



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



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  
Eugster  
Brechbuehler  
Reber  
Schmid

First name  
Manuela  
Raphael  
Andreas  
Fabian

### Supervising lecturer

Last name  
Baliotti  
Donnay

First name  
Stefano  
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](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.

Zurich, 20.11.2012

Place and date

Signature

*R. Brechbuehler*  
*A. Reber*  
*F. Baliotti*  
*G. Eugster*

\*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

<b>1 Abstract</b>	<b>6</b>
<b>2 Individual contributions</b>	<b>6</b>
<b>3 Introduction and Motivations</b>	<b>7</b>
3.1 Introduction . . . . .	7
3.2 Motivation . . . . .	7
3.3 Fundamental Questions . . . . .	7
3.4 Expected Results . . . . .	8
<b>4 Description of the model</b>	<b>10</b>
4.1 Social force model . . . . .	10
4.2 Ship structure . . . . .	11
<b>5 Implementation</b>	<b>12</b>
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
<b>6 Results and Discussion</b>	<b>22</b>
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
<b>7</b>	<b>Outlook</b>	<b>26</b>
<b>8</b>	<b>References</b>	<b>27</b>
<b>9</b>	<b>Appendix</b>	<b>28</b>
9.1	Code . . . . .	28
9.1.1	<i>standard</i> code . . . . .	28
9.1.2	<i>C</i> code . . . . .	53
9.1.3	<i>crew command</i> 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 [2]. A implementation of this has already been done by a former MSSSM group [3] 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 implenting 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 timeline 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 save 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 [4]. 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, behaviour 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 [7]. 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 [8]. Certainly we know that this science is very advanced and practised 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 [6]. After we localised 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 analyse 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 [3]. 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 [2], 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.

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i f_D + \sum_{j(\neq i)} f_{ij} + \sum_W f_{iW} \quad (1)$$

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  $\tau_i$ .

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

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.

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

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.

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

## 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 [6]. 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 [3], 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.

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

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

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

### 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 [3] 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 continuative 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 helpfull 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 rescueboats, 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 variable *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 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. Afterwards 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 too 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<sup>th</sup> or 100<sup>th</sup>. 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 analyse 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 analyse 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 fulfill 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 colours have to be replaced by predefined colours 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.

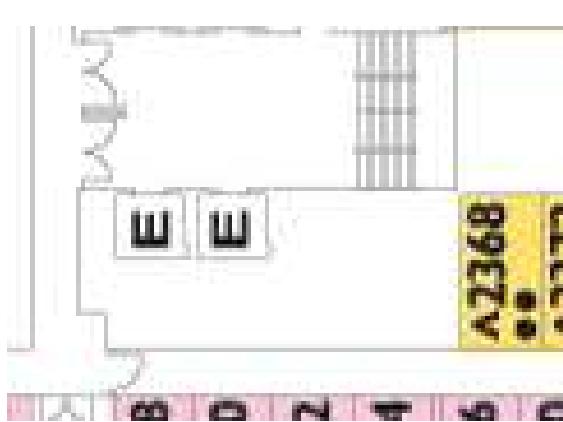


Figure 6: Deck before conversion

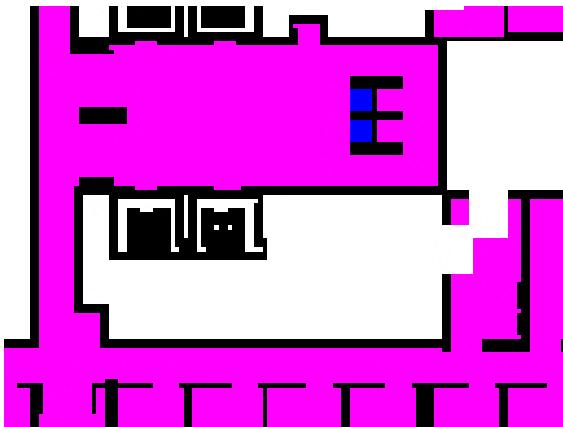


Figure 7: Deck after conversion

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" [3]. 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 spiralling 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 spiralling stairs. For the ship in that project this is sufficient, because the modelling fault will be in a lower degree than the one of other assumptions.

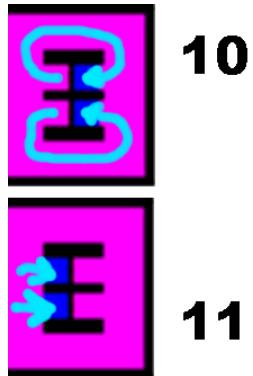


Figure 8: Stairs connection between floor 10 and floor 11

### 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 neglectable 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.

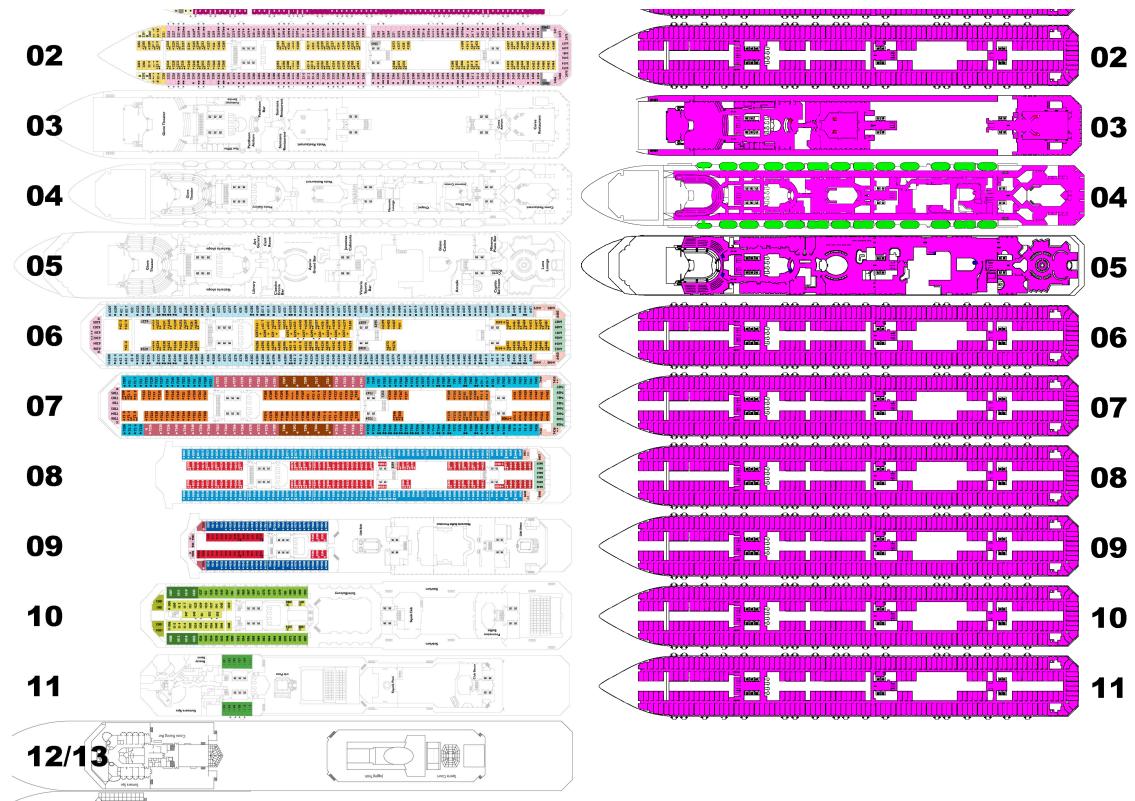


Figure 9: Original decks before conversion

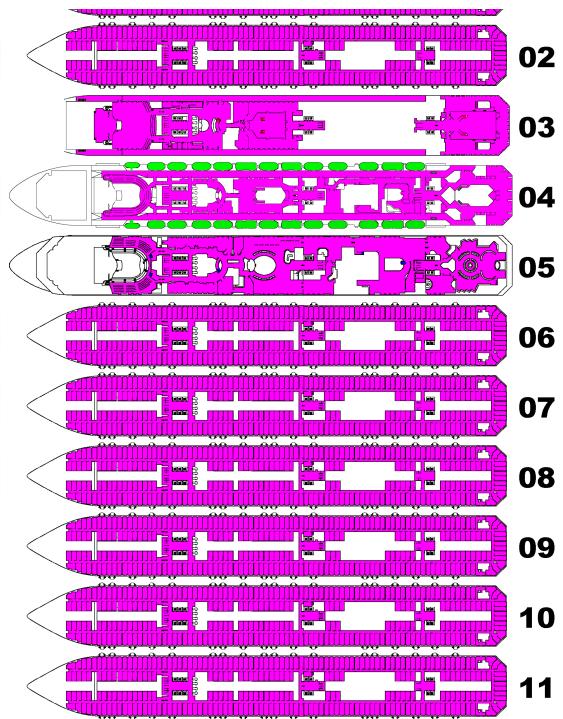


Figure 10: Deck approximated and converted

## 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 behaviour 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:

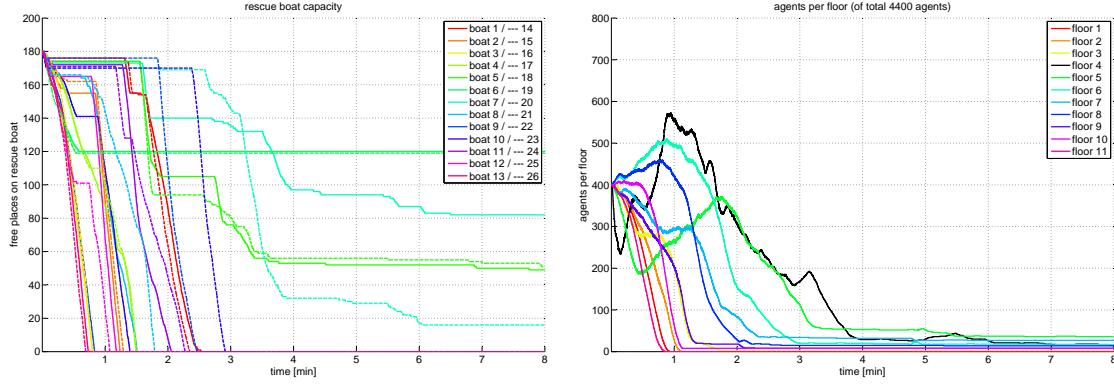


Figure 11: Standard simulation: Boat capacities during first simulation

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.

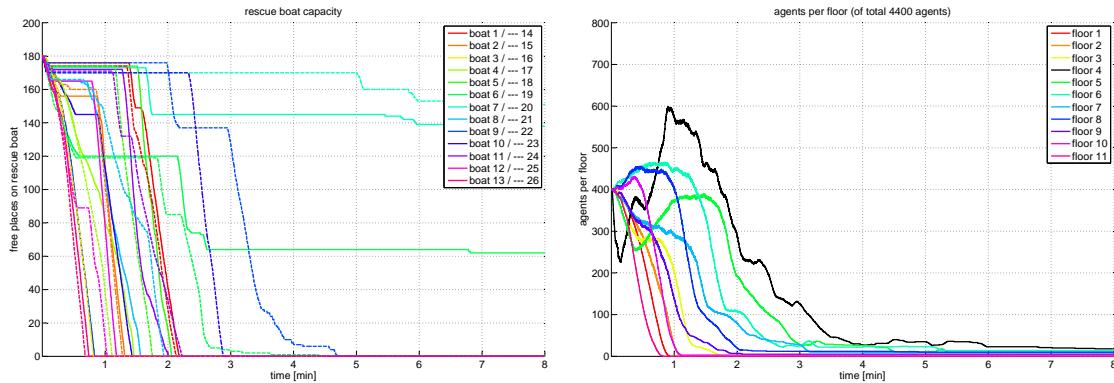


Figure 13: Added stairs simulation: Boat capacities during simulation

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 rescueboats 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.

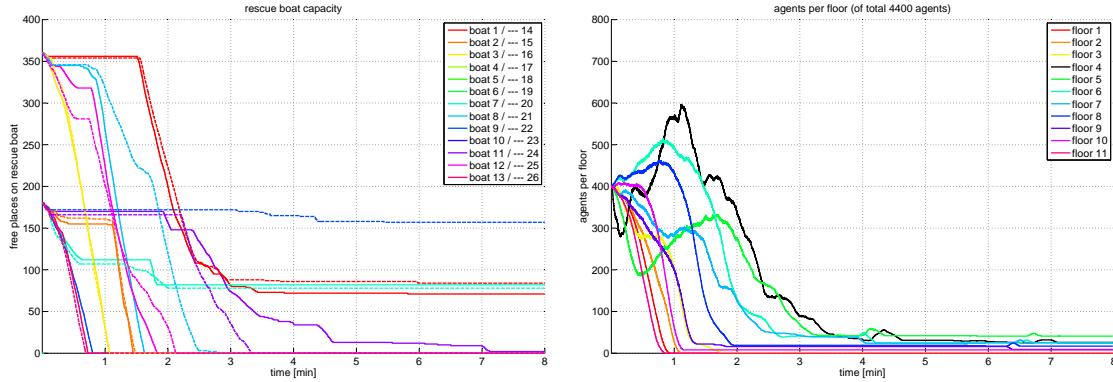


Figure 15: Varied boatsize simulation: Boat capacities during simulation

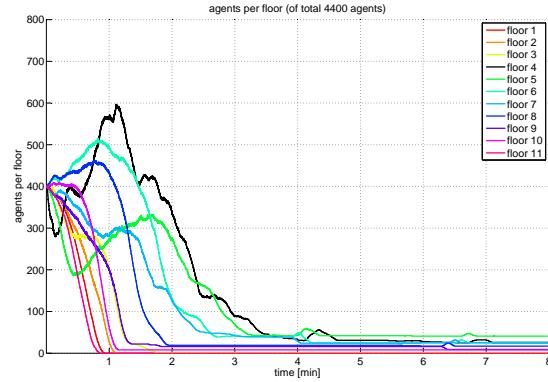


Figure 16: Varied boatsize simulation: Number of agents per floor

### 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 extention 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.

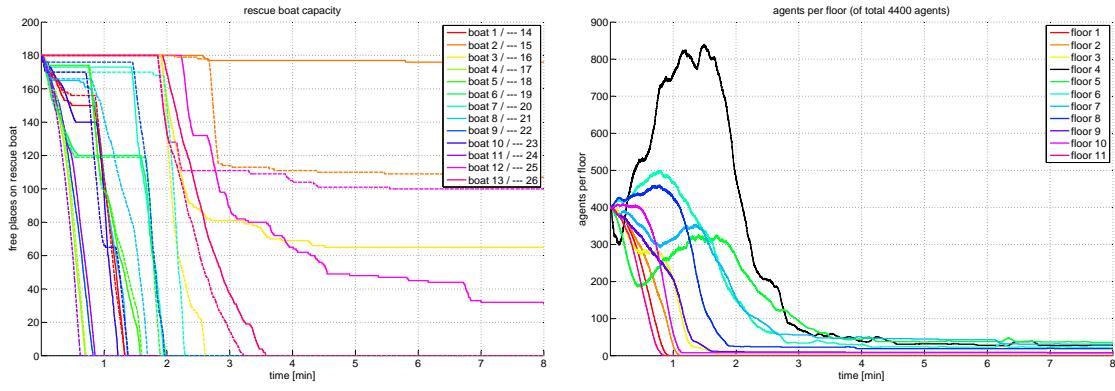


Figure 17: Crew command simulation: Boat capacities during simulation

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 stanard model. In contrast to that the rescueboat capacity utilisation 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 extention 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 saver. 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 possibilites to improve the model. The target of an ongoing project could be to make the model more realistic. There are a lot of possibilites to achieve this goal.

Some suggestions:

- To take into account that not all people automatically know where exactly the nearest exit is, the initialisation 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 rescueboats by a crew member. It would be interesting to analyse 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] Helbing, Dirk (1995): Social Force Model for Pedestrians Dynamics.
- [2] Helbling, Dirk et al (2000): Simulating dynamical features of escape panic.
- [3] Hardmeier,Jenal,Kueng,Thaler (2012): Modelling Situations of Evacuation in a Multi-level Building.
- [4] 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](http://www.imo.org/about/conventions/listofconventions/pages/international-convention-for-the-safety-of-life-at-sea-(solas),-1974.aspx)
- [5] SHIP EVACUATION: <http://www.shipevacuation.com/>
- [6] CRUISE DECK PLANS PUBLIC SITE: <http://www.cruisedeckplans.com/DP/Main/decks.php?ship=CostaSerena>
- [7] University of Greenwich (2011): maritimeEXODUS. [http://fseg.gre.ac.uk/fire/marine\\_evac\\_model.html](http://fseg.gre.ac.uk/fire/marine_evac_model.html)

- [8] Haverie of Costa Concordia (2012): [http://de.wikipedia.org/wiki/Costa\\_Concordia#Havarie\\_2012](http://de.wikipedia.org/wiki/Costa_Concordia#Havarie_2012)

## 9 Appendix

### 9.1 Code

#### 9.1.1 *standard code*

---

```

1 function data = addAgentRepulsiveForce(data)
%ADDAGENTREPULSIVEFORCE Summary of this function goes here
3 % Detailed explanation goes here

5 % Obstruction effects in case of physical interaction

7 % get maximum agent distance for which we calculate force
r_max = data.r_influence;
9 tree = 0;

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

15     % update range tree of lower floor
tree_lower = tree;

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

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

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

29         % use range tree to get the indices of all agents near agent ai
31         idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
                           pi(2) - r_max, pi(2) + r_max)';
33
35             % loop over agents near agent ai
            for aj = idx

37                 % if force has not been calculated yet...
                    if aj > ai

```

```

39         pj = data.floor(fi).agents(aj).p;
40         vj = data.floor(fi).agents(aj).v;
41         rj = data.floor(fi).agents(aj).r;

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

45         % distance of agents
46         d = norm(nij);

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

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

53         % repulsive interaction forces
54         if d < rij
55             T1 = data.k*(rij - d);
56             T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
57         else
58             T1 = 0;
59             T2 = 0;
60         end
61
62         F = (data.A * exp((rij - d)/data.B) + T1)*nij + T2;
63
64         data.floor(fi).agents(ai).f = ...
65             data.floor(fi).agents(ai).f + F;
66         data.floor(fi).agents(aj).f = ...
67             data.floor(fi).agents(aj).f - F;
68
69     end
70
71     end
72
73     % include agents on stairs!
74     if fi > 1
75         % use range tree to get the indices of all agents near agent ai
76         if ~isempty(data.floor(fi-1).agents)
77             idx = rangeQuery(tree_lower, pi(1) - r_max, ...
78                               pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';
79
80             % if there are any agents...
81             if ~isempty(idx)
82                 for aj = idx
83                     pj = data.floor(fi-1).agents(aj).p;
84                     if data.floor(fi-1).img_stairs_up(round(pj(1)),
85                                                 round(pj(2)))
86
87

```

```

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

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

93         % distance of agents
94         d = norm(nij);

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

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

103        % repulsive interaction forces
104        if d < rij
105            T1 = data.k*(rij - d);
106            T2 = data.kappa*(rij - d)*dot((vj -
107                vi),tij)*tij;
108        else
109            T1 = 0;
110            T2 = 0;
111        end
112
113        F = (data.A * exp((rij - d)/data.B) + T1)*nij
114            + T2;
115
116        data.floor(fi).agents(ai).f = ...
117            data.floor(fi).agents(ai).f + F;
118        data.floor(fi-1).agents(aj).f = ...
119            data.floor(fi-1).agents(aj).f - F;
120
121    end
122
123    end
124
125 end
end

```

Listing 1: addAgentRepulsiveForce.m

---

```

1 function data = addDesiredForce(data)
2 %ADDDESIREDFORCE add 'desired' force contribution (towards nearest exit or
3 %staircase)

5 for fi = 1:data.floor_count

```

```

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

9     % get agent's data
10    p = data.floor(fi).agents(ai).p;
11    m = data.floor(fi).agents(ai).m;
12    v0 = data.floor(fi).agents(ai).v0;
13    v = data.floor(fi).agents(ai).v;

15
16    % get direction towards nearest exit
17    ex = lerp2(data.floor(fi).img_dir_x, p(1), p(2));
18    ey = lerp2(data.floor(fi).img_dir_y, p(1), p(2));
19    e = [ex ey];

21    % get force
22    Fi = m * (v0*e - v)/data.tau;
23
24    % add force
25    data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
26  end
27 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);

```

```

28         T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
29     else
30         T1 = 0;
31         T2 = 0;
32     end
33     Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;
34
35     % add force to agent's current force
36     data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
37
38 end

```

Listing 3: addWallForce.m

```

function data = applyForcesAndMove(data)
%APPLYFORCESANDMOVE apply current forces to agents and move them using
%the timestep and current velocity
1
2 n_velocity_clamps = 0;
3
4 % loop over all floors higher than exit floor
5 for fi = data.floor_exit:data.floor_count
6
7     % init logical arrays to indicate agents that change the floor or exit
    % the simulation
8     floorchange = false(length(data.floor(fi).agents),1);
9     exited = false(length(data.floor(fi).agents),1);
10
11     % loop over all agents
12     for ai=1:length(data.floor(fi).agents)
13         % add current force contributions to velocity
14         v = data.floor(fi).agents(ai).v + data.dt * ...
15             data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
16
17         % clamp velocity
18         if norm(v) > data.v_max
19             v = v / norm(v) * data.v_max;
20             n_velocity_clamps = n_velocity_clamps + 1;
21         end
22
23         % get agent's new position
24         newp = data.floor(fi).agents(ai).p + ...
25             v * data.dt / data.meter_per_pixel;
26
27         % if the new position is inside a wall, remove perpendicular
28         % component of the agent's velocity
29         if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
30             data.floor(fi).agents(ai).r
31
32             % get agent's position
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
645
646
647
647
648
649
649
650
651
652
653
653
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1
```

```

38     p = data.floor(fi).agents(ai).p;
39
40     % get wall distance gradient (which is off course perpendicular
41     % to the nearest wall)
42     nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
43     ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
44     n = [nx ny];
45
46     % project out perpendicular component of velocity vector
47     v = v - dot(n,v)/dot(n,n)*n;
48
49     % get agent's new position
50     newp = data.floor(fi).agents(ai).p + ...
51             v * data.dt / data.meter_per_pixel;
52 end
53
54     % check if agents position is ok
55     % repositioning after 50 times clogging
56     % deleting if agent has a NaN position
57 if ~isnan(newp)
58     if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
59         newp = data.floor(fi).agents(ai).p;
60         v = [0 0];
61         data.floor(fi).agents(ai).clogged =
62             data.floor(fi).agents(ai).clogged + 1;
63         fprintf('WARNING: clogging agent %i on floor %i (%i).
64             Position
65             (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
66 if data.floor(fi).agents(ai).clogged >= 40
67     nx = rand(1)*2 - 1;
68     ny = rand(1)*2 - 1;
69     n = [nx ny];
70     v = n*data.v_max/2;
71     fprintf('WARNING: agent %i on floor %i velocity set
72             random to get out of wall. Position
73             (%f,%f).\n',ai,fi,newp(1),newp(2))
74
75     % get agent's new position
76     newp = data.floor(fi).agents(ai).p + ...
77             v * data.dt / data.meter_per_pixel;
78     if isnan(newp)
79         % get rid of disturbing agent
80         fprintf('WARNING: position of an agent is NaN!
81             Deleted this agent.\n')
82         exited(ai) = 1;
83         data.agents_exited = data.agents_exited +1;
84         data.output.deleted_agents=data.output.deleted_agents+1;
85         newp = [1 1];
86     end
87 end

```

```

        end
82    else
83        % get rid of disturbing agent
84        fprintf('WARNING: position of an agent is NaN! Deleted this
85            agent.\n')
86        exited(ai) = 1;
87        data.agents_exited = data.agents_exited +1;
88        data.output.deleted_agents=data.output.deleted_agents+1;
89        newp = [1 1];
90    end

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

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

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

104    % check if agent reached an exit
105    if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
106       exited(ai) = 1;
107       data.agents_exited = data.agents_exited +1;
108
109    %
110    fprintf('agent exited from upper loop\n');

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

114    %
115    fprintf(int2str(data.current_exit));

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

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

122    %change current exit to wall
123    data.floor(data.floor_exit).img_wall =
124        data.floor(data.floor_exit).img_wall == 1 ...
125            | (data.exit_nr == (data.current_exit));

```

```

126
128
130
132 %
134
136
138
140
142
144
146
148
150
152
154
156
158
160
162
164
166
168
170

    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

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
    end
end

```

```

172     newp = data.floor(fi).agents(ai).p + ...
174         v * data.dt / data.meter_per_pixel;
176
176     % if the new position is inside a wall, remove perpendicular
176     % component of the agent's velocity
177     if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
178         data.floor(fi).agents(ai).r
179
180         % get agent's position
181         p = data.floor(fi).agents(ai).p;
182
184         % get wall distance gradient (which is of course perpendicular
184         % to the nearest wall)
185         nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
186         ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
187         n = [nx ny];
188
189         % project out perpendicular component of velocity vector
190         v = v - dot(n,v)/dot(n,n)*n;
191
192         % get agent's new position
193         newp = data.floor(fi).agents(ai).p + ...
194             v * data.dt / data.meter_per_pixel;
195     end
196
197
198     % check if agents position is ok
199     % repositioning after 50 times clogging
200     % deleting if agent has a NaN position
201     if ~isnan(newp)
202         if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
203             newp = data.floor(fi).agents(ai).p;
204             v = [0 0];
205             data.floor(fi).agents(ai).clogged =
206                 data.floor(fi).agents(ai).clogged + 1;
207             fprintf('WARNING: clogging agent %i on floor %i (%i).
207                 Position
207                 (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
208         if data.floor(fi).agents(ai).clogged >= 40
209             nx = rand(1)*2 - 1;
210             ny = rand(1)*2 - 1;
211             n = [nx ny];
212             v = n*data.v_max/2;
213             fprintf('WARNING: agent %i on floor %i velocity set
213                 random to get out of wall. Position
213                 (%f,%f).\n',ai,fi,newp(1),newp(2))
214
215         % get agent's new position
216         newp = data.floor(fi).agents(ai).p + ...
216             v * data.dt / data.meter_per_pixel;

```

```

218         if isnan(newp)
219             % get rid of disturbing agent
220             fprintf('WARNING: position of an agent is NaN!
221                 Deleted this agent.\n')
222             exited(ai) = 1;
223             data.agents_exited = data.agents_exited +1;
224             data.output.deleted_agents=data.output.deleted_agents+1;
225             newp = [1 1];
226         end
227     end
228 else
229     % get rid of disturbing agent
230     fprintf('WARNING: position of an agent is NaN! Deleted this
231         agent.\n')
232     exited(ai) = 1;
233     data.agents_exited = data.agents_exited +1;
234     data.output.deleted_agents=data.output.deleted_agents+1;
235     newp = [1 1];
236 end
237
238 % update agent's velocity and position
239 data.floor(fi).agents(ai).v = v;
240 data.floor(fi).agents(ai).p = newp;
241
242 % reset forces for next timestep
243 data.floor(fi).agents(ai).f = [0 0];
244
245 % check if agent reached a staircase up and indicate floor change
246 if data.floor(fi).img_stairs_up(round(newp(1)), round(newp(2)))
247     floorchange(ai) = 1;
248 end
249
250 % check if agent reached an exit
251 if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
252     exited(ai) = 1;
253     data.agents_exited = data.agents_exited +1;
254 %
255     fprintf('agent exited from lower loop\n');
256
257 %save current exit nr
258 data.current_exit = data.exit_nr(round(newp(1)),
259                                 round(newp(2)));
260
261 %update exit_left
262 data.exit_left(1,data.current_exit) =
263     data.exit_left(1,data.exit_nr(round(newp(1)),
264                     round(newp(2)))) - 1;
265
266 %close exit if there is no more free space

```

```

262         if data.exit_left(1,data.current_exit) < 1
264
266             %change current exit to wall
267             data.floor(data.floor_exit).img_wall =
268                 data.floor(data.floor_exit).img_wall == 1 ...
269                     | (data.exit_nr == (data.current_exit));
270             data.floor(data.floor_exit).img_exit =
271                 data.floor(data.floor_exit).img_exit == 1 ...
272                     & (data.exit_nr ~= (data.current_exit));
273
274             %redo initEscapeRoutes and initWallForces with new exit
275             % and wall parameters
276             data = initEscapeRoutes(data);
277             data = initWallForces(data);
278
279             fprintf('new routes from lower loop\n');
280
281         end
282
283     end
284
285     % add appropriate agents to next lower floor
286     if fi < data.floor_exit
287         data.floor(fi+1).agents = [data.floor(fi+1).agents ...
288                                     data.floor(fi).agents(floorchange)];
289     end
290
291     % delete these and exited agents
292     data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
293
294 % 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:

```

```

12 % 0           for no intersection
13 % 1           has an intersection with wall
14 % 2           with another agent

16 val = 0;

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

20
21 % check for agent intersection
22 for i=1:length(data.floor(floor_idx).agents)
23     if i~=agent_idx
24         if norm(data.floor(floor_idx).agents(i).p-p)*data.meter_per_pixel
25             ...
26             <= r + data.floor(floor_idx).agents(i).r
27         val=2;
28         return;
29     end
30 end

32
33 % check for wall intersection
34 if lerp2(data.floor(floor_idx).img_wall_dist, p(1), p(2)) < r
35     val = 1;
36 end

```

Listing 5: checkForIntersection.m

---

```

1 mex 'fastSweeping.c'
2 mex 'getNormalizedGradient.c'
3 mex 'lerp2.c'
4 mex 'createRangeTree.c'
5 mex 'rangeQuery.c'

```

---

Listing 6: compileC.m

---

```

1 function data = initAgents(data)

3 % place agents randomly in desired spots, without overlapping
4

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

11 data.agents_exited = 0; %how many agents have reached the exit
12 data.total_agent_count = 0;
13

```

---

```

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

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

25 % init agent
27 data.floor(i).agents(cur_agent).r = getAgentRadius();
data.floor(i).agents(cur_agent).v = [0, 0];
29 data.floor(i).agents(cur_agent).f = [0, 0];
data.floor(i).agents(cur_agent).m = data.m;
31 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

33 tries = 10;
35 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
43 if tries > -1
    %remove the last agent
    data.floor(i).agents = data.floor(i).agents(1:end-1);
end
47 end
49 data.total_agent_count = data.total_agent_count +
    length(data.floor(i).agents);

51 if length(data.floor(i).agents) ~= agent_count
    fprintf(['WARNING: could only place %d agents on floor %d , ...
53 'instead of the desired %d.\n'], ...
    length(data.floor(i).agents), i, agent_count);
55 end
end
57 end
if floors_with_agents==0
    error('no spots to place agents!');
end
59
61

```

```
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);
22 fprintf('Init agents...\n');
    data = initAgents(data);
24
% maximum influence of agents on each other
26
data.r_influence = data.pixel_per_meter * ...
28     fzero(@(r) data.A * exp((2*data.r_max-r)/data.B) - 1e-4, data.r_max);

30 fprintf('Init plots...\n');
    %init the plots
32 %exit plot
    data.figure_exit=figure;
34 hold on;
    axis([0 data.duration 0 data.total_agent_count]);
36 title(sprintf('agents that reached the exit (total agents: %i)', ...
    data.total_agent_count));

38 % floors plot
    data.figure_floors=figure;
40 % figure('units','normalized','outerposition',[0 0 1 1])
    data.figure_floors_subplots_w = data.floor_count;
42 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
52 end

54 % init output matrizes
    data.output = struct;
56 data.output.config = config;
    data.output.agents_per_floor =
        ones(data.floor_count,data.duration/data.dt).*(-1);

```

```

58 data.output.exit_left = zeros(data.exit_count,data.duration/data.dt);

60 % prepare output file name
61 data.output_file_name = [ 'output_' data.frame_basename];
62
63 % prepare video file name
64 data.video_file_name = [ 'video_' data.frame_basename '.avi'];

66 % set deleted_agents to zero
67 data.output.deleted_agents = 0;

```

Listing 9: initialize.m

```

1 function data = initWallForces(data)
2 %INITWALLFORCES init wall distance maps and gradient maps for each floor

4 for i=1:data.floor_count

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

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

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

```

Listing 10: initWallForces.m

```

1 function config = loadConfig(config_file)
2 % load the configuration file
3 %
4 % arguments:
5 % config_file      string, which configuration file to load
6 %
7

9 % get the path from the config file -> to read the images
10 config_path = fileparts(config_file);
11 if strcmp(config_path, '') == 1
12     config_path = '.';
13 end

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

```

```

17 fclose(fid);

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

21 %convert numerical values from string to double
23 v = str2double(values);
idx = ~isnan(v);
25 values(idx) = num2cell(v(idx));

27 config = cell2struct(values, keynames);

29
% read the images
31 for i=1:config.floor_count

33 %building structure
file = config.(sprintf('floor_%d_build', i));
35 file_name = [config_path '/' file];
img_build = imread(file_name);

37 % decode images
39 config.floor(i).img_wall = (img_build(:, :, 1) == 0 ...
& img_build(:, :, 2) == 0 ...
& img_build(:, :, 3) == 0);

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

45

47 %second possibility:
%pixel is exit if 1-->0, 3-->0, and if 2 is between 255 and 230 or if no
49 %red or blue

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

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

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

59

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

63

65 if i == config.floor_exit

```

```

67      %make the exit_nr matrix where the number of exit is indicated in
       each
       %pixel
69
71      %make a zeroes matrix as big as img_exit
config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));

73      %make a zeros vector as long as floor_exit
config.exit_left = zeros(1,config.exit_count);
75
77      %loop over all exits
for e=1:config.exit_count

79          %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 )
    ;
81
83          %build the exit_left matrix
config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));

85      end
end
87
89      %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];
95 end

```

Listing 11: loadConfig.m

---

```

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

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

6 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 ]);
8
10 if floor_idx~=data.floor_count
    set(h,'ytick',[]) %hide y-axis label
end
12 axis([0 data.time+data.dt 0 data.agents_per_floor*2]);

```

```

14 %axis([0 data.duration 0 data.agents_per_floor*2]);
16 hold on;
18 plot(data.time, length(data.floor(floor_idx).agents), 'b-');
hold off;
20 title(sprintf('%i', floor_idx));

```

Listing 12: plotAgentsPerFloor.m

```

function plotExitedAgents(data)
%plot time vs exited agents

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

```

Listing 13: plotExitedAgents.m

```

function plotFloor(data, floor_idx)
2
if floor_idx == data.floor_exit-1 || floor_idx == data.floor_exit ||
    floor_idx == data.floor_exit+1
4 h=subplot(data.floor(floor_idx).building_plot);

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

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

12 %plot options
14 colormap(data.color_map);
axis equal;
16 axis manual; %do not change axis on window resize

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

20 % plot agents
22 if ~isempty(data.floor(floor_idx).agents)
    ang = [linspace(0,2*pi, 10) nan]';
24    rmul = [cos(ang) sin(ang)] * data.pixel_per_meter;
    draw = cell2mat(arrayfun(@(a) repmat(a.p,length(ang),1) + a.r*rmul, ...
26        data.floor(floor_idx).agents, 'UniformOutput', false));
    line(draw(:,2), draw(:,1), 'Color', 'r');
28 end

```

```
30 hold off;
end
32 end
```

Listing 14: plotFloor.m

```
% post processing of output.mat data from simulation
2 % to run, you need to load the output first:
% load('output_FILENAME');
4
% tabula rasa
6 clc

8 % read in data from output
agents_per_floor = output.agents_per_floor;
10 config = output.config;
exit_left = output.exit_left;
12 simulation_time_real = output.simulation_time;
dt = config.dt;
14 deleted_agents = output.deleted_agents;

16
% get users screen size
18 screen_size = get(0, 'ScreenSize');

20 % agents on boat
agents_on_boat = sum(agents_per_floor(:,1:1:length(agents_per_floor)));
22
% check if whole simulation was performed
24 steps=config.duration/dt-1;
for i=1:steps
    if agents_on_boat(i)<0
        steps=i-2;
        break
    end
30 end

32 simulation_time_sim = steps*dt;

34 % recalculate agents on boat
agents_on_boat = sum(agents_per_floor(:,1:1:steps));
36 agents_start = agents_on_boat(1);
agents_left = agents_start-agents_on_boat;
38
% find out t10, t50, t90, t100
40 t10=0;
for i=1:steps
    if agents_left(i)<agents_start/10
        t10=t10+dt;
```

```

44     end
45 end
46 if t10 ~=0
47     t10=t10+dt;
48 end

50 t50=0;
51 for i=1:steps
52     if agents_left(i)<agents_start/2
53         t50=t50+dt;
54     end
55 end
56 if t50 ~=0
57     t50=t50+dt;
58 end

60 t90=0;
61 for i=1:steps
62     if agents_left(i)<agents_start*0.9
63         t90=t90+dt;
64     end
65 end
66 if t90 ~=0
67     t90=t90+dt;
68 end

70 t95=0;
71 for i=1:steps
72     if agents_left(i)<agents_start*0.95
73         t95=t95+dt;
74     end
75 end
76 if t95 ~=0
77     t95=t95+dt;
78 end

80 t100=0;
81 if agents_left==agents_start
82     for i=1:steps
83         if agents_left(i)<agents_start
84             t100=t100+dt;
85         end
86     end
87 end

88 % create time axis
89 if t100 ~=0
90     time = [0:dt:t100];
91 else
92     time = [0:dt:simulation_time_sim];

```

```

94 end
steps = length(time);
96
% recalculate agents on boat
98 agents_on_boat = sum(agents_per_floor(:,1:1:steps));
agents_start = agents_on_boat(1);
100 agents_left = agents_start-agents_on_boat;
agents_per_floor = agents_per_floor(:,1:1:steps);
102 exit_left = exit_left(:,1:1:steps);

104 % plot agents left over time
f1 = figure;
106 hold on
grid on
108 set(gca,'XTick',[1:1:8], 'FontSize',16)
plot(time/60,agents_left/agents_start*100, 'LineWidth', 2)
110 axis([0 8 0 100])
title(sprintf('rescued agents (of total %i agents)',agents_start));
112 xlabel('time [min]')
ylabel('rescued agents out of all agents [%]')

114 % plot agents_per_floor over time
116 f2 = figure;
hold on
grid on
set(gca,'XTick',[1:1:8], 'FontSize',16)
118 list = cell(config.floor_count,1);
color = hsv(config.floor_count);
120 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)
128 axis([0 8 0 800])
130 title(sprintf('agents per floor (of total %i agents)',agents_start));
xlabel('time [min]')
132 ylabel('agents per floor')

134 % plot free places in rescue boats over time
f3 = figure;
136 hold on
grid on
138 set(gca,'XTick',[1:1:8], 'FontSize',16)
list = cell(config.exit_count/2,1);
color = hsv(config.exit_count/2);
140 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

```

```

144 end
145 for i=config.exit_count/2+1:config.exit_count
146     plot(time/60,exit_left(i,:), '--', 'LineWidth',
147           2,'color',color(i-config.exit_count/2,:))
148 end
149 legend(list)

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

154 % scale plots up to screen size
155 set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );
156 set(f2, 'Position', [0 0 screen_size(3) screen_size(4) ] );
157 set(f3, 'Position', [0 0 screen_size(3) screen_size(4) ] );

158
159
160 % print out
161
162 fprintf('Timestep: %f s\n', dt)
163 fprintf('Steps simulated: %i\n', steps)
164 fprintf('Simulation time: %f min\n', simulation_time_sim/60)
165 fprintf('Agents on ship on start: %i\n', agents_start)
166 fprintf('Agents on ship on simulation end: %i\n', agents_on_boat(end))
167 fprintf('Agents deleted due to NaN-positions: %i\n', deleted_agents)

168
169
170 fprintf('t_10: %f\n', t10)
171 fprintf('t_50: %f\n', t50)
172 fprintf('t_90: %f\n', t90)
173 fprintf('t_95: %f\n', t95)
174 fprintf('t_100: %f\n', t100)

```

Listing 15: plotFloor.m

---

```

1 function simulate(config_file)
2 % run this to start the simulation
3
4 % start recording the matlab output window for debugging reasons
5 diary log
6
7 if nargin==0
8     config_file='../../data/config1.conf';
9 end
10
11 fprintf('Load config file...\n');
12 config = loadConfig(config_file);
13
14 data = initialize(config);
15

```

```

    data.step = 1;
17 data.time = 0;
    fprintf('Start simulation..\n');
19
% tic until simulation end
21 simstart = tic;

23 %make video while simulation
if data.save_frames==1
25     vidObj=VideoWriter(data.video_file_name);
        open(vidObj);
27     end

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

37     % dump agents_per_floor to output
38     for floor=1:data.floor_count
            data.output.agents_per_floor(floor,data.step) =
                length(data.floor(floor).agents);
        end
41
% dump exit_left to output
43 data.output.exit_left(:,data.step) = data.exit_left';

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

47     % do the plotting
48     set(0,'CurrentFigure',data.figure_floors);
49     for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
            plotFloor(data, floor);
        end
53
54     if data.save_frames==1
55         % print('-depsc2',sprintf('frames/%s_%04i.eps', ...
56         % data.frame_basename,data.step), data.figure_floors);
57
%         make video while simulate
59         currFrame=getframe(data.figure_floors);
            writeVideo(vidObj,currFrame);
61
        end
63     set(0,'CurrentFigure',data.figure_exit);

```

```

65     plotExitedAgents(data);

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

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

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

80     % save output
81     output = data.output;
82     save(data.output_file_name, 'output')
83     fprintf('Frame %i done (took %.3fs; %.3fs out of %.3gs
84         simulated).\n', data.step, data.telapsed, data.time,
85             data.duration);
86
87     end

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

90     % update time
91     if (data.time + data.dt > data.duration)
92         data.dt = data.duration - data.time;
93         data.time = data.duration;
94     else
95         data.time = data.time + data.dt;
96     end

97 end

98 %make video while simulation
99 close(vidObj);

100 % toc of whole simulation
101 data.output.simulation_time = toc(simstart);

102 % save complete simulation
103 output = data.output;
104 save('output','output')
105 fprintf('Simulation done in %i seconds and saved data to output file.\n',
106         data.output.simulation_time);

```

```
111 % save diary  
113 diary
```

Listing 16: simulate.m

### 9.1.2 C code

```
1  
2 #include <mex.h>  
3 #include <string.h>  
4  
5 #include "tree_build.c"  
6 #include "tree_query.c"  
7 #include "tree_free.c"  
8  
9 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray  
*prhs[]){  
10     point_t *points;  
11     tree_t *tree;  
12     int m, n;  
13     uchar *data;  
14     int *root_index;  
15  
16     if (nlhs < 1)  
17         return;  
18  
19     points = (point_t*) mxGetPr(prhs[0]);  
20     m = mxGetM(prhs[0]);  
21     n = mxGetN(prhs[0]);  
22  
23     if (m != 2)  
24         mexErrMsgTxt("...");  
25  
26     tree = build_tree(points, n);  
27  
28     plhs[0] = mxCreateNumericMatrix(tree->first_free + sizeof(int), 1,  
29                                     mxUINT8_CLASS, mxREAL);  
30     data = (uchar*) mxGetPr(plhs[0]);  
31  
32     root_index = (int*) data;  
33     *root_index = tree->root_index;  
34     memcpy(data + sizeof(int), tree->data, tree->first_free);  
35  
36     free_tree(tree);  
37 }
```

Listing 17: createRangeTree.c

```

1 #include "mex.h"
3 #include <math.h>
5 #if defined __GNUC__ && defined __FAST_MATH__ && !defined __STRICT_ANSI__
#define MIN(i, j) fmin(i, j)
7 #define MAX(i, j) fmax(i, j)
#define ABS(i)      fabs(i)
9 #else
#define MIN(i, j) ((i) < (j) ? (i) : (j))
11 #define MAX(i, j) ((i) > (j) ? (i) : (j))
#define ABS(i)      ((i) < 0.0 ? -(i) : (i))
13 #endif
15
#define SOLVE_AND_UPDATE    udiff = uxmin - uymin; \
17     if (ABS(udiff) >= 1.0) \
{ \
19         up = MIN(uxmin, uymin) + 1.0; \
} \
21     else \
{ \
23         up = (uxmin + uymin + sqrt(2.0 - udiff * \
25             udiff)) / 2.0; \
27     up = MIN(uij, up); \
} \
err_loc = MAX(ABS(uij - up), err_loc); \
u[ij] = up;
29 #define I_STEP(_uxmin, _uymin, _st) if (boundary[ij] == 0.0) \
{ \
31         uij = un; \
33         un = u[ij + _st]; \
35         uxmin = _uxmin; \
37         uymin = _uymin; \
39         SOLVE_AND_UPDATE \
41         ij += _st; \
43     } \
45
47 #define I_STEP_UP(_uxmin, _uymin)    I_STEP(_uxmin, _uymin, 1)

```

```

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

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

55 #define UY_RIGHT u[ij + m]
#define UY_LEFT u[ij - m]
57 #define UY_BOTH MIN(UY_LEFT, UY_RIGHT)

59

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

67     m2 = m - 2;
68     n2 = n - 2;
69
70     *err = 0.0;
71     err_loc = 0.0;

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

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

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

85     /* i = 0->m-1, j = 1->n2 */
86     for (j = 1; j <= n2; ++j)
87     {
88         I_STEP_UP(UX_NEXT, UY_BOTH)

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

91         I_STEP_UP(UX_PREV, UY_BOTH)
92     }

93     /* i = 0, j = n-1 */

```

```

I_STEP_UP(UX_NEXT, UY_LEFT)

99
/* i = 1->m2, j = n-1 */
101 for (i = 1; i <= m2; ++i)
    I_STEP_UP(UX_BOTH, UY_LEFT)

103
/* i = m-1, j = n-1 */
105 I_STEP_UP(UX_PREV, UY_LEFT)

107
/* sweep 2 */
109 /* i = 0, j = n-1 */
111 ij = (n-1)*m;
112 un = u[ij];
113 I_STEP_UP(UX_NEXT, UY_LEFT)

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

117
/* i = m-1, j = n-1 */
119 I_STEP_UP(UX_PREV, UY_LEFT)

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

127
    for (i = 1; i <= m2; ++i)
        I_STEP_UP(UX_BOTH, UY_BOTH)

129
    I_STEP_UP(UX_PREV, UY_BOTH)
}

131
/* i = 0, j = 0 */
132 ij = 0;
133 un = u[ij];
134 I_STEP_UP(UX_NEXT, UY_RIGHT)

136
/* i = 1->m2, j = 0 */
137 for (i = 1; i <= m2; ++i)
    I_STEP_UP(UX_BOTH, UY_RIGHT)

139
/* i = m-1, j = 0 */
140 I_STEP_UP(UX_PREV, UY_RIGHT)

142
/* sweep 3 */
143 /* i = m-1, j = n-1 */

```

```

149    ij = m*n - 1;
150    un = u[ij];
151    I_STEP_DOWN(UX_NEXT, UY_LEFT)

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

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

159    /* i = m-1->0, j = n2->1 */
160    for (j = n2; j >= 1; --j)
161    {
162        I_STEP_DOWN(UX_NEXT, UY_BOTH)

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

167        I_STEP_DOWN(UX_PREV, UY_BOTH)
168    }

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

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

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

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

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

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

193    /* i = m-1->0, j = 1->n2 */
194    for (j = 1; j <= n2; ++j)
195    {
196        ij = m - 1 + j*m;
197        un = u[ij];

```

```

    I_STEP_DOWN(UX_NEXT, UY_BOTH)

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

203     I_STEP_DOWN(UX_PREV, UY_BOTH)
204 }

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

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

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

216 *err = MAX(*err, err_loc);
217 }

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

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

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

235

237     /* Get boundary */
238     if (!mxIsDouble(prhs[0]) || mxIsClass(prhs[0], "sparse"))
239         mexErrMsgTxt("Boundary field needs to be a full double precision
matrix.");
240

241     boundary = mxGetPr(prhs[0]);

```

```

247     m = mxGetM(prhs[0]);
248     n = mxGetN(prhs[0]);
249     entries = m * n;
250
251     /* Get max iterations */
252     if (nrhs >= 2)
253     {
254         if (!mxIsDouble(prhs[1]) || mxGetM(prhs[1]) != 1 ||
255             mxGetN(prhs[1]) != 1)
256             mexErrMsgTxt("Maximum iteration needs to be positive
257                         integer.");
258         max_iter = (int) *mxGetPr(prhs[1]);
259         if (max_iter <= 0)
260             mexErrMsgTxt("Maximum iteration needs to be positive
261                         integer.");
262     }
263     else
264         max_iter = 20;
265
266     /* Get tolerance */
267     if (nrhs >= 3)
268     {
269         if (!mxIsDouble(prhs[2]) || mxGetM(prhs[2]) != 1 ||
270             mxGetN(prhs[2]) != 1)
271             mexErrMsgTxt("Tolerance needs to be a positive real number.");
272         tol = *mxGetPr(prhs[2]);
273         if (tol < 0)
274             mexErrMsgTxt("Tolerance needs to be a positive real number.");
275     }
276     else
277         tol = 1e-12;
278
279     /* create and init output (distance) matrix */
280     plhs[0] = mxCreateDoubleMatrix(m, n, mxREAL);
281     u = mxGetPr(plhs[0]);
282
283     for (i = 0; i < entries; ++i)
284         u[i] = boundary[i] < 0.0 ? 0.0 : 1.0e10;
285
286     err = 0.0;
287     i = 0;
288     do
289     {
290         iteration(u, boundary, m, n, &err);
291         ++i;
292     } while (err > tol && i < max_iter);
293 }

```

Listing 18: fastSweeping.c

```

1 #include "mex.h"
3 #include <math.h>
5 #define INTERIOR(i, j) (boundary[(i) + m*(j)] == 0)
7 #define DIST(i, j) dist[(i) + m*(j)]
# define XGRAD(i, j) xgrad[(i) + m*(j)]
9 #define YGRAD(i, j) ygrad[(i) + m*(j)]
11 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
13     double *xgrad, *ygrad, *boundary, *dist;
14     double dxp, dyp, dym, xns, yns, nrm;
15     int m, n, i, j, nn;
17
/* Check number of outputs */
18     if (nlhs < 2)
19         mexErrMsgTxt("At least 2 output argument needed.");
20     else if (nlhs > 2)
21         mexErrMsgTxt("At most 2 output argument needed.");
23
/* Get inputs */
24     if (nrhs < 2)
25         mexErrMsgTxt("At least 2 input argument needed.");
26     else if (nrhs > 2)
27         mexErrMsgTxt("At most 2 input argument used.");
29
31
/* Get boundary */
32     if (!mxIsDouble(prhs[0]) || mxIsClass(prhs[0], "sparse"))
33         mexErrMsgTxt("Boundary field needs to be a full double precision
matrix.");
35     boundary = mxGetPr(prhs[0]);
36     m = mxGetM(prhs[0]);
37     n = mxGetN(prhs[0]);
39
/* Get distance field */
40     if (!mxIsDouble(prhs[1]) || mxIsClass(prhs[1], "sparse") ||
41         mxGetM(prhs[1]) != m || mxGetN(prhs[1]) != n)
42         mexErrMsgTxt("Distance field needs to be a full double precision
matrix with same dimension as the boundary.");
43
        dist = mxGetPr(prhs[1]);
44     m = mxGetM(prhs[1]);
45     n = mxGetN(prhs[1]);

```

```

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

53

55  for (j = 0; j < n; ++j)
56    for (i = 0; i < m; ++i)
57      if (INTERIOR(i,j))
58      {
59        if (i > 0)
60          dxm = INTERIOR(i-1,j) ? DIST(i-1,j) : DIST(i,j);
61        else
62          dxm = DIST(i,j);
63
64        if (i < m-1)
65          dyp = INTERIOR(i+1,j) ? DIST(i+1,j) : DIST(i,j);
66        else
67          dyp = DIST(i,j);
68
69        if (j > 0)
70          dyd = INTERIOR(i,j-1) ? DIST(i,j-1) : DIST(i,j);
71        else
72          dyd = DIST(i,j);
73
74        if (j < n-1)
75          dyp = INTERIOR(i,j+1) ? DIST(i,j+1) : DIST(i,j);
76        else
77          dyp = DIST(i,j);
78
79        XGRAD(i, j) = (dyp - dxm) / 2.0;
80        YGRAD(i, j) = (dyp - dyd) / 2.0;
81        nrm = sqrt(XGRAD(i, j)*XGRAD(i, j) + YGRAD(i, j)*YGRAD(i,
82                      j));
83        if (nrm > 1e-12)
84        {
85          XGRAD(i, j) /= nrm;
86          YGRAD(i, j) /= nrm;
87        }
88      else
89      {
90        XGRAD(i, j) = 0.0;
91        YGRAD(i, j) = 0.0;
92      }
93
94  for (j = 0; j < n; ++j)

```

```

95     for (i = 0; i < m; ++i)
96         if (!INTERIOR(i, j))
97     {
98         xns = 0.0;
99         yns = 0.0;
100        nn = 0;
101        if (i > 0 && INTERIOR(i-1, j))
102        {
103            xns += XGRAD(i-1, j);
104            yns += YGRAD(i-1, j);
105            ++nn;
106        }
107        if (i < m-1 && INTERIOR(i+1, j))
108        {
109            xns += XGRAD(i+1, j);
110            yns += YGRAD(i+1, j);
111            ++nn;
112        }
113        if (j > 0 && INTERIOR(i, j-1))
114        {
115            xns += XGRAD(i, j-1);
116            yns += YGRAD(i, j-1);
117            ++nn;
118        }
119        if (j < n-1 && INTERIOR(i, j+1))
120        {
121            xns += XGRAD(i, j+1);
122            yns += YGRAD(i, j+1);
123            ++nn;
124        }
125
126        if (nn > 0)
127        {
128            XGRAD(i, j) = xns / nn;
129            YGRAD(i, j) = yns / nn;
130        }
131    }

```

Listing 19: getNormalizedGradient.c

---

```

2 #include <mex.h>

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

```

```

10     if (nlhs < 1)
11         return;
12     else if (nlhs > 1)
13         mexErrMsgTxt("Exactly one output argument needed.");
14
15     if (nrhs != 3)
16         mexErrMsgTxt("Exactly three input arguments needed.");
17
18     m = mxGetM(prhs[0]);
19     n = mxGetN(prhs[0]);
20     data = mxGetPr(prhs[0]);
21     x = *mxGetPr(prhs[1]) - 1;
22     y = *mxGetPr(prhs[2]) - 1;
23
24     plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
25     out = mxGetPr(plhs[0]);
26
27     x = x < 0 ? 0 : x > m - 1 ? m - 1 : x;
28     y = y < 0 ? 0 : y > n - 1 ? n - 1 : y;
29     i0 = (int) x;
30     j0 = (int) y;
31     i1 = i0 + 1;
32     i1 = i1 > m - 1 ? m - 1 : i1;
33     j1 = j0 + 1;
34     j1 = j1 > n - 1 ? n - 1 : j1;
35
36     idx00 = i0 + m * j0;
37     d00 = data[idx00];
38     d01 = data[idx00 + m];
39     d10 = data[idx00 + 1];
40     d11 = data[idx00 + m + 1];
41
42     wx1 = x - i0;
43     wy1 = y - j0;
44     wx0 = 1.0 - wx1;
45     wy0 = 1.0 - wy1;
46
47     *out = wx0 * (wy0 * d00 + wy1 * d01) + wx1 * (wy0 * d10 + wy1 * d11);
48 }

```

Listing 20: lerp2.c

---

```

2 #include <mex.h>
3 #include <string.h>
4
5 #include "tree_build.c"
6 #include "tree_query.c"
7 #include "tree_free.c"

```

```

8 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
10 {
11     tree_t *tree;
12     int n, i;
13     int *point_idx, *root_idx;
14     range_t *range;
15     uchar *data;
16
17     if (nlhs != 1)
18         mexErrMsgTxt("...");

19     if (nrhs < 5)
20         mexErrMsgTxt("...");
21     else if (nrhs > 5)
22         mexErrMsgTxt("...");

23
24     data = (uchar*) mxGetPr(prhs[0]);
25
26     tree = (tree_t*) malloc(sizeof(tree_t));
27     tree->first_free = mxGetM(prhs[0]) - sizeof(int);
28     tree->total_size = tree->first_free;
29     root_idx = (int*) data;
30     tree->root_index = *root_idx;
31     tree->data = data + sizeof(int);

32
33     n = mxGetN(prhs[0]);
34     if (n != 1)
35         mexErrMsgTxt("...");

36
37     range = range_query(tree, *mxGetPr(prhs[1]), *mxGetPr(prhs[2]),
38                         *mxGetPr(prhs[3]), *mxGetPr(prhs[4]));

39
40     plhs[0] = mxCreateNumericMatrix(range->n, 1, mxUINT32_CLASS, mxREAL);
41     point_idx = (int*) mxGetPr(plhs[0]);
42
43     for (i = 0; i < range->n; ++i)
44         point_idx[i] = range->point_idx[i] + 1;

45
46     free_range(range);
47     free(tree);
48 }

```

Listing 21: rangeQuery.c

---

```

1 #ifndef TREE_H
2 #define TREE_H

3 #include "tree_types.h"

```

```

6 /* build a 2D range tree using the given points */
7 tree_t* build_tree(point_t *points, int n);
8
9 /* query a range tree */
10 range_t* range_query(tree_t *tree, double x_min, double x_max, double
11 y_min, double y_max);
12
13 /* free memory of a tree */
14 void free_tree(tree_t *tree);
15
16 /* free memory of a range */
17 void free_range(range_t *range);
18
19 #endif

```

Listing 22: tree.h

```

1 #ifndef TREE_BUILD_H
2 #define TREE_BUILD_H
3
4 #include "tree.h"
5
6 /* recursively build a subtree */
7 int build_subtree(tree_t *tree, double *x_vals, const int nx, point_t
8 *points, int *point_idx, const int np);
9
10 /* double comparison for qsort */
11 int compare_double(const void *a, const void *b);
12
13 /* index array sorting functions, sort point index array by point y
14 coordinates */
15 void index_sort_y(const point_t *points, int *point_idx, const int n);
16 void index_quicksort_y(const point_t *points, int *point_idx, int l, int
r);
17 int index_partition_y(const point_t *points, int *point_idx, int l, int r);
18
19 #endif

```

Listing 23: tree\_build.h

```

1
2 #include <assert.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 #include "tree_build.h"
8
9 tree_t* build_tree(point_t *points, int n)

```

```

{
11    int nx, i, j, *point_idx;
12    double *x_vals;
13    tree_t *tree;

15    /* get x coordinate values of all points */
16    x_vals = (double*) malloc(n * sizeof(double));
17    for (i = 0; i < n; ++i)
18        x_vals[i] = points[i].x;

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

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

29    /* remove duplicates */
30    j = 0;
31    for (i = 0; i < nx; ++i)
32    {
33        x_vals[i] = x_vals[j];
34        while (x_vals[i] == x_vals[j])
35            ++j;
36    }

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

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

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

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

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

57    /* recursively build tree */
58    build_subtree(tree, x_vals, nx, points, point_idx, n);
59}

```

```

61     /* free temporaries */
62     free(x_vals);
63     return tree;
64 }
65
66 int build_subtree(tree_t *tree, double *x_vals, const int nx, point_t
67 *points, int *point_idx, const int np)
68 {
69     int i, j, k, nx_left, np_left, node_size, right_idx;
70     node_t *node;
71     int *node_point_idx, *point_idx_left, *point_idx_right, node_idx;
72     uchar *new_data;
73
74     assert(nx > 0);
75     assert(np > 0);
76
77     /* allocate memory in the tree data structure */
78     node_size = sizeof(node_t) + np * sizeof(int);
79     while (tree->first_free + node_size > tree->total_size)
80     {
81         tree->total_size <= 1;
82         new_data = (uchar*) malloc(tree->total_size * sizeof(uchar));
83         for (i = 0; i < tree->first_free; ++i)
84             new_data[i] = tree->data[i];
85         free(tree->data);
86         tree->data = new_data;
87     }
88     node_idx = tree->first_free;
89     node = (node_t*) &tree->data[node_idx];
90     tree->first_free += node_size;
91
92     /* set number of stored points */
93     node->np = np;
94     node_point_idx = (int*) (node + 1);
95
96     /* copy point indices to node */
97     memcpy(node_point_idx, point_idx, np * sizeof(int));
98
99     /* create child node if there is only one x value left, otherwise
100      create interior node */
101    if (nx == 1)
102    {
103        node->right_idx = -1;
104        node->x_val = x_vals[0];
105    }
106    else
107    {
108        /* get median of x values */

```

```

109     nx_left = nx >> 1;
110     node->x_val = x_vals[nx_left - 1];

111     /* count points belonging to the left child */
112     np_left = 0;
113     for (i = 0; i < np; ++i)
114     {
115         if (points[point_idx[i]].x <= node->x_val)
116             ++np_left;
117     }

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

121     /* fill index arrays */
122     j = 0;
123     k = 0;
124     for (i = 0; i < np; ++i)
125     {
126         if (points[point_idx[i]].x <= node->x_val)
127             point_idx_left[j++] = point_idx[i];
128         else
129             point_idx_right[k++] = point_idx[i];
130     }

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

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

136     /* build right subtree and get its root node index */
137     right_idx = build_subtree(tree, x_vals + nx_left, nx - nx_left,
138                               points, point_idx_right, np - np_left);
139     /* update node pointer (could have changed during build_subtree,
140      because of data allocation) */
141     node = (node_t*) &tree->data[node_idx];
142     /* update node's right child index */
143     node->right_idx = right_idx;
144 }

145     /* return node index to parent */
146     return node_idx;
147 }
148 }

149 int compare_double(const void *a, const void *b)
150 {
151     double ad, bd;

```

```

155     ad = *((double*) a);
156     bd = *((double*) b);
157     return (ad < bd) ? -1 : (ad > bd) ? 1 : 0;
158 }
159
160 void index_sort_y(const point_t *points, int *point_idx, const int n)
161 {
162     index_quicksort_y(points, point_idx, 0, n - 1);
163 }
164
165 void index_quicksort_y(const point_t *points, int *point_idx, int l, int r)
166 {
167     int p;
168
169     /* quicksort point indices by point y coordinates, don't touch point
170      array itself */
171     while (l < r)
172     {
173         p = index_partition_y(points, point_idx, l, r);
174         if (r - p > p - l)
175         {
176             index_quicksort_y(points, point_idx, l, p - 1);
177             l = p + 1;
178         }
179         else
180         {
181             index_quicksort_y(points, point_idx, p + 1, r);
182             r = p - 1;
183         }
184     }
185 }
186
187 int index_partition_y(const point_t *points, int *point_idx, int l, int r)
188 {
189     int i, j, tmp;
190     double pivot;
191
192     /* rightmost element is pivot */
193     i = l;
194     j = r - 1;
195     pivot = points[point_idx[r]].y;
196
197     /* quicksort partition */
198     do
199     {
200         while (points[point_idx[i]].y <= pivot && i < r)
201             ++i;
202
203         while (points[point_idx[j]].y >= pivot && j > l)
204             --j;

```

```

205     if (i < j)
206     {
207         tmp = point_idx[i];
208         point_idx[i] = point_idx[j];
209         point_idx[j] = tmp;
210     }
211 } while (i < j);

213 if (points[point_idx[i]].y > pivot)
214 {
215     tmp = point_idx[i];
216     point_idx[i] = point_idx[r];
217     point_idx[r] = tmp;
218 }
219
220     return i;
221 }
```

Listing 24: tree\_build.c

```

1 #include <stdlib.h>
3
5 #include "tree.h"
7
9 void free_tree(tree_t *tree)
10 {
11     free(tree->data);
12 }

14 void free_range(range_t *range)
15 {
16     free(range->point_idx);
17 }
```

Listing 25: tree\_free.c

```

1 #ifndef TREE_QUERY_H
2 #define TREE_QUERY_H

4 #include "tree_types.h"

6 /* appends a point-index to a range, increases range capacity if needed */
7 void range_append(range_t *range, int idx);
8
9 /* finds the split node of a given query */
10 int find_split_node(tree_t *tree, int node_idx, range_t *range);

12 /* 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);
14
#endif

```

Listing 26: tree\_query.h

```

1
2 #include <assert.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include "tree_query.h"
7
8 #define LEFT_CHILD_IDX(node_idx, node) (node_idx) + sizeof(node_t) +
9     (node)->np * sizeof(int)
10 #define RIGHT_CHILD_IDX(node_idx, node) (node)->right_idx
11 #define NODE_FROM_IDX(tree, node_idx) (node_t*) &(tree)->data[node_idx];
12
13 range_t* range_query(tree_t *tree, double x_min, double x_max, double
14     y_min, double y_max)
15 {
16     int split_node_idx, node_idx;
17     node_t *split_node, *node;
18     range_t *range;
19
20     /* init range */
21     range = (range_t*) malloc(sizeof(range_t));
22     range->min.x = x_min;
23     range->max.x = x_max;
24     range->min.y = y_min;
25     range->max.y = y_max;
26     range->n = 0;
27     range->total_size = 16;
28     range->point_idx = (int*) malloc(range->total_size * sizeof(int));
29
30     /* find split node */
31     split_node_idx = find_split_node(tree, tree->root_index, range);
32     split_node = NODE_FROM_IDX(tree, split_node_idx);
33
34     /* if split node is a child */
35     if (split_node->right_idx == -1)
36     {
37         range_query_y(tree, split_node_idx, range);
38         return range;
39     }
40
41     /* follow left path of the split node */
42     node_idx = LEFT_CHILD_IDX(split_node_idx, split_node);
43     node = NODE_FROM_IDX(tree, node_idx);
44     while (node->right_idx != -1)
45 
```

```

43     {
44         if (range->min.x <= node->x_val)
45         {
46             range_query_y(tree, RIGHT_CHILD_IDX(node_idx, node), range);
47             node_idx = LEFT_CHILD_IDX(node_idx, node);
48         }
49         else
50             node_idx = RIGHT_CHILD_IDX(node_idx, node);
51         node = NODE_FROM_IDX(tree, node_idx);
52     }
53     range_query_y(tree, node_idx, range);

55     /* follow right path of the split node */
56     node_idx = split_node->right_idx;
57     node = NODE_FROM_IDX(tree, node_idx);
58     while (node->right_idx != -1)
59     {
60         if (range->max.x > node->x_val)
61         {
62             range_query_y(tree, LEFT_CHILD_IDX(node_idx, node), range);
63             node_idx = RIGHT_CHILD_IDX(node_idx, node);
64         }
65         else
66             node_idx = LEFT_CHILD_IDX(node_idx, node);
67         node = NODE_FROM_IDX(tree, node_idx);
68     }
69     range_query_y(tree, node_idx, range);

70     return range;
71 }
72
73 void range_append(range_t *range, int idx)
74 {
75     int *new_point_idx;
76     int new_size, i;

77     /* just append if there is enough place, otherwise double capacity and
78      * append */
79     if (range->n < range->total_size)
80         range->point_idx[range->n++] = idx;
81     else
82     {
83         new_size = range->total_size << 1;
84         new_point_idx = (int*) malloc(new_size * sizeof(int));
85         for (i = 0; i < range->n; ++i)
86             new_point_idx[i] = range->point_idx[i];
87         new_point_idx[range->n++] = idx;
88         free(range->point_idx);
89         range->point_idx = new_point_idx;
90         range->total_size = new_size;
91     }

```

```

        }
93 }

95 int find_split_node(tree_t *tree, int node_idx, range_t *range)
{
97     node_t *node;

99     node = (node_t*) &tree->data[node_idx];
100    /* check if this node is the split node */
101    if (range->min.x <= node->x_val && range->max.x > node->x_val)
102        return node_idx;
103
104    /* ...or if it is a child (and therefor the split node) */
105    if (node->right_idx == -1)
106        return node_idx;
107
108    /* otherwise search the split node at the left or right of the current
       node */
109    if (range->max.x <= node->x_val)
110        return find_split_node(tree, LEFT_CHILD_IDX(node_idx, node),
111                               range);
112    else
113        return find_split_node(tree, RIGHT_CHILD_IDX(node_idx, node),
114                               range);
114 }

115 void range_query_y(tree_t *tree, int node_idx, range_t *range)
{
116
117     point_t *points;
118     double y;
119     int i, j, k, m, start, end;
120     int *point_idx;
121     node_t *node;

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

125
126    /* return if all points are outside the range */
127    if (points[point_idx[0]].y > range->max.y || points[point_idx[node->np
128      - 1]].y < range->min.y)
129        return;

130
131    /* binary search for lower end of the range */
132    y = range->min.y;
133    j = 0;
134    k = node->np - 1;
135    while (j != k)
136    {
137        m = (j + k) / 2;

```

```

139         if (points[point_idx[m]].y >= y)
140             k = m;
141         else
142             j = m + 1;
143     }
144     start = j;
145
146     /* binary search for higher end of the range */
147     y = range->max.y;
148     j = 0;
149     k = node->np - 1;
150     while (j != k)
151     {
152         m = (j + k + 1) / 2;
153         if (points[point_idx[m]].y > y)
154             k = m - 1;
155         else
156             j = m;
157     }
158     end = j;
159
160     /* append found points to the range */
161     for (i = start; i <= end; ++i)
162         if (points[point_idx[i]].x <= range->max.x)
163             range_append(range, point_idx[i]);
164
165 }
```

Listing 27: tree\_query.c

---

```

1 #ifndef TREE_TYPES_H
2 #define TREE_TYPES_H
3
4 typedef unsigned char uchar;
5
6 /* 2D point */
7 typedef struct
8 {
9     double x;
10    double y;
11 } point_t;
12
13 /* tree */
14 typedef struct
15 {
16     /* byte data array with points and nodes */
17     uchar *data;
18
19     /* index of first unused byte */
20     int first_free;
21 }
```

```

22     /* total number of allocated bytes */
23     int total_size;
24
25     /* index of the root node in the data array*/
26     int root_index;
27 } tree_t;
28
29 /* node */
30 typedef struct
31 {
32     /* index of the right child node (left child follows directly after
33         current) */
34     int right_idx;
35
36     /* number of associated points */
37     int np;
38
39     /* associated x-coordinate value */
40     double x_val;
41 } node_t;
42
43 /* range */
44 typedef struct
45 {
46     /* point index list */
47     int *point_idx;
48
49     /* number of saved indices */
50     int n;
51
52     /* total number of allocated indices */
53     int total_size;
54
55     /* minimum range point */
56     point_t min;
57
58     /* maximum range point */
59     point_t max;
60 } range_t;
61
62 #endif

```

Listing 28: tree\_types.h

### 9.1.3 crew command code

---

```

1 function data = addAgentRepulsiveForce(data)
%ADDAGENTREPULSIVEFORCE Summary of this function goes here

```

```

3 % Detailed explanation goes here

5 % Obstruction effects in case of physical interaction

7 % get maximum agent distance for which we calculate force
r_max = data.r_influence;
9 tree = 0;

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

15     % update range tree of lower floor
    tree_lower = tree;

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

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

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

29         % use range tree to get the indices of all agents near agent ai
31         idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
                           pi(2) - r_max, pi(2) + r_max)';
33

35         % loop over agents near agent ai
            for aj = idx

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

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

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

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

```

```

53
55         % sum of radii
rij = (ri + rj);

57         % 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

65 F = (data.A * exp((rij - d)/data.B) + T1)*nij + T2;
67
69     data.floor(fi).agents(ai).f = ...
       data.floor(fi).agents(ai).f + F;
71     data.floor(fi).agents(aj).f = ...
       data.floor(fi).agents(aj).f - F;
73 end

75 % 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)';
81
83     % 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)))
87
89         vj = data.floor(fi-1).agents(aj).v;
         rj = data.floor(fi-1).agents(aj).r;

91         % vector pointing from j to i
nij = (pi - pj) * data.meter_per_pixel;
93
95         % distance of agents
d = norm(nij);

97         % normalized vector pointing from j to i
nij = nij / d;
% tangential direction
tij = [-nij(2), nij(1)];
101

```

```

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

105             % repulsive interaction forces
106             if d < rij
107                 T1 = data.k*(rij - d);
108                 T2 = data.kappa*(rij - d)*dot((vj -
109                                         vi),tij)*tij;
110             else
111                 T1 = 0;
112                 T2 = 0;
113             end
114
115             F = (data.A * exp((rij - d)/data.B) + T1)*nij
116             + T2;
117
118             data.floor(fi).agents(ai).f = ...
119             data.floor(fi).agents(ai).f + F;
120             data.floor(fi-1).agents(aj).f = ...
121             data.floor(fi-1).agents(aj).f - F;
122         end
123     end
124 end
125
126 end

```

Listing 29: addAgentRepulsiveForce.m

```

1 function data = addDesiredForce(data)
%ADDDESIREDFORCE add 'desired' force contribution (towards nearest exit or
3 %staircase)

5 for fi = 1:data.floor_count

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

9         % get agent's data
10        p = data.floor(fi).agents(ai).p;
11        m = data.floor(fi).agents(ai).m;
12        v0 = data.floor(fi).agents(ai).v0;
13        v = data.floor(fi).agents(ai).v;

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

```

```

21      % get force
22      Fi = m * (v0*e - v)/data.tau;
23
24      % add force
25      data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
26  end
27 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

```

1    function data = applyForcesAndMove(data)
2 %APPLYFORCESANDMOVE apply current forces to agents and move them using
%the timestep and current velocity
3
4    n_velocity_clamps = 0;
5
6    % loop over all floors higher than exit floor
7    for fi = data.floor_exit:data.floor_count
8
9        % init logical arrays to indicate agents that change the floor or exit
% the simulation
10       floorchange = false(length(data.floor(fi).agents),1);
11       exited = false(length(data.floor(fi).agents),1);
12
13       % loop over all agents
14       for ai=1:length(data.floor(fi).agents)
15           % add current force contributions to velocity
16           v = data.floor(fi).agents(ai).v + data.dt * ...
17               data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
18
19           % clamp velocity
20           if norm(v) > data.v_max
21               v = v / norm(v) * data.v_max;
22               n_velocity_clamps = n_velocity_clamps + 1;
23           end
24
25           % get agent's new position
26           newp = data.floor(fi).agents(ai).p + ...
27               v * data.dt / data.meter_per_pixel;
28
29           % if the new position is inside a wall, remove perpendicular
% component of the agent's velocity
30           if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
31               data.floor(fi).agents(ai).r
32
33               % get agent's position
34               p = data.floor(fi).agents(ai).p;
35
36               % get wall distance gradient (which is off course perpendicular
% to the nearest wall)
37               nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
38               ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
39               n = [nx ny];
40
41               % project out perpendicular component of velocity vector
42               v = v - dot(n,v)/dot(n,n)*n;
43
44               % get agent's new position
45               newp = data.floor(fi).agents(ai).p + ...
46
47
48

```

```

50             v * data.dt / data.meter_per_pixel;
end

52
% check if agents position is ok
54 % repositioning after 50 times clogging
% deleting if agent has a NaN position
56 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];
58        data.floor(fi).agents(ai).clogged =
            data.floor(fi).agents(ai).clogged + 1;
59        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))
60    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))

62
% get agent's new position
63 newp = data.floor(fi).agents(ai).p + ...
64     v * data.dt / data.meter_per_pixel;
65 if isnan(newp)
    % get rid of disturbing agent
66    fprintf('WARNING: position of an agent is NaN!
Deleted this agent.\n')
67    exited(ai) = 1;
68    data.agents_exited = data.agents_exited +1;
69    data.output.deleted_agents=data.output.deleted_agents+1;
70    newp = [1 1];
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
% update agent's velocity and position

```

```

94     data.floor(fi).agents(ai).v = v;
95     data.floor(fi).agents(ai).p = newp;

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

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

102    % check if agent reached an exit
103    if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
104       exited(ai) = 1;
105       data.agents_exited = data.agents_exited +1;
106   %

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

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

111   %

112   fprintf(int2str(data.current_exit));

113   %

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

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

120       %

121       %change current exit to wall
122       data.floor(data.floor_exit).img_wall =
123           data.floor(data.floor_exit).img_wall == 1 ...
124               | (data.exit_nr == (data.current_exit));
125       data.floor(data.floor_exit).img_exit =
126           data.floor(data.floor_exit).img_exit == 1 ...
127               & (data.exit_nr ~= (data.current_exit));

128       %

129       %redo initEscapeRoutes and initWallForces with new exit
130       % and wall parameters
131       data = initEscapeRoutes(data);
132       data = initWallForces(data);

133       %

134       end
135   end
136 end

```

```

138     % add appropriate agents to next lower floor
139     if fi > data.floor_exit
140         data.floor(fi-1).agents = [data.floor(fi-1).agents
141             data.floor(fi).agents(floorchange)];
142     end
143
144     % delete these and exited agents
145     data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
146 end
147
148
149
150 % loop over all floors lower than exit floor
151 for fi = 1:data.floor_exit
152
153     % init logical arrays to indicate agents that change the floor or exit
154     % the simulation
155     floorchange = false(length(data.floor(fi).agents),1);
156     exited = false(length(data.floor(fi).agents),1);
157
158     % loop over all agents
159     for ai=1:length(data.floor(fi).agents)
160         % add current force contributions to velocity
161         v = data.floor(fi).agents(ai).v + data.dt * ...
162             data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
163
164         % clamp velocity
165         if norm(v) > data.v_max
166             v = v / norm(v) * data.v_max;
167             n_velocity_clamps = n_velocity_clamps + 1;
168         end
169
170         % get agent's new position
171         newp = data.floor(fi).agents(ai).p + ...
172             v * data.dt / data.meter_per_pixel;
173
174         % if the new position is inside a wall, remove perpendicular
175         % component of the agent's velocity
176         if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
177             data.floor(fi).agents(ai).r
178
179             % get agent's position
180             p = data.floor(fi).agents(ai).p;
181
182             % get wall distance gradient (which is of course perpendicular
183             % to the nearest wall)
184             nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));

```

```

186     ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
187     n = [nx ny];
188
189     % project out perpendicular component of velocity vector
190     v = v - dot(n,v)/dot(n,n)*n;
191
192     % get agent's new position
193     newp = data.floor(fi).agents(ai).p + ...
194         v * data.dt / data.meter_per_pixel;
195 end
196
197
198     % check if agents position is ok
199     % repositioning after 50 times clogging
200     % deleting if agent has a NaN position
201 if ~isnan(newp)
202     if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
203         newp = data.floor(fi).agents(ai).p;
204         v = [0 0];
205         data.floor(fi).agents(ai).clogged =
206             data.floor(fi).agents(ai).clogged + 1;
207         fprintf('WARNING: clogging agent %i on floor %i (%i).
208             Position
209             (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
210     if data.floor(fi).agents(ai).clogged >= 40
211         nx = rand(1)*2 - 1;
212         ny = rand(1)*2 - 1;
213         n = [nx ny];
214         v = n*data.v_max/2;
215         fprintf('WARNING: agent %i on floor %i velocity set
216             random to get out of wall. Position
217             (%f,%f).\n',ai,fi,newp(1),newp(2))
218
219     % get agent's new position
220     newp = data.floor(fi).agents(ai).p + ...
221         v * data.dt / data.meter_per_pixel;
222     if isnan(newp)
223         % get rid of disturbing agent
224         fprintf('WARNING: position of an agent is NaN!
225             Deleted this agent.\n')
226         exited(ai) = 1;
227         data.agents_exited = data.agents_exited +1;
228         data.output.deleted_agents=data.output.deleted_agents+1;
229         newp = [1 1];
230     end
231 end
232 end
233 else
234     % get rid of disturbing agent

```

```

        fprintf('WARNING: position of an agent is NaN! Deleted this
                agent.\n')
230    exited(ai) = 1;
232    data.agents_exited = data.agents_exited +1;
233    data.output.deleted_agents=data.output.deleted_agents+1;
234    newp = [1 1];
235
236    % update agent's velocity and position
237    data.floor(fi).agents(ai).v = v;
238    data.floor(fi).agents(ai).p = newp;
239
240    % reset forces for next timestep
241    data.floor(fi).agents(ai).f = [0 0];
242
243    % check if agent reached a staircase up and indicate floor change
244    if data.floor(fi).img_stairs_up(round(newp(1)), round(newp(2)))
245        floorchange(ai) = 1;
246    end
247
248    % check if agent reached an exit
249    if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
250        exited(ai) = 1;
251        data.agents_exited = data.agents_exited +1;
252
253    %
254        fprintf('agent exited from lower loop\n');
255
256        %save current exit nr
257        data.current_exit = data.exit_nr(round(newp(1)),
258                                         round(newp(2)));
259
260        %update exit_left
261        data.exit_left(1,data.current_exit) =
262            data.exit_left(1,data.exit_nr(round(newp(1)),
263                           round(newp(2)))) - 1;
264
265            %close exit if there is no more free space
266            if data.exit_left(1,data.current_exit) < 1
267
268                %change current exit to wall
269                data.floor(data.floor_exit).img_wall =
270                    data.floor(data.floor_exit).img_wall == 1 ...
271                        | (data.exit_nr == (data.current_exit));
272                data.floor(data.floor_exit).img_exit =
273                    data.floor(data.floor_exit).img_exit == 1 ...
274                        & (data.exit_nr ~= (data.current_exit));
275
276                %redo initEscapeRoutes and initWallForces with new exit
277                % and wall parameters
278                data = initEscapeRoutes(data);

```

```

272         data = initWallForces(data);

274 %           fprintf('new routes from lower loop\n');

276     end

278   end
end

280 % add appropriate agents to next lower floor
282 if fi < data.floor_exit
    data.floor(fi+1).agents = [data.floor(fi+1).agents ...
                                data.floor(fi).agents(floorchange)];
end

286 % delete these and exited agents
288 data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
end

290 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)
298 end

300 % if n_velocity_clamps > 0
%     fprintf(['WARNING: clamped velocity of %d agents, ' ...
302 %             'possible simulation instability.\n'], n_velocity_clamps);
% end

```

Listing 32: applyForcesAndMove.m

---

```

1 function val = checkForIntersection(data, floor_idx, agent_idx)
% check an agent for an intersection with another agent or a wall
3 % the check is kept as simple as possible
%
5 % arguments:
%   data          global data structure
7 %   floor_idx    which floor to check
%   agent_idx    which agent on that floor
9 %   agent_new_pos vector: [x,y], desired agent position to check
%
11 % return:

```

```

% 0           for no intersection
13 % 1           has an intersection with wall
% 2           with another agent
15
val = 0;
17
p = data.floor(floor_idx).agents(agent_idx).p;
19 r = data.floor(floor_idx).agents(agent_idx).r;

21 % 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
            ...
25             <= r + data.floor(floor_idx).agents(i).r
        val=2;
27         return;
    end
29 end
end
31

33 % check for wall intersection
if lerp2(data.floor(floor_idx).img_wall_dist, p(1), p(2)) < r
35     val = 1;
end

```

Listing 33: checkForIntersection.m

---

```

1 mex 'fastSweeping.c'
mex 'getNormalizedGradient.c'
3 mex 'lerp2.c'
mex 'createRangeTree.c'
5 mex 'rangeQuery.c'

```

---

Listing 34: compileC.m

---

```

1 function data = initAgents(data)

3 % place agents randomly in desired spots, without overlapping
5

7 function radius = getAgentRadius()
    %radius of an agent in meters
9     radius = data.r_min + (data.r_max-data.r_min)*rand();
end
11
11 data.agents_exited = 0; %how many agents have reached the exit
13 data.total_agent_count = 0;

```

---

```

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

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

25 % init agent
27 data.floor(i).agents(cur_agent).r = getAgentRadius();
data.floor(i).agents(cur_agent).v = [0, 0];
29 data.floor(i).agents(cur_agent).f = [0, 0];
data.floor(i).agents(cur_agent).m = data.m;
31 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

33 tries = 10;
35 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;
39 end
41 if tries > -1
    %remove the last agent
    data.floor(i).agents = data.floor(i).agents(1:end-1);
43 end
45 end
47 end
49 data.total_agent_count = data.total_agent_count +
    length(data.floor(i).agents);

51 if length(data.floor(i).agents) ~= agent_count
    fprintf(['WARNING: could only place %d agents on floor %d , ...
53 'instead of the desired %d.\n'], ...
    length(data.floor(i).agents), i, agent_count);
55 end
57 end
59 if floors_with_agents==0
    error('no spots to place agents!');
end
61

```

```
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);
22 fprintf('Init agents...\n');
    data = initAgents(data);
24
% maximum influence of agents on each other
26
data.r_influence = data.pixel_per_meter * ...
28     fzero(@(r) data.A * exp((2*data.r_max-r)/data.B) - 1e-4, data.r_max);

30 fprintf('Init plots...\n');
    %init the plots
32 %exit plot
    data.figure_exit=figure;
34 hold on;
    axis([0 data.duration 0 data.total_agent_count]);
36 title(sprintf('agents that reached the exit (total agents: %i)', ...
    data.total_agent_count));

38 % floors plot
    data.figure_floors=figure;
40 % figure('units','normalized','outerposition',[0 0 1 1])
    data.figure_floors_subplots_w = data.floor_count;
42 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
52 end

54 % init output matrizes
    data.output = struct;
56 data.output.config = config;
    data.output.agents_per_floor =
        ones(data.floor_count,data.duration/data.dt).*(-1);

```

```

58 data.output.exit_left = zeros(data.exit_count,data.duration/data.dt);

60 % prepare output file name
61 data.output_file_name = [ 'output_' data.frame_basename];
62
63 % prepare video file name
64 data.video_file_name = [ 'video_' data.frame_basename '.avi'];

66 % set deleted_agents to zero
67 data.output.deleted_agents = 0;

```

Listing 37: initialize.m

```

1 function data = initWallForces(data)
2 %INITWALLFORCES init wall distance maps and gradient maps for each floor

4 for i=1:data.floor_count

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

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

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

```

Listing 38: initWallForces.m

```

1 function config = loadConfig(config_file)
2 % load the configuration file
3 %
4 % arguments:
5 % config_file      string, which configuration file to load
6 %
7

9 % get the path from the config file -> to read the images
10 config_path = fileparts(config_file);
11 if strcmp(config_path, '') == 1
12     config_path = '.';
13 end

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

```

```

17 fclose(fid);

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

21 %convert numerical values from string to double
23 v = str2double(values);
idx = ~isnan(v);
25 values(idx) = num2cell(v(idx));

27 config = cell2struct(values, keynames);

29
% read the images
31 for i=1:config.floor_count

33 %building structure
file = config.(sprintf('floor_%d_build', i));
35 file_name = [config_path '/' file];
img_build = imread(file_name);

37 % decode images
39 config.floor(i).img_wall = (img_build(:, :, 1) == 0 ...
& img_build(:, :, 2) == 0 ...
& img_build(:, :, 3) == 0);

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

45

47 %second possibility:
%pixel is exit if 1-->0, 3-->0, and if 2 is between 255 and 230 or if no
49 %red or blue

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

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

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

59

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

63

65 if i == config.floor_exit

```

```

67      %make the exit_nr matrix where the number of exit is indicated in
       each
       %pixel
69
71      %make a zeroes matrix as big as img_exit
config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));

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

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

79          %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 )
    ;

81
83          %build the exit_left matrix
config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));

85      end
87
89      %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];
95 end

97

99 % build open_second matrix
100 for i=1:config.open_second_nr
101     config.open_second(i)=config.(sprintf('open_second_%i', i));
102 end

103 % save the "all exits open" configuration
104 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;

107 % replace the open_second exits in img_exit with a wall
108 for i=1:config.open_second_nr
109

```

```

    config.floor(config.floor_exit).img_wall(find(config.exit_nr ==
    config.open_second(i))) = 1;
111 config.floor(config.floor_exit).img_exit(find(config.exit_nr ==
    config.open_second(i))) = 0;
end
113 % set the boolean to check if switch from first to second mode has already
115 % been executed
config.switch_done = 0;

```

Listing 39: loadConfig.m

```

function plotAgentsPerFloor(data, floor_idx)
2 %plot time vs agents on floor

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

6 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]);
8
10 if floor_idx~=data.floor_count
    set(h,'ytick',[]) %hide y-axis label
end
12 axis([0 data.time+data.dt 0 data.agents_per_floor*2]);
14 %axis([0 data.duration 0 data.agents_per_floor*2]);
16
18 hold on;
18 plot(data.time, length(data.floor(floor_idx).agents), 'b-');
hold off;
20 title(sprintf('%i', floor_idx));

```

Listing 40: plotAgentsPerFloor.m

```

function plotExitedAgents(data)
2 %plot time vs exited agents

4 hold on;
5 plot(data.time, data.agents_exited, 'r-');
6 hold off;

```

Listing 41: plotExitedAgents.m

```

function plotFloor(data, floor_idx)
2
if floor_idx == data.floor_exit-1 || floor_idx == data.floor_exit ||
    floor_idx == data.floor_exit+1

```

```

4     h=subplot(data.floor(floor_idx).building_plot);
5
6 set(h,
7     'position',[0,0.35+0.65/3*(floor_idx-data.floor_exit+1),1,0.65/3-0.005]);
8
9 hold off;
10 % the building image
11 imagesc(data.floor(floor_idx).img_plot);
12 hold on;
13
14 %plot options
15 colormap(data.color_map);
16 axis equal;
17 axis manual; %do not change axis on window resize
18
19 set(h, 'Visible', 'off')
20 % title(sprintf('floor %i', floor_idx))
21
22 % plot agents
23 if ~isempty(data.floor(floor_idx).agents)
24     ang = [linspace(0,2*pi, 10) nan]';
25     rmul = [cos(ang) sin(ang)] * data.pixel_per_meter;
26     draw = cell2mat(arrayfun(@(a) repmat(a.p,length(ang),1) + a.r*rmul, ...
27         data.floor(floor_idx).agents, 'UniformOutput', false));
28     line(draw(:,2), draw(:,1), 'Color', 'r');
29 end
30
31 hold off;
32 end

```

Listing 42: plotFloor.m

```

1 function simulate(config_file)
2 % run this to start the simulation
3
4 % start recording the matlab output window for debugging reasons
5 diary log
6
7 if nargin==0
8     config_file='../../data/config1.conf';
9 end
10
11 fprintf('Load config file...\n');
12 config = loadConfig(config_file);
13
14 data = initialize(config);
15
16 data.step = 1;
17 data.time = 0;

```

```

18 fprintf('Start simulation...\n');

20 % tic until simulation end
simstart = tic;
22
%make video while simulation
24 if data.save_frames==1
    vidObj=VideoWriter(data.video_file_name);
26     open(vidObj);
    end
28
while (data.time < data.duration)
    % tic until timestep end
    tstart=tic;
32     data = addDesiredForce(data);
    data = addWallForce(data);
34     data = addAgentRepulsiveForce(data);
    data = applyForcesAndMove(data);
36
    % dump agents_per_floor to output
38     for floor=1:data.floor_count
        data.output.agents_per_floor(floor,data.step) =
            length(data.floor(floor).agents);
    end
42
    % dump exit_left to output
    data.output.exit_left(:,data.step) = data.exit_left';
44
    if mod(data.step,data.save_step) == 0
46
        % do the plotting
48        set(0,'CurrentFigure',data.figure_floors);
        for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
            plotFloor(data, floor);
        end
52
        if data.save_frames==1
            %
                print('-depsc2',sprintf('frames/%s_%04i.eps', ...
            data.frame_basename,data.step), data.figure_floors);
56 %

58 %      make video while simulate
59     currFrame=getframe(data.figure_floors);
60     writeVideo(vidObj,currFrame);
62
    end
64
    set(0,'CurrentFigure',data.figure_exit);
    plotExitedAgents(data);
66

```

```

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

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

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

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

86 end

87 % update step
88 data.step = data.step+1;

89 % update time
90 if (data.time + data.dt > data.duration)
91     data.dt = data.duration - data.time;
92     data.time = data.duration;
93 else
94     data.time = data.time + data.dt;
95 end

96 end

97 %make video while simulation
98 close(vidObj);

99 % toc of whole simulation
100 data.output.simulation_time = toc(simstart);

101 % save complete simulation
102 output = data.output;
103 save('output','output')
104 fprintf('Simulation done in %i seconds and saved data to output file.\n',
105     data.output.simulation_time);

106 % save diary

```

`diary`

---

Listing 43: simulate.m