_BOTH, UY_RIGHT)

    /* i = 0, j = 0 */
    I_STEP_DOWN(UX_PREV, UY_RIGHT)

    /* sweep 4 */
    /* i = m-1, j = 0 */
    ij = m - 1;
    un = u[ij];
    I_STEP_DOWN(UX_NEXT, UY_RIGHT)

    /* i = m2->1, j = 0 */
    for (i = m2; i >= 1; --i)
        I_STEP_DOWN(UX_BOTH, UY_RIGHT)

    /* i = 0, j = 0 */
    I_STEP_DOWN(UX_PREV, UY_RIGHT)

    /* i = m-1->0, j = 1->n2 */
    for (j = 1; j <= n2; ++j)
    {
        ij = m - 1 + j*m;
        un = u[ij];
        I_STEP_DOWN(UX_NEXT, UY_BOTH)

        for (i = m2; i >= 1; --i)
            I_STEP_DOWN(UX_BOTH, UY_BOTH)

        I_STEP_DOWN(UX_PREV, UY_BOTH)
    }
```

```
    /* i = m-1, j = n-1 */
    ij = m*n - 1;
    un = u[ij];
    I_STEP_DOWN(UX_NEXT, UY_LEFT)

    /* i = m2->1, j = n-1 */
    for (i = m2; i >= 1; --i)
        I_STEP_DOWN(UX_BOTH, UY_LEFT)

    /* i = 0, j = n-1 */
    I_STEP_DOWN(UX_PREV, UY_LEFT)

    *err = MAX(*err, err_loc);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *u, *boundary;
    double tol, err;
    int m, n, entries, max_iter, i;

    /* Check number of outputs */
    if (nlhs < 1)
        return;
    else if (nlhs > 1)
        mexErrMsgTxt("At most 1 output argument needed.");

    /* Get inputs */
    if (nrhs < 1)
        mexErrMsgTxt("At least 1 input argument needed.");
    else if (nrhs > 3)
        mexErrMsgTxt("At most 3 input arguments used.");



    /* Get boundary */
    if (!mxIsDouble(prhs[0]) || mxIsClass(prhs[0], "sparse"))
        mexErrMsgTxt("Boundary field needs to be a full double precision matrix.");

    boundary = mxGetPr(prhs[0]);
    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);
    entries = m * n;

    /* Get max iterations  */
    if (nrhs >= 2)
    {
```

```
        if (!mxIsDouble(prhs[1]) || mxGetM(prhs[1]) != 1 ||
            mxGetN(prhs[1]) != 1)
            mexErrMsgTxt("Maximum iteration needs to be positive
                integer.");
        max_iter = (int) *mxGetPr(prhs[1]);
        if (max_iter <= 0)
            mexErrMsgTxt("Maximum iteration needs to be positive
                integer.");
    }
    else
        max_iter = 20;

    /* Get tolerance */
    if (nrhs >= 3)
    {
        if (!mxIsDouble(prhs[2]) || mxGetM(prhs[2]) != 1 ||
            mxGetN(prhs[2]) != 1)
            mexErrMsgTxt("Tolerance needs to be a positive real number.");
        tol = *mxGetPr(prhs[2]);
        if (tol < 0)
            mexErrMsgTxt("Tolerance needs to be a positive real number.");
    }
    else
        tol = 1e-12;


    /* create and init output (distance) matrix */
    plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
    u = mxGetPr(plhs[0]);

    for (i = 0; i < entries; ++i)
        u[i] = boundary[i] < 0.0 ? 0.0 : 1.0e10;

    err = 0.0;
    i = 0;
    do
    {
        iteration(u, boundary, m, n, &err);
        ++i;
    } while (err > tol && i < max_iter);
}
```

Listing 18: fastSweeping.c

```
#include "mex.h"

#include <math.h>

#define INTERIOR(i, j)  (boundary[(i) + m*(j)] == 0)
```

```c
#define DIST(i, j)  dist[(i) + m*(j)]
#define XGRAD(i, j) xgrad[(i) + m*(j)]
#define YGRAD(i, j) ygrad[(i) + m*(j)]

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    double *xgrad, *ygrad, *boundary, *dist;
    double dxp, dxm, dyp, dym, xns, yns, nrm;
    int m, n, i, j, nn;

    /* Check number of outputs */
    if (nlhs < 2)
        mexErrMsgTxt("At least 2 output argument needed.");
    else if (nlhs > 2)
        mexErrMsgTxt("At most 2 output argument needed.");

    /* Get inputs */
    if (nrhs < 2)
        mexErrMsgTxt("At least 2 input argument needed.");
    else if (nrhs > 2)
        mexErrMsgTxt("At most 2 input argument used.");



    /* Get boundary */
    if (!mxIsDouble(prhs[0]) || mxIsClass(prhs[0], "sparse"))
        mexErrMsgTxt("Boundary field needs to be a full double precision matrix.");

    boundary = mxGetPr(prhs[0]);
    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);

    /* Get distance field */
    if (!mxIsDouble(prhs[1]) || mxIsClass(prhs[1], "sparse") || mxGetM(prhs[1]) != m || mxGetN(prhs[1]) != n)
        mexErrMsgTxt("Distance field needs to be a full double precision matrix with same dimension as the boundary.");

    dist = mxGetPr(prhs[1]);
    m = mxGetM(prhs[1]);
    n = mxGetN(prhs[1]);

    /* create and init output (gradient) matrices */
    plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
    plhs[1] = mxCreateDoubleMatrix(m, n, mxREAL);
    xgrad = mxGetPr(plhs[0]);
    ygrad = mxGetPr(plhs[1]);
```

```
    for (j = 0; j < n; ++j)
        for (i = 0; i < m; ++i)
            if (INTERIOR(i,j))
            {
                if (i > 0)
                    dxm = INTERIOR(i-1,j) ? DIST(i-1,j) : DIST(i,j);
                else
                    dxm = DIST(i,j);

                if (i < m-1)
                    dxp = INTERIOR(i+1,j) ? DIST(i+1,j) : DIST(i,j);
                else
                    dxp = DIST(i,j);

                if (j > 0)
                    dym = INTERIOR(i,j-1) ? DIST(i,j-1) : DIST(i,j);
                else
                    dym = DIST(i,j);

                if (j < n-1)
                    dyp = INTERIOR(i,j+1) ? DIST(i,j+1) : DIST(i,j);
                else
                    dyp = DIST(i,j);

                XGRAD(i, j) = (dxp - dxm) / 2.0;
                YGRAD(i, j) = (dyp - dym) / 2.0;
                nrm = sqrt(XGRAD(i, j)*XGRAD(i, j) + YGRAD(i, j)*YGRAD(i,
                    j));
                if (nrm > 1e-12)
                {
                    XGRAD(i, j) /= nrm;
                    YGRAD(i, j) /= nrm;
                }
            }
            else
            {
                XGRAD(i, j) = 0.0;
                YGRAD(i, j) = 0.0;
            }

    for (j = 0; j < n; ++j)
        for (i = 0; i < m; ++i)
            if (!INTERIOR(i, j))
            {
                xns = 0.0;
                yns = 0.0;
                nn = 0;
                if (i > 0 && INTERIOR(i-1,j))
```

```
                {
                    xns += XGRAD(i-1,j);
                    yns += YGRAD(i-1,j);
                    ++nn;
                }
                if (i < m-1 && INTERIOR(i+1,j))
                {
                    xns += XGRAD(i+1,j);
                    yns += YGRAD(i+1,j);
                    ++nn;
                }
                if (j > 0 && INTERIOR(i,j-1))
                {
                    xns += XGRAD(i,j-1);
                    yns += YGRAD(i,j-1);
                    ++nn;
                }
                if (j < n-1 && INTERIOR(i,j+1))
                {
                    xns += XGRAD(i,j+1);
                    yns += YGRAD(i,j+1);
                    ++nn;
                }

                if (nn > 0)
                {
                    XGRAD(i, j) = xns / nn;
                    YGRAD(i, j) = yns / nn;
                }
            }
}
```

Listing 19: getNormalizedGradient.c

```

#include <mex.h>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
{
    int m, n, i0, i1, j0, j1, idx00;
    double *data, *out, x, y, wx0, wy0, wx1, wy1;
    double d00, d01, d10, d11;

    if (nlhs < 1)
        return;
    else if (nlhs > 1)
        mexErrMsgTxt("Exactly one output argument needed.");

    if (nrhs != 3)
```

```
        mexErrMsgTxt("Exactly three input arguments needed.");

    m = mxGetM(prhs[0]);
    n = mxGetN(prhs[0]);
    data = mxGetPr(prhs[0]);
    x = *mxGetPr(prhs[1]) - 1;
    y = *mxGetPr(prhs[2]) - 1;

    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
    out = mxGetPr(plhs[0]);

    x = x < 0 ? 0 : x > m - 1 ? m - 1 : x;
    y = y < 0 ? 0 : y > n - 1 ? n - 1 : y;
    i0 = (int) x;
    j0 = (int) y;
    i1 = i0 + 1;
    i1 = i1 > m - 1 ? m - 1 : i1;
    j1 = j0 + 1;
    j1 = j1 > n - 1 ? n - 1 : j1;

    idx00 = i0 + m * j0;
    d00 = data[idx00];
    d01 = data[idx00 + m];
    d10 = data[idx00 + 1];
    d11 = data[idx00 + m + 1];

    wx1 = x - i0;
    wy1 = y - j0;
    wx0 = 1.0 - wx1;
    wy0 = 1.0 - wy1;

    *out = wx0 * (wy0 * d00 + wy1 * d01) + wx1 * (wy0 * d10 + wy1 * d11);
}
```

Listing 20: lerp2.c

```

#include <mex.h>
#include <string.h>

#include "tree_build.c"
#include "tree_query.c"
#include "tree_free.c"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
{
    tree_t *tree;
    int n, i;
    int *point_idx, *root_idx;
```

```c
    range_t *range;
    uchar *data;

    if (nlhs != 1)
        mexErrMsgTxt("...");

    if (nrhs < 5)
        mexErrMsgTxt("...");
    else if (nrhs > 5)
        mexErrMsgTxt("...");

    data = (uchar*) mxGetPr(prhs[0]);

    tree = (tree_t*) malloc(sizeof(tree_t));
    tree->first_free = mxGetM(prhs[0]) - sizeof(int);
    tree->total_size = tree->first_free;
    root_idx = (int*) data;
    tree->root_index = *root_idx;
    tree->data = data + sizeof(int);

    n = mxGetN(prhs[0]);
    if (n != 1)
        mexErrMsgTxt("...");

    range = range_query(tree, *mxGetPr(prhs[1]), *mxGetPr(prhs[2]),
        *mxGetPr(prhs[3]), *mxGetPr(prhs[4]));

    plhs[0] = mxCreateNumericMatrix(range->n, 1, mxUINT32_CLASS, mxREAL);
    point_idx = (int*) mxGetPr(plhs[0]);

    for (i = 0; i < range->n; ++i)
        point_idx[i] = range->point_idx[i] + 1;

    free_range(range);
    free(tree);
}
```

Listing 21: rangeQuery.c

```c
#ifndef TREE_H
#define TREE_H

#include "tree_types.h"

/* build a 2D range tree using the given points */
tree_t* build_tree(point_t *points, int n);

/* query a range tree */
range_t* range_query(tree_t *tree, double x_min, double x_max, double
    y_min, double y_max);
```

```
/* free memory of a tree */
void free_tree(tree_t *tree);

/* free memory of a range */
void free_range(range_t *range);

#endif
```

Listing 22: tree.h

```
#ifndef TREE_BUILD_H
#define TREE_BUILD_H

#include "tree.h"

/* recursively build a subtree */
int build_subtree(tree_t *tree, double *x_vals, const int nx, point_t
    *points, int *point_idx, const int np);

/* double comparison for qsort */
int compare_double(const void *a, const void *b);

/* index array sorting functions, sort point index array by point y
    coordinates */
void index_sort_y(const point_t *points, int *point_idx, const int n);
void index_quicksort_y(const point_t *points, int *point_idx, int l, int
    r);
int index_partition_y(const point_t *points, int *point_idx, int l, int r);

#endif
```

Listing 23: tree_build.h

```

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "tree_build.h"

tree_t* build_tree(point_t *points, int n)
{
    int nx, i, j, *point_idx;
    double *x_vals;
    tree_t *tree;

    /* get x coordinate values of all points */
    x_vals = (double*) malloc(n * sizeof(double));
```

```
    for (i = 0; i < n; ++i)
        x_vals[i] = points[i].x;

    /* sort x values */
    qsort(x_vals, n, sizeof(double), compare_double);

    /* count number of unique x values */
    nx = 1;
    for (i = 1; i < n; ++i)
        if (x_vals[i] != x_vals[i - 1])
            ++nx;

    /* remove duplicates */
    j = 0;
    for (i = 0; i < nx; ++i)
    {
        x_vals[i] = x_vals[j];
        while (x_vals[i] == x_vals[j])
            ++j;
    }

    /* create an index array */
    point_idx = (int*) malloc(n * sizeof(int));
    for (i = 0; i < n; ++i)
        point_idx[i] = i;

    /* sort index array by y coordinates of associated points */
    index_sort_y(points, point_idx, n);

    /* init tree */
    tree = (tree_t*) malloc(sizeof(tree_t));
    tree->total_size = n * sizeof(point_t);
    tree->data = (uchar*) malloc(tree->total_size);

    /* copy point coordinates to tree data */
    memcpy(tree->data, points, n * sizeof(point_t));

    /* set first free byte and root index of the tree */
    tree->first_free = n * sizeof(point_t);
    tree->root_index = tree->first_free;

    /* recursively build tree */
    build_subtree(tree, x_vals, nx, points, point_idx, n);

    /* free temporaries */
    free(x_vals);
    return tree;
}
```

```
int build_subtree(tree_t *tree, double *x_vals, const int nx, point_t
    *points, int *point_idx, const int np)
{
    int i, j, k, nx_left, np_left, node_size, right_idx;
    node_t *node;
    int *node_point_idx, *point_idx_left, *point_idx_right, node_idx;
    uchar *new_data;

    assert(nx > 0);
    assert(np > 0);

    /* allocate memory in the tree data structure */
    node_size = sizeof(node_t) + np * sizeof(int);
    while (tree->first_free + node_size > tree->total_size)
    {
        tree->total_size <<= 1;
        new_data = (uchar*) malloc(tree->total_size * sizeof(uchar));
        for (i = 0; i < tree->first_free; ++i)
            new_data[i] = tree->data[i];
        free(tree->data);
        tree->data = new_data;
    }
    node_idx = tree->first_free;
    node = (node_t*) &tree->data[node_idx];
    tree->first_free += node_size;

    /* set number of stored points */
    node->np = np;
    node_point_idx = (int*) (node + 1);

    /* copy point indices to node */
    memcpy(node_point_idx, point_idx, np * sizeof(int));

    /* create child node if there is only one x value left, otherwise
       create interior node */
    if (nx == 1)
    {
        node->right_idx = -1;
        node->x_val = x_vals[0];
    }
    else
    {
        /* get median of x values */
        nx_left = nx >> 1;
        node->x_val = x_vals[nx_left - 1];

        /* count points belonging to the left child */
        np_left = 0;
        for (i = 0; i < np; ++i)
        {
```

```
            if (points[point_idx[i]].x <= node->x_val)
                ++np_left;
        }

        /* allocate memory for children's index arrays */
        point_idx_left = (int*) malloc(np_left * sizeof(int));
        point_idx_right = (int*) malloc((np - np_left) * sizeof(int));

        /* fill index arrays */
        j = 0;
        k = 0;
        for (i = 0; i < np; ++i)
        {
            if (points[point_idx[i]].x <= node->x_val)
                point_idx_left[j++] = point_idx[i];
            else
                point_idx_right[k++] = point_idx[i];
        }

        /* free current node's temporary index array */
        free(point_idx);

        /* build left subtree */
        build_subtree(tree, x_vals, nx_left, points, point_idx_left,
            np_left);

        /* build right subtree and get its root node index */
        right_idx = build_subtree(tree, x_vals + nx_left, nx - nx_left,
            points, point_idx_right, np - np_left);
        /* update node pointer (could have changed during build_subtree,
            because of data allocation) */
        node = (node_t*) &tree->data[node_idx];
        /* update node's right child index */
        node->right_idx = right_idx;
    }

    /* return node index to parent */
    return node_idx;
}

int compare_double(const void *a, const void *b)
{
    double ad, bd;
    ad = *((double*) a);
    bd = *((double*) b);
    return (ad < bd) ? -1 : (ad > bd) ? 1 : 0;
}

void index_sort_y(const point_t *points, int *point_idx, const int n)
{
```

```
    index_quicksort_y(points, point_idx, 0, n - 1);
}

void index_quicksort_y(const point_t *points, int *point_idx, int l, int r)
{
    int p;

    /* quicksort point indices by point y coordinates, don't touch point
       array itself */
    while (l < r)
    {
        p = index_partition_y(points, point_idx, l, r);
        if (r - p > p - l)
        {
            index_quicksort_y(points, point_idx, l, p - 1);
            l = p + 1;
        }
        else
        {
            index_quicksort_y(points, point_idx, p + 1, r);
            r = p - 1;
        }
    }
}

int index_partition_y(const point_t *points, int *point_idx, int l, int r)
{
    int i, j, tmp;
    double pivot;

    /* rightmost element is pivot */
    i = l;
    j = r - 1;
    pivot = points[point_idx[r]].y;

    /* quicksort partition */
    do
    {
        while (points[point_idx[i]].y <= pivot && i < r)
            ++i;

        while (points[point_idx[j]].y >= pivot && j > l)
            --j;

        if (i < j)
        {
            tmp = point_idx[i];
            point_idx[i] = point_idx[j];
            point_idx[j] = tmp;
        }
```

```c
    } while (i < j);

    if (points[point_idx[i]].y > pivot)
    {
        tmp = point_idx[i];
        point_idx[i] = point_idx[r];
        point_idx[r] = tmp;
    }

    return i;
}
```

Listing 24: tree_build.c

```c

#include <stdlib.h>

#include "tree.h"

void free_tree(tree_t *tree)
{
    free(tree->data);
}

void free_range(range_t *range)
{
    free(range->point_idx);
}
```

Listing 25: tree_free.c

```c
#ifndef TREE_QUERY_H
#define TREE_QUERY_H

#include "tree_types.h"

/* appends a point-index to a range, icnreases range capacity if needed */
void range_append(range_t *range, int idx);

/* finds the split node of a given query */
int find_split_node(tree_t *tree, int node_idx, range_t *range);

/* query the points of a node by a given range by y-coordinate */
void range_query_y(tree_t *tree, int node_idx, range_t *range);

#endif
```

Listing 26: tree_query.h

```c
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

#include "tree_query.h"

#define LEFT_CHILD_IDX(node_idx, node)  (node_idx) + sizeof(node_t) +
    (node)->np * sizeof(int)
#define RIGHT_CHILD_IDX(node_idx, node) (node)->right_idx
#define NODE_FROM_IDX(tree, node_idx)   (node_t*) &(tree)->data[node_idx];

range_t* range_query(tree_t *tree, double x_min, double x_max, double
    y_min, double y_max)
{
    int split_node_idx, node_idx;
    node_t *split_node, *node;
    range_t *range;

    /* init range */
    range = (range_t*) malloc(sizeof(range_t));
    range->min.x = x_min;
    range->max.x = x_max;
    range->min.y = y_min;
    range->max.y = y_max;
    range->n = 0;
    range->total_size = 16;
    range->point_idx = (int*) malloc(range->total_size * sizeof(int));

    /* find split node */
    split_node_idx = find_split_node(tree, tree->root_index, range);
    split_node = NODE_FROM_IDX(tree, split_node_idx);

    /* if split node is a child */
    if (split_node->right_idx == -1)
    {
        range_query_y(tree, split_node_idx, range);
        return range;
    }

    /* follow left path of the split node */
    node_idx = LEFT_CHILD_IDX(split_node_idx, split_node);
    node = NODE_FROM_IDX(tree, node_idx);
    while (node->right_idx != -1)
    {
        if (range->min.x <= node->x_val)
        {
            range_query_y(tree, RIGHT_CHILD_IDX(node_idx, node), range);
            node_idx = LEFT_CHILD_IDX(node_idx, node);
```

```
        }
        else
            node_idx = RIGHT_CHILD_IDX(node_idx, node);
        node = NODE_FROM_IDX(tree, node_idx);
    }
    range_query_y(tree, node_idx, range);

    /* follow right path of the split node */
    node_idx = split_node->right_idx;
    node = NODE_FROM_IDX(tree, node_idx);
    while (node->right_idx != -1)
    {
        if (range->max.x > node->x_val)
        {
            range_query_y(tree, LEFT_CHILD_IDX(node_idx, node), range);
            node_idx = RIGHT_CHILD_IDX(node_idx, node);
        }
        else
            node_idx = LEFT_CHILD_IDX(node_idx, node);
        node = NODE_FROM_IDX(tree, node_idx);
    }
    range_query_y(tree, node_idx, range);

    return range;
}

void range_append(range_t *range, int idx)
{
    int *new_point_idx;
    int new_size, i;

    /* just append if there is enough place, otherwise double capacity and
       append */
    if (range->n < range->total_size)
        range->point_idx[range->n++] = idx;
    else
    {
        new_size = range->total_size << 1;
        new_point_idx = (int*) malloc(new_size * sizeof(int));
        for (i = 0; i < range->n; ++i)
            new_point_idx[i] = range->point_idx[i];
        new_point_idx[range->n++] = idx;
        free(range->point_idx);
        range->point_idx = new_point_idx;
        range->total_size = new_size;
    }
}

int find_split_node(tree_t *tree, int node_idx, range_t *range)
{
```

```
    node_t *node;

    node = (node_t*) &tree->data[node_idx];
    /* check if this node is the split node */
    if (range->min.x <= node->x_val && range->max.x > node->x_val)
        return node_idx;

    /* ...or if it is a child (and therefor the split node) */
    if (node->right_idx == -1)
        return node_idx;

    /* otherwise search the split node at the left or right of the current
       node */
    if (range->max.x <= node->x_val)
        return find_split_node(tree, LEFT_CHILD_IDX(node_idx, node),
            range);
    else
        return find_split_node(tree, RIGHT_CHILD_IDX(node_idx, node),
            range);
}

void range_query_y(tree_t *tree, int node_idx, range_t *range)
{
    point_t *points;
    double y;
    int i, j, k, m, start, end;
    int *point_idx;
    node_t *node;

    node = (node_t*) &tree->data[node_idx];
    points = (point_t*) tree->data;
    point_idx = (int*) (node + 1);

    /* return if all points are outside the range */
    if (points[point_idx[0]].y > range->max.y || points[point_idx[node->np
       - 1]].y < range->min.y)
        return;

    /* binary search for lower end of the range */
    y = range->min.y;
    j = 0;
    k = node->np - 1;
    while (j != k)
    {
        m = (j + k) / 2;
        if (points[point_idx[m]].y >= y)
            k = m;
        else
            j = m + 1;
    }
```

```c
    start = j;

    /* binary search for higher end of the range */
    y = range->max.y;
    j = 0;
    k = node->np - 1;
    while (j != k)
    {
        m = (j + k + 1) / 2;
        if (points[point_idx[m]].y > y)
            k = m - 1;
        else
            j = m;
    }
    end = j;

    /* append found points to the range */
    for (i = start; i <= end; ++i)
        if (points[point_idx[i]].x <= range->max.x)
            range_append(range, point_idx[i]);

}
```

Listing 27: tree_query.c

```c
#ifndef TREE_TYPES_H
#define TREE_TYPES_H

typedef unsigned char uchar;

/* 2D point */
typedef struct
{
    double x;
    double y;
} point_t;

/* tree */
typedef struct
{
    /* byte data array with points and nodes */
    uchar *data;

    /* index of first unused byte */
    int first_free;

    /* total number of allocated bytes */
    int total_size;

    /* index of the root node in the data array*/
```

```c
    int root_index;
} tree_t;

/* node */
typedef struct
{
    /* index of the right child node (left child follows directly after
        current) */
    int right_idx;

    /* number of associated points */
    int np;

    /* associated x-coordinate value */
    double x_val;
} node_t;

/* range */
typedef struct
{
    /* point index list */
    int *point_idx;

    /* number of saved indices */
    int n;

    /* total number of allocated indices */
    int total_size;

    /* minimum range point */
    point_t min;

    /* maximum range point */
    point_t max;
} range_t;

#endif
```

Listing 28: tree_types.h

9.1.3 *crew command* code

```matlab
function data = addAgentRepulsiveForce(data)
%ADDAGENTREPULSIVEFORCE Summary of this function goes here
%   Detailed explanation goes here

% Obstruction effects in case of physical interaction

% get maximum agent distance for which we calculate force
```

```
r_max = data.r_influence;
tree = 0;

for fi = 1:data.floor_count
    pos = [arrayfun(@(a) a.p(1), data.floor(fi).agents);
           arrayfun(@(a) a.p(2), data.floor(fi).agents)];

    % update range tree of lower floor
    tree_lower = tree;

    agents_on_floor = length(data.floor(fi).agents);

    % init range tree of current floor
    if agents_on_floor > 0
        tree = createRangeTree(pos);
    end

    for ai = 1:agents_on_floor
        pi = data.floor(fi).agents(ai).p;
        vi = data.floor(fi).agents(ai).v;
        ri = data.floor(fi).agents(ai).r;

        % use range tree to get the indices of all agents near agent ai
        idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
                                      pi(2) - r_max, pi(2) + r_max)';

        % loop over agents near agent ai
        for aj = idx

            % if force has not been calculated yet...
            if aj > ai
                pj = data.floor(fi).agents(aj).p;
                vj = data.floor(fi).agents(aj).v;
                rj = data.floor(fi).agents(aj).r;

                % vector pointing from j to i
                nij = (pi - pj) * data.meter_per_pixel;

                % distance of agents
                d = norm(nij);

                % normalized vector pointing from j to i
                nij = nij / d;
                % tangential direction
                tij = [-nij(2), nij(1)];

                % sum of radii
                rij = (ri + rj);

                % repulsive interaction forces
```

```
                if d < rij
                   T1 = data.k*(rij - d);
                   T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
                else
                   T1 = 0;
                   T2 = 0;
                end

                F =  (data.A * exp((rij - d)/data.B) + T1)*nij + T2;

                data.floor(fi).agents(ai).f = ...
                    data.floor(fi).agents(ai).f + F;
                data.floor(fi).agents(aj).f = ...
                    data.floor(fi).agents(aj).f - F;
            end
        end

        % include agents on stairs!
        if fi > 1
            % use range tree to get the indices of all agents near agent ai
            if ~isempty(data.floor(fi-1).agents)
                idx = rangeQuery(tree_lower, pi(1) - r_max, ...
                        pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';

                % if there are any agents...
                if ~isempty(idx)
                    for aj = idx
                        pj = data.floor(fi-1).agents(aj).p;
                        if data.floor(fi-1).img_stairs_up(round(pj(1)),
                            round(pj(2)))

                            vj = data.floor(fi-1).agents(aj).v;
                            rj = data.floor(fi-1).agents(aj).r;

                            % vector pointing from j to i
                            nij = (pi - pj) * data.meter_per_pixel;

                            % distance of agents
                            d = norm(nij);

                            % normalized vector pointing from j to i
                            nij = nij / d;
                            % tangential direction
                            tij = [-nij(2), nij(1)];

                            % sum of radii
                            rij = (ri + rj);

                            % repulsive interaction forces
                            if d < rij
```

```
                               T1 = data.k*(rij - d);
                               T2 = data.kappa*(rij - d)*dot((vj -
                                   vi),tij)*tij;
                            else
                               T1 = 0;
                               T2 = 0;
                            end

                            F = (data.A * exp((rij - d)/data.B) + T1)*nij
                                + T2;

                            data.floor(fi).agents(ai).f = ...
                                data.floor(fi).agents(ai).f + F;
                            data.floor(fi-1).agents(aj).f = ...
                                data.floor(fi-1).agents(aj).f - F;
                        end
                    end
                end
            end
        end
    end
end
```

Listing 29: addAgentRepulsiveForce.m

```
function data = addDesiredForce(data)
%ADDDESIREDFORCE add 'desired' force contribution (towards nearest exit or
%staircase)

for fi = 1:data.floor_count

    for ai=1:length(data.floor(fi).agents)

        % get agent's data
        p = data.floor(fi).agents(ai).p;
        m = data.floor(fi).agents(ai).m;
        v0 = data.floor(fi).agents(ai).v0;
        v = data.floor(fi).agents(ai).v;


        % get direction towards nearest exit
        ex = lerp2(data.floor(fi).img_dir_x, p(1), p(2));
        ey = lerp2(data.floor(fi).img_dir_y, p(1), p(2));
        e = [ex ey];

        % get force
        Fi = m * (v0*e - v)/data.tau;

        % add force
        data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
```

```
    end
end
```

Listing 30: addDesiredForce.m

```
function data = addWallForce(data)
%ADDWALLFORCE adds wall's force contribution to each agent

for fi = 1:data.floor_count

    for ai=1:length(data.floor(fi).agents)
        % get agents data
        p = data.floor(fi).agents(ai).p;
        ri = data.floor(fi).agents(ai).r;
        vi = data.floor(fi).agents(ai).v;

        % get direction from nearest wall to agent
        nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
        ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));

        % get distance to nearest wall
        diW = lerp2(data.floor(fi).img_wall_dist, p(1), p(2));

        % get perpendicular and tangential unit vectors
        niW = [ nx ny];
        tiW = [-ny nx];


        % calculate force
        if diW < ri
            T1 = data.k * (ri - diW);
            T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
        else
            T1 = 0;
            T2 = 0;
        end
        Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;

        % add force to agent's current force
        data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
    end
end
```

Listing 31: addWallForce.m

```
function data = applyForcesAndMove(data)
%APPLYFORCESANDMOVE apply current forces to agents and move them using
%the timestep and current velocity

n_velocity_clamps = 0;
```

```
% loop over all floors higher than exit floor
for fi = data.floor_exit:data.floor_count

    % init logical arrays to indicate agents that change the floor or exit
    % the simulation
    floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);

    % loop over all agents
    for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
        v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f  / data.floor(fi).agents(ai).m;

        % clamp velocity
        if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
            n_velocity_clamps = n_velocity_clamps + 1;
        end

        % get agent's new position
        newp = data.floor(fi).agents(ai).p + ...
               v * data.dt / data.meter_per_pixel;

        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
        if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
                data.floor(fi).agents(ai).r

            % get agent's position
            p = data.floor(fi).agents(ai).p;

            % get wall distance gradient (which is off course perpendicular
            % to the nearest wall)
            nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
            ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
            n = [nx ny];

            % project out perpendicular component of velocity vector
            v = v - dot(n,v)/dot(n,n)*n;

            % get agent's new position
            newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
        end

        % check if agents position is ok
        % repositioning after 50 times clogging
        % deleting if agent has a NaN position
```

```
        if ~isnan(newp)
            if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
                newp = data.floor(fi).agents(ai).p;
                v = [0 0];
                data.floor(fi).agents(ai).clogged = data.floor(fi).agents(ai).clogged + 1;
                fprintf('WARNING: clogging agent %i on floor %i (%i). Position (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
                if data.floor(fi).agents(ai).clogged >= 40
                     nx = rand(1)*2 - 1;
                     ny = rand(1)*2 - 1;
                     n = [nx ny];
                     v = n*data.v_max/2;
                     fprintf('WARNING: agent %i on floor %i velocity set random to get out of wall. Position (%f,%f).\n',ai,fi,newp(1),newp(2))

                    % get agent's new position
                    newp = data.floor(fi).agents(ai).p + ...
                    v * data.dt / data.meter_per_pixel;
                    if isnan(newp)
                         % get rid of disturbing agent
                         fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
                         exited(ai) = 1;
                         data.agents_exited = data.agents_exited +1;
                         data.output.deleted_agents=data.output.deleted_agents+1;
                         newp = [1 1];
                    end
                end
            end
        else
            % get rid of disturbing agent
            fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;
            data.output.deleted_agents=data.output.deleted_agents+1;
            newp = [1 1];
        end


        % update agent's velocity and position
        data.floor(fi).agents(ai).v = v;
        data.floor(fi).agents(ai).p = newp;

        % reset forces for next timestep
        data.floor(fi).agents(ai).f = [0 0];

```

```
        % check if agent reached a staircase down and indicate floor change
        if data.floor(fi).img_stairs_down(round(newp(1)), round(newp(2)))
            floorchange(ai) = 1;
        end

        % check if agent reached an exit
        if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;

%             fprintf('agent exited from upper loop\n');

            %save current exit nr
            data.current_exit = data.exit_nr(round(newp(1)),
                round(newp(2)));

%             fprintf(int2str(data.current_exit));

            %update exit_left
            data.exit_left(1,data.current_exit) =
                data.exit_left(1,data.exit_nr(round(newp(1)),
                round(newp(2)))) - 1;

            %close exit if there is no more free space
            if data.exit_left(1,data.current_exit) < 1

                %change current exit to wall
                data.floor(data.floor_exit).img_wall =
                    data.floor(data.floor_exit).img_wall == 1 ...
                              | (data.exit_nr == (data.current_exit));
                data.floor(data.floor_exit).img_exit =
                    data.floor(data.floor_exit).img_exit == 1 ...
                              & (data.exit_nr ~= (data.current_exit));

                %redo initEscapeRoutes and initWallForces with new exit
                    and wall parameters
                data = initEscapeRoutes(data);
                data = initWallForces(data);

%                 fprintf('new routes from upper loop\n');

            end
        end
    end

    % add appropriate agents to next lower floor
    if fi > data.floor_exit
        data.floor(fi-1).agents = [data.floor(fi-1).agents
            data.floor(fi).agents(floorchange)];
    end
```

```
    % delete these and exited agents
    data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
end




% loop over all floors lower than exit floor
for fi = 1:data.floor_exit

    % init logical arrays to indicate agents that change the floor or exit
    % the simulation
    floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);

    % loop over all agents
    for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
        v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f  / data.floor(fi).agents(ai).m;

        % clamp velocity
        if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
            n_velocity_clamps = n_velocity_clamps + 1;
        end

        % get agent's new position
        newp = data.floor(fi).agents(ai).p + ...
               v * data.dt / data.meter_per_pixel;

        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
        if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
                data.floor(fi).agents(ai).r

            % get agent's position
            p = data.floor(fi).agents(ai).p;

            % get wall distance gradient (which is of course perpendicular
            % to the nearest wall)
            nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
            ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
            n = [nx ny];

            % project out perpendicular component of velocity vector
            v = v - dot(n,v)/dot(n,n)*n;
```

```
            % get agent's new position
            newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
        end


        % check if agents position is ok
        % repositioning after 50 times clogging
        % deleting if agent has a NaN position
        if ~isnan(newp)
            if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
               newp = data.floor(fi).agents(ai).p;
               v = [0 0];
               data.floor(fi).agents(ai).clogged =
                   data.floor(fi).agents(ai).clogged + 1;
               fprintf('WARNING: clogging agent %i on floor %i (%i). Position (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
               if data.floor(fi).agents(ai).clogged >= 40
                    nx = rand(1)*2 - 1;
                    ny = rand(1)*2 - 1;
                    n = [nx ny];
                    v = n*data.v_max/2;
                    fprintf('WARNING: agent %i on floor %i velocity set random to get out of wall. Position (%f,%f).\n',ai,fi,newp(1),newp(2))

                   % get agent's new position
                   newp = data.floor(fi).agents(ai).p + ...
                   v * data.dt / data.meter_per_pixel;
                   if isnan(newp)
                        % get rid of disturbing agent
                        fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
                        exited(ai) = 1;
                        data.agents_exited = data.agents_exited +1;
                        data.output.deleted_agents=data.output.deleted_agents+1;
                        newp = [1 1];
                   end
               end
            end
        else
            % get rid of disturbing agent
            fprintf('WARNING: position of an agent is NaN! Deleted this agent.\n')
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;
            data.output.deleted_agents=data.output.deleted_agents+1;
            newp = [1 1];
        end
```

```
        % update agent's velocity and position
        data.floor(fi).agents(ai).v = v;
        data.floor(fi).agents(ai).p = newp;

        % reset forces for next timestep
        data.floor(fi).agents(ai).f = [0 0];

        % check if agent reached a staircase up and indicate floor change
        if data.floor(fi).img_stairs_up(round(newp(1)), round(newp(2)))
            floorchange(ai) = 1;
        end

        % check if agent reached an exit
        if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
            exited(ai) = 1;
            data.agents_exited = data.agents_exited +1;

%               fprintf('agent exited from lower loop\n');

            %save current exit nr
            data.current_exit = data.exit_nr(round(newp(1)),
                round(newp(2)));

            %update exit_left
            data.exit_left(1,data.current_exit) =
                data.exit_left(1,data.exit_nr(round(newp(1)),
                round(newp(2)))) - 1;

                        %close exit if there is no more free space
            if data.exit_left(1,data.current_exit) < 1

                %change current exit to wall
                data.floor(data.floor_exit).img_wall =
                    data.floor(data.floor_exit).img_wall == 1 ...
                                | (data.exit_nr == (data.current_exit));
                data.floor(data.floor_exit).img_exit =
                    data.floor(data.floor_exit).img_exit == 1 ...
                                & (data.exit_nr ~= (data.current_exit));

                %redo initEscapeRoutes and initWallForces with new exit
                    and wall parameters
                data = initEscapeRoutes(data);
                data = initWallForces(data);

%                   fprintf('new routes from lower loop\n');

            end

        end
```

```
    end

    % add appropriate agents to next lower floor
    if fi < data.floor_exit
        data.floor(fi+1).agents = [data.floor(fi+1).agents ...
                                   data.floor(fi).agents(floorchange)];
    end

    % delete these and exited agents
    data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
end

if data.switch_done==0 && data.step ~=1 &&
    data.open_on_x_agents_on_boat>sum(data.output.agents_per_floor(:,data.step-1))
     data.floor(data.floor_exit).img_exit =
         data.floor(data.floor_exit).img_exit_second;
     data.floor(data.floor_exit).img_wall =
         data.floor(data.floor_exit).img_wall_second;
     data = initEscapeRoutes(data);
     data = initWallForces(data);
     data.switch_done=1;
     fprintf('ALL BOATS ARE OPEN NOW FOR EVACUATION! Opened on time
         %i\n',data.step*data.dt)
end

% if n_velocity_clamps > 0
%     fprintf(['WARNING: clamped velocity of %d agents, ' ...
%             'possible simulation instability.\n'], n_velocity_clamps);
% end
```

Listing 32: applyForcesAndMove.m

```
function val = checkForIntersection(data, floor_idx, agent_idx)
% check an agent for an intersection with another agent or a wall
% the check is kept as simple as possible
%
%  arguments:
%   data            global data structure
%   floor_idx       which floor to check
%   agent_idx       which agent on that floor
%   agent_new_pos   vector: [x,y], desired agent position to check
%
%  return:
%   0               for no intersection
%   1               has an intersection with wall
%   2                                   with another agent

val = 0;

p = data.floor(floor_idx).agents(agent_idx).p;
```

```
r = data.floor(floor_idx).agents(agent_idx).r;

% check for agent intersection
for i=1:length(data.floor(floor_idx).agents)
    if i~=agent_idx
        if norm(data.floor(floor_idx).agents(i).p-p)*data.meter_per_pixel
            ...
                <= r +  data.floor(floor_idx).agents(i).r
            val=2;
            return;
        end
    end
end


% check for wall intersection
if lerp2(data.floor(floor_idx).img_wall_dist, p(1), p(2)) < r
    val = 1;
end
```

Listing 33: checkForIntersection.m

```
mex 'fastSweeping.c'
mex 'getNormalizedGradient.c'
mex 'lerp2.c'
mex 'createRangeTree.c'
mex 'rangeQuery.c'
```

Listing 34: compileC.m

```
function data = initAgents(data)

% place agents randomly in desired spots, without overlapping



function radius = getAgentRadius()
    %radius of an agent in meters
    radius = data.r_min + (data.r_max-data.r_min)*rand();
end

data.agents_exited = 0; %how many agents have reached the exit
data.total_agent_count = 0;

floors_with_agents = 0;
agent_count = data.agents_per_floor;
for i=1:data.floor_count
    data.floor(i).agents = [];
    [y,x] = find(data.floor(i).img_spawn);
```

```
    if ~isempty(x)
        floors_with_agents = floors_with_agents + 1;
        for j=1:agent_count
            cur_agent = length(data.floor(i).agents) + 1;

            % init agent
            data.floor(i).agents(cur_agent).r = getAgentRadius();
            data.floor(i).agents(cur_agent).v = [0, 0];
            data.floor(i).agents(cur_agent).f = [0, 0];
            data.floor(i).agents(cur_agent).m = data.m;
            data.floor(i).agents(cur_agent).v0 = data.v0;
            data.floor(i).agents(cur_agent).clogged = 0; %to check if
                agent is hanging in the wall

            tries = 10;
            while tries > 0
                % randomly pick a spot and check if it's free
                idx = randi(length(x));
                data.floor(i).agents(cur_agent).p = [y(idx), x(idx)];
                if checkForIntersection(data, i, cur_agent) == 0
                    tries = -1; % leave the loop
                end
                tries = tries - 1;
            end
            if tries > -1
                %remove the last agent
                data.floor(i).agents = data.floor(i).agents(1:end-1);
            end
        end
        data.total_agent_count = data.total_agent_count +
            length(data.floor(i).agents);

        if length(data.floor(i).agents) ~= agent_count
            fprintf(['WARNING: could only place %d agents on floor %d ' ...
                'instead of the desired %d.\n'], ...
                length(data.floor(i).agents), i, agent_count);
        end
    end
end
if floors_with_agents==0
    error('no spots to place agents!');
end

end
```

Listing 35: initAgents.m

```
function data = initEscapeRoutes(data)
%INITESCAPEROUTES Summary of this function goes here
%   Detailed explanation goes here
```

```

for i=1:data.floor_count

    boundary_data = zeros(size(data.floor(i).img_wall));
    boundary_data(data.floor(i).img_wall) =  1;

if i<data.floor_exit
    boundary_data(data.floor(i).img_stairs_up) = -1;

elseif i>data.floor_exit
        boundary_data(data.floor(i).img_stairs_down) = -1;

    else
        boundary_data(data.floor(i).img_exit) = -1;

end
    exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    [data.floor(i).img_dir_x, data.floor(i).img_dir_y] = ...
        getNormalizedGradient(boundary_data, -exit_dist);
end
```

Listing 36: initEscapeRoutes.m

```
function data = initialize(config)
% initialize the internal data from the config data
%
%  arguments:
%   config      data structure from loadConfig()
%
%  return:
%   data        data structure: all internal data used for the main loop
%
%               all internal data is stored in pixels NOT in meters


data = config;

%for convenience
data.pixel_per_meter = 1/data.meter_per_pixel;

fprintf('Init escape routes...\n');
data = initEscapeRoutes(data);
fprintf('Init wall forces...\n');
data = initWallForces(data);
fprintf('Init agents...\n');
data = initAgents(data);

% maximum influence of agents on each other

data.r_influence = data.pixel_per_meter * ...
```

```
    fzero(@(r) data.A * exp((2*data.r_max-r)/data.B) - 1e-4, data.r_max);

fprintf('Init plots...\n');
%init the plots
%exit plot
data.figure_exit=figure;
hold on;
axis([0 data.duration 0 data.total_agent_count]);
title(sprintf('agents that reached the exit (total agents: %i)',
    data.total_agent_count));

% floors plot
data.figure_floors=figure;
% figure('units','normalized','outerposition',[0 0 1 1])
data.figure_floors_subplots_w = data.floor_count;
data.figure_floors_subplots_h = 4;
for i=1:config.floor_count
        data.floor(i).agents_on_floor_plot =
            subplot(data.figure_floors_subplots_h,
            data.figure_floors_subplots_w, 3*data.floor_count - i+1 +
            data.figure_floors_subplots_w);
        if i == config.floor_exit - 1
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w,
                [(2*config.floor_count+1):3*config.floor_count]);
        elseif i == config.floor_exit
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w,
                [(config.floor_count+1):2*config.floor_count]);
        elseif i == config.floor_exit + 1
            data.floor(i).building_plot =
                subplot(data.figure_floors_subplots_h,
                data.figure_floors_subplots_w, [1:config.floor_count]);
        end
end

% init output matrizes
data.output = struct;
data.output.config = config;
data.output.agents_per_floor =
    ones(data.floor_count,data.duration/data.dt).*(-1);
data.output.exit_left = zeros(data.exit_count,data.duration/data.dt);

% prepare output file name
data.output_file_name = ['output_' data.frame_basename];

% prepare video file name
data.video_file_name = ['video_' data.frame_basename '.avi'];
```

```
% set deleted_agents to zero
data.output.deleted_agents = 0;
```

Listing 37: initialize.m

```
function data = initWallForces(data)
%INITWALLFORCES init wall distance maps and gradient maps for each floor

for i=1:data.floor_count

    % init boundary data for fast sweeping method
    boundary_data = zeros(size(data.floor(i).img_wall));
    boundary_data(data.floor(i).img_wall) = -1;

    % get wall distance
    wall_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    data.floor(i).img_wall_dist = wall_dist;

    % get normalized wall distance gradient
    [data.floor(i).img_wall_dist_grad_x, ...
     data.floor(i).img_wall_dist_grad_y] = ...
     getNormalizedGradient(boundary_data, wall_dist-data.meter_per_pixel);
end
```

Listing 38: initWallForces.m

```
function config = loadConfig(config_file)
% load the configuration file
%
%  arguments:
%   config_file     string, which configuration file to load
%


% get the path from the config file -> to read the images
config_path = fileparts(config_file);
if strcmp(config_path, '') == 1
    config_path = '.';
end

fid = fopen(config_file);
input = textscan(fid, '%s=%s');
fclose(fid);

keynames = input{1};
values = input{2};

%convert numerical values from string to double
v = str2double(values);
```

```
idx = ~isnan(v);
values(idx) = num2cell(v(idx));

config = cell2struct(values, keynames);


% read the images
for i=1:config.floor_count

    %building structure
    file = config.(sprintf('floor_%d_build', i));
    file_name = [config_path '/' file];
    img_build = imread(file_name);

    % decode images
    config.floor(i).img_wall = (img_build(:, :, 1) ==   0 ...
                              & img_build(:, :, 2) ==   0 ...
                              & img_build(:, :, 3) ==   0);

    config.floor(i).img_spawn = (img_build(:, :, 1) == 255 ...
                               & img_build(:, :, 2) ==   0 ...
                               & img_build(:, :, 3) == 255);

%second possibility:
%pixel is exit if 1-->0, 3-->0, and if 2 is between 255 and 230 or if no
%red or blue

    config.floor(i).img_exit = (img_build(:, :, 1) ==   0 ...
                              & img_build(:, :, 2) ~=   0 ...
                              & img_build(:, :, 3) ==   0);


    config.floor(i).img_stairs_up = (img_build(:, :, 1) == 255 ...
                                   & img_build(:, :, 2) ==   0 ...
                                   & img_build(:, :, 3) ==   0);

    config.floor(i).img_stairs_down = (img_build(:, :, 1) ==   0 ...
                                     & img_build(:, :, 2) ==   0 ...
                                     & img_build(:, :, 3) == 255);


    if i == config.floor_exit

        %make the exit_nr matrix where the number of exit is indicated in
            each
        %pixel

        %make a zeroes matrix as big as img_exit
        config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));
```

```
        %make a zeros vector as long as floor_exit
        config.exit_left = zeros(1,config.exit_count);

        %loop over all exits
        for e=1:config.exit_count

            %build the exit_nr matrix
            config.exit_nr = config.exit_nr + e*( img_build(:, :, 1) == 0
                & img_build(:, :, 2) == (256-e) & img_build(:, :, 3) == 0 )
                ;

            %build the exit_left matrix
            config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));

        end
    end

    %init the plot image here, because this won't change
    config.floor(i).img_plot = 5*config.floor(i).img_wall ...
        + 4*config.floor(i).img_stairs_up ...
        + 3*config.floor(i).img_stairs_down ...
        + 2*config.floor(i).img_exit ...
        + 1*config.floor(i).img_spawn;
    config.color_map = [1 1 1; 0.9 0.9 0.9; 0 1 0; 0.4 0.4 1; 1 0.4 0.4; 0
        0 0];
end



% build open_second matrix
for i=1:config.open_second_nr
    config.open_second(i)=config.(sprintf('open_second_%i', i));
end

% save the "all exits open" configuration
config.floor(config.floor_exit).img_exit_second =
    config.floor(config.floor_exit).img_exit;
config.floor(config.floor_exit).img_wall_second =
    config.floor(config.floor_exit).img_wall;

% replace the open_second exits in img_exit with a wall
for i=1:config.open_second_nr
    config.floor(config.floor_exit).img_wall(find(config.exit_nr ==
        config.open_second(i))) = 1;
    config.floor(config.floor_exit).img_exit(find(config.exit_nr ==
        config.open_second(i))) = 0;
end

% set the boolean to check if switch from first to second mode has already
% been executed
```

```
config.switch_done = 0;
```

Listing 39: loadConfig.m

```
function plotAgentsPerFloor(data, floor_idx)
%plot time vs agents on floor

h = subplot(data.floor(floor_idx).agents_on_floor_plot);

set(h, 'position',[0.05+(data.floor_count -
    floor_idx)/(data.figure_floors_subplots_w+0.2), ...
     0.05, 1/(data.figure_floors_subplots_w*1.2), 0.3-0.05 ]);

if floor_idx~=data.floor_count
     set(h,'ytick',[]) %hide y-axis label
end

axis([0 data.time+data.dt 0 data.agents_per_floor*2]);

%axis([0 data.duration 0 data.agents_per_floor*2]);

hold on;
plot(data.time, length(data.floor(floor_idx).agents), 'b-');
hold off;

title(sprintf('%i', floor_idx));
```

Listing 40: plotAgentsPerFloor.m

```
function plotExitedAgents(data)
%plot time vs exited agents

hold on;
plot(data.time, data.agents_exited, 'r-');
hold off;
```

Listing 41: plotExitedAgents.m

```
function plotFloor(data, floor_idx)

if floor_idx == data.floor_exit-1 || floor_idx == data.floor_exit ||
    floor_idx == data.floor_exit+1
     h=subplot(data.floor(floor_idx).building_plot);

set(h,
    'position',[0,0.35+0.65/3*(floor_idx-data.floor_exit+1),1,0.65/3-0.005]);

hold off;
% the building image
imagesc(data.floor(floor_idx).img_plot);
```

```
hold on;

%plot options
colormap(data.color_map);
axis equal;
axis manual; %do not change axis on window resize

set(h, 'Visible', 'off')
% title(sprintf('floor %i', floor_idx))

% plot agents
if ~isempty(data.floor(floor_idx).agents)
    ang = [linspace(0,2*pi, 10) nan]';
    rmul = [cos(ang) sin(ang)] * data.pixel_per_meter;
    draw = cell2mat(arrayfun(@(a) repmat(a.p,length(ang),1) + a.r*rmul, ...
           data.floor(floor_idx).agents, 'UniformOutput', false)');
    line(draw(:,2), draw(:,1), 'Color', 'r');
end

hold off;
end
end
```

Listing 42: plotFloor.m

```
function simulate(config_file)
% run this to start the simulation

% start recording the matlab output window for debugging reasons
diary log

if nargin==0
    config_file='../data/config1.conf';
end

fprintf('Load config file...\n');
config = loadConfig(config_file);

data = initialize(config);

data.step = 1;
data.time = 0;
fprintf('Start simulation...\n');

% tic until simulation end
simstart = tic;

%make video while simulation
if data.save_frames==1
        vidObj=VideoWriter(data.video_file_name);
```

```
            open(vidObj);
        end

while (data.time < data.duration)
    % tic until timestep end
    tstart=tic;
    data = addDesiredForce(data);
    data = addWallForce(data);
    data = addAgentRepulsiveForce(data);
    data = applyForcesAndMove(data);

    % dump agents_per_floor to output
    for floor=1:data.floor_count
          data.output.agents_per_floor(floor,data.step) =
              length(data.floor(floor).agents);
    end

    % dump exit_left to output
    data.output.exit_left(:,data.step) = data.exit_left';

    if mod(data.step,data.save_step) == 0

        % do the plotting
        set(0,'CurrentFigure',data.figure_floors);
        for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
            plotFloor(data, floor);
        end

        if data.save_frames==1
%              print('-depsc2',sprintf('frames/%s_%04i.eps', ...
%                  data.frame_basename,data.step), data.figure_floors);

%           make video while simulate
            currFrame=getframe(data.figure_floors);
            writeVideo(vidObj,currFrame);

        end

        set(0,'CurrentFigure',data.figure_exit);
        plotExitedAgents(data);

        if data.agents_exited == data.total_agent_count
            fprintf('All agents are now saved (or are they?). Time: %.2f
                sec\n', data.time);
            fprintf('Total Agents: %i\n', data.total_agent_count);

            print('-depsc2',sprintf('frames/exited_agents_%s.eps', ...
                data.frame_basename), data.figure_floors);
            break;
```

```
        end

        % toc of timestep
        data.telapsed = toc(tstart);
        % toc of whole simulation
        data.output.simulation_time = toc(simstart);

        % save output
        output = data.output;
        save(data.output_file_name,'output')
        fprintf('Frame %i done (took %.3fs; %.3fs out of %.3gs
            simulated).\n', data.step, data.telapsed, data.time,
            data.duration);

    end

    % update step
    data.step = data.step+1;

    % update time
    if (data.time + data.dt > data.duration)
        data.dt = data.duration - data.time;
        data.time = data.duration;
    else
        data.time = data.time + data.dt;
    end

end

%make video while simulation
close(vidObj);

% toc of whole simulation
data.output.simulation_time = toc(simstart);

% save complete simulation
output = data.output;
save('output','output')
fprintf('Simulation done in %i seconds and saved data to output file.\n',
    data.output.simulation_time);

% save diary
diary
```

Listing 43: simulate.m