



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 13, 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

Balietti

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). 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

2. Brechbuehler
A. Reber
F. Schmid
M. 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

1	Abstract	6
2	Individual contributions	6
3	Introduction and Motivations	7
3.1	Introduction	7
3.2	Motivation	7
3.3	Fundamental Questions	8
4	Description of the model	9
4.1	Social force model	9
4.2	Ship structure	10
5	Implementation	11
5.1	Basic code	11
5.2	Code adjustments	11
5.2.1	Exits	11
5.2.2	Different exits	12
5.2.3	Closing exits during simulation	13
5.2.4	Controlled evacuation	15
5.3	Video creation	17
5.4	System output	18
5.5	Postprocessing	20
5.6	Parameter input and config file	20
5.7	Ship Decks	20
6	Simulation Results and Discussion	24
6.1	Expected Results	24
6.2	Simulation Results	24
6.2.1	Standard ship	24
6.2.2	Modified room disposition	25
6.2.3	Modified rescueboat size	26
6.2.4	Crew command	27
6.3	Comparison	27
6.3.1	Standard - Modified room disposition	27
6.3.2	Standard - Modified rescueboat size	27
6.3.3	Standard - Crew command	27
6.4	Discussion	27

7	Summary and Outlook	27
7.1	Summary	27
7.2	Outlook	27
8	References	28
9	Appendix	29
9.1	Code	29
9.1.1	<i>standard</i> code	29
9.1.2	<i>control exit</i> code	54
9.1.3	<i>C</i> code	82
9.1.4	<i>crew command</i> code	105

1 Abstract

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.

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 reproduction, 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 PC 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 and 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?

If we have time to spare 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?

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 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 Helbing, Farkas and Vicsek [2]. In order to observe an escape panic situation and especially bottlenecks, they built up a continuous-space model. In mathematical terms they took Newton's 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 τ_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 force 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 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. In order to answer our question we had to make several adjustments and adapt it to our simulation model. In this chapter we are going to explain the most significant changes we did. For the exact understanding of the original code and the process of optimization, please refer to the documentation of the previous project group, especially chapter five and nine[3] 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

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 the 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 by splitting the loop, in which we calculate the forces and moves of the agents, in two parts. First we loop over all floors higher than the exit floor in which the agents only are allowed to move down or take an exit. Secondly we do the same for all people in the floors lower than the exit floor where the passengers only can move up or take an exit.

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. Means for each agent there is one number set: If the agent reaches a staircase and therefore changes the floor he is a 1 otherwise he is a 0. The assumption also is a great simplification for the pictures since we do not have to mess with the problem of having overlapping stairs.

5.2.2 Different exits

Reason:

Because the exits model rescueboats, which can only hold a limited number of agents, we were forced to find a way how to differ the exits from each other and to assign a specific number to every exit.

Function:

loadConfig.m

New variables:

- *Exit_count*, to define the number of exit.
- For each exit k: *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:

We had to make some changes in the decoding of the pictures. 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 exits, to be able to distinguish them during the simulation. By defining every pixel which is red to value=0, blue value=0 and green value unequal to zero, we can define a lot of different colored exits by using green values from 256 to 256-*exit_count*.

We implemented the matrix *exit_nr* similar to the already existing one *imgerit*, in which we store a count to number the different exits. The number is defined by the

green value of the pixel we are at.

```
%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 (defined by his number).

```
%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
config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e)); save the number of agents the
exit can hold
end
```

Figure 2: Implementation of the exit_left matrix

In the loop where the forces are calculated and the agents moved, (in function `applyForcesandMove.m`) we added a piece of code, which updates the `exit_left` matrix at every time-step.

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.

```
%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`

If the allowed number of agents exited the number is 0. Now we have 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

5.2.4 Controlled evacuation

Reason:

With the goal of a faster evacuation, we tried to control the agents to go to specific exits.

Assumption:

We realised that the biggest problem-zones are the stairs. So, in order to get the agents as fast as possible away from the stairs, once they changed the floor, we decided to split the agents in two groups. One group only reaches one half of the exits and the other group for the others.

Function:

init_agents.m, addDesiredForce.m, initEscapeRoutes_even.m, initEscapeRoutes_odd.m, addDesiredForce.m, initialize.m, initAgents.m, loadConfig.m

New variables:

- *numbr*, each agents gets a number (0 or 1, selected randomly)
- *control_exit*, in the config file you can decide whether or not you want to have controlled exits during your simulation

Modifications:

Essentially, we had to modify the calculation of the force dragging a specific agent to the nearest exit. Therefore we wrote two new functions to init the escape routes.

- *initEscapeRoutes_even.m*, this function only considers the exits which are identified by an even number.

```
temp1=double(mod(data.exit_nr,2)); %matrix in which every number which is even
turns to zero, odd turns to one
temp2=logical((data.floor(i).img_exit)-(temp1));
boundary_data(temp2)=-1; %boundary_data considers only the exits with even num-
bers -- > -1
```

Figure 6: Implementation to init escape routes for even numbers

→ *initEscapeRoutes_.m*, this function only considers the exits which are identified by an odd number

```
temp=logical(mod(data.exit_nr,2)); %matrix in which every number which is even
turns to zero, odd turns to one
boundary_data(temp)=-1; %boundary_data considers only the exits with odd num-
bers
```

Figure 7: Implementation to init escape routes for odd numbers

This as a basis we got two different directions to the next exit.

```
→[data.floor(i).img_dir_x_odd, data.floor(i).img_dir_y_odd]
→[data.floor(i).img_dir_x_even, data.floor(i).img_dir_y_even]
```

Figure 8: Functions used to get the forces which drag the agents to the nearest exit

The agents are splitted in even-agents and odd-agents defined by the randomly added number (0 or 1).

```
%even agents
if numbr==0;
%get direction towards nearest exit
ex = lerp2(data.floor(fi).img_dir_x_even, p(1), p(2));
ey = lerp2(data.floor(fi).img_dir_y_even, p(1), p(2));
e = [ex ey];
%get force
Fi = m * (v0*e - v)/data.tau;
% add force
data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
end
```

Figure 9: Implementation to randomly split agents to even and odd

5.3 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 10th or 100th. 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

Modification:

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

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

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

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

% make video while simulation
close(vidObj);
```

Figure 10: Implementation to produce video file

5.4 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

Modification:

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

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

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

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

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

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

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

% save complete simulation
output = data.output;
save(textcolormagenta'output','output')
```

Figure 11: Implementation to produce the output

5.5 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_{99}\$ times.

Function:

postprocessing.m

New variables:

none of relevance

Modification:

We created the whole script ourself. It can be found in the annex.

5.6 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 maximal 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 for the previous group. The parameters concerning the ship shape were adjusted so that they represent reality as far as possible.

In the table in the annex a typical simulation input of the config file and some explanations can be found.

5.7 Ship Decks

Ship Deck Implementation:

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 full fill different purposes. There are decks for entertainment, eating, shopping, sport 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.

-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. Violet for spawn of agents, black for walls, red for upstairs, blue for downstairs and different green type for each rescue boat.

In the Figures 12 and 13 a small portion of a deck with stairs, rooms, doors and elevators is shown before and after conversion. In the converted picture is a red one pixel row included for the stairs, which is hard to see because of low contrast. The stairs conversion will just be explained in the next part.

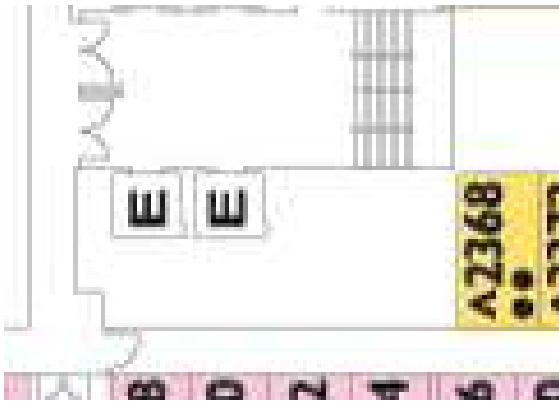


Figure 12: Deck before conversion

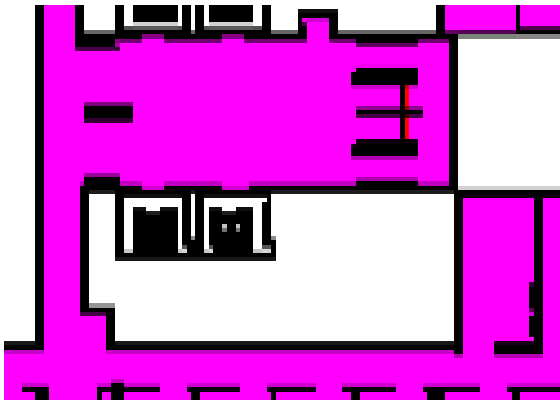


Figure 13: Deck after conversion

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.

In the work "Modeling Situations of Evacuation in a Multi-level Building" [3], which is the base of this project, are special additional floors to overcome the mentioned problem. 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 that additional floor for each connection.

While there is a finite number of floors on the ship and the agents only march 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 the Figure 14 a spiralling stairs connection between floor 01 and floor 02 is shown as example. The lines and arrows in light blue are used to clarify the technique. The agents arrive from the left on floor 01 and go into the direction of the arrow until they reach the red one pixel line. There they are skipped up on the floor 02 and try to go to the red one pixel line again. The line is now moved 2 pixels to the right and the agents go around to be skipped again on the floor 03 and so on. With that technique only a certain amount of floors can be realised corresponding to the pixel length of the stairs. But for the ship in that project it is sufficient, so that technique is used to implement the spiralling stairs.

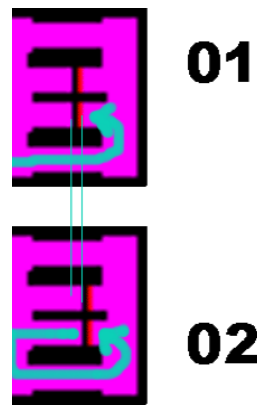


Figure 14: Stairs connection between floor 01 and floor 02

-Similarities and Reduction

The most important floor for the simulation is of course floor 04 because it holds all the rescue boats as seen on the Figure 16. 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 stairs correspond good to the reality, the further decks can differ more. So it is 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 the 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.

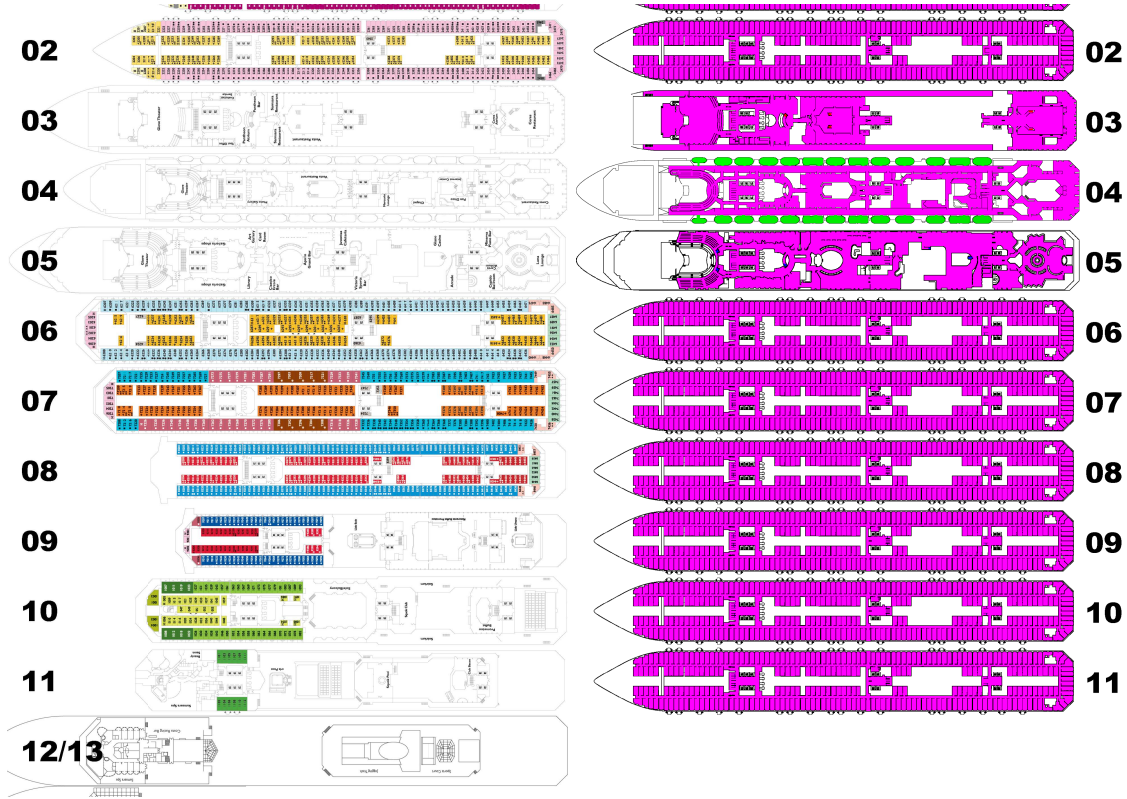


Figure 15: Original decks before conversion Figure 16: Deck approximated and converted

6 Simulation Results and Discussion

6.1 Expected Results

- 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 alinging 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 [6] we hope to see some real evacuation dynamics in this system and therefore make conclusion on the fundamental questions.

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%	99%
evacuation time	69s	272s	468s	614s

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

Further our standard ship simulation showed the following performance:

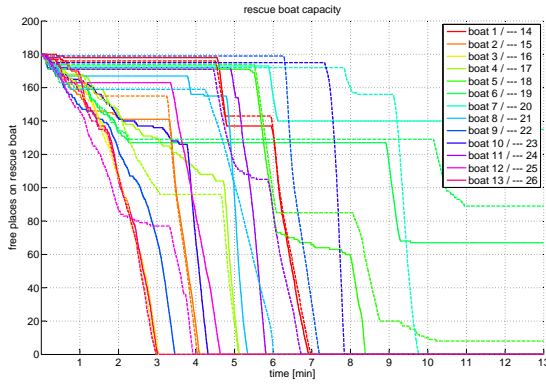


Figure 17: Standard simulation: Boat capacities during simulation

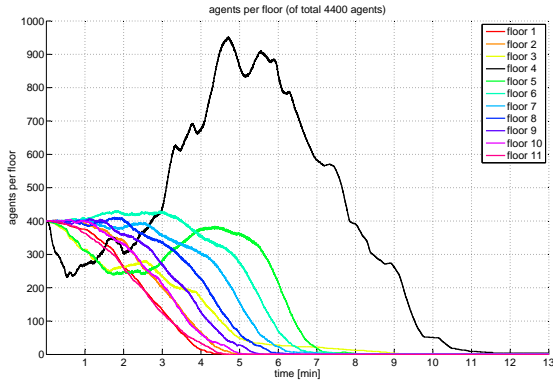


Figure 18: Standard simulation: Number of agents per floor

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:

percentage of agents.	10%	50%	90%	99%
evacuation time	72s	259s	461s	595s

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

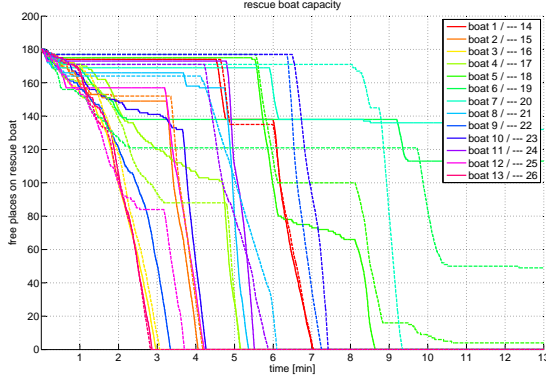


Figure 19: Added stairs simulation: Boat capacities during simulation

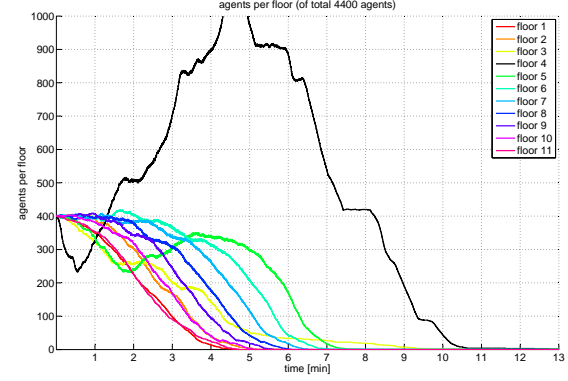


Figure 20: Added stairs simulation: Number of agents per floor

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 the lifeboats near to 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%	99%
evacuation time	71.44s	257.12s	419.4s	571.4s

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

6.2.4 Crew command

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 almost 20 seconds. Further 50% of all agents entered the exits approximately 13 seconds earlier compared to the standard model. In contrast to that the rescueboat capacity utilisation remains basically the same. Another point which was actually not important for our research is that you get a much higher agent density on the exit floor.

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 second half of the evacuation. The last 10% of agents are evacuated 48.6 seconds faster and the last 1% even 42.6 seconds faster. 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

6.4 Discussion

7 Summary and Outlook

7.1 Summary

7.2 Outlook

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

Some suggestions:

- To take into account that not all people automatically know where exactly the nearest exit is, the 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.

8 References

References

- [1] Helbing, Dirk (1995): Social Force Model for Pedestrians Dynamics.
- [2] Helbing, 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

- [8] Haverie of Costa Concordia (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)
2 %ADDAGENTREPULSIVEFORCE Summary of this function goes here
3 % Detailed explanation goes here
4
5 % Obstruction effects in case of physical interaction
6
7 % get maximum agent distance for which we calculate force
8 r_max = data.r_influence;
9 tree = 0;
10
11 for fi = 1:data.floor_count
12     pos = [arrayfun(@(a) a.p(1), data.floor(fi).agents);
13           arrayfun(@(a) a.p(2), data.floor(fi).agents)];
14
15     % update range tree of lower floor
16     tree_lower = tree;
17
18     agents_on_floor = length(data.floor(fi).agents);
19
20     % init range tree of current floor
21     if agents_on_floor > 0
22         tree = createRangeTree(pos);
23     end
24
25     for ai = 1:agents_on_floor
26         pi = data.floor(fi).agents(ai).p;
27         vi = data.floor(fi).agents(ai).v;
28         ri = data.floor(fi).agents(ai).r;
29
30         % 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, ...
32                           pi(2) - r_max, pi(2) + r_max)';
33
34         % loop over agents near agent ai
35         for aj = idx
36
37             % if force has not been calculated yet...
38             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;

42
43     % vector pointing from j to i
44     nij = (pi - pj) * data.meter_per_pixel;
45
46     % distance of agents
47     d = norm(nij);
48
49     % normalized vector pointing from j to i
50     nij = nij / d;
51     % tangential direction
52     tij = [-nij(2), nij(1)];
53
54     % sum of radii
55     rij = (ri + rj);
56
57     % repulsive interaction forces
58     if d < rij
59         T1 = data.k*(rij - d);
60         T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
61     else
62         T1 = 0;
63         T2 = 0;
64     end
65
66     F = (data.A * exp((rij - d)/data.B) + T1)*nij + T2;
67
68     data.floor(fi).agents(ai).f = ...
69         data.floor(fi).agents(ai).f + F;
70     data.floor(fi).agents(aj).f = ...
71         data.floor(fi).agents(aj).f - F;
72
73     end
74
75     % include agents on stairs!
76     if fi > 1
77         % use range tree to get the indices of all agents near agent ai
78         if ~isempty(data.floor(fi-1).agents)
79             idx = rangeQuery(tree_lower, pi(1) - r_max, ...
80                             pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';
81
82             % if there are any agents...
83             if ~isempty(idx)
84                 for aj = idx
85                     pj = data.floor(fi-1).agents(aj).p;
86                     if data.floor(fi-1).img_stairs_up(round(pj(1)),
87                                                         round(pj(2)))

```

```

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     % distance of agents
    d = norm(nij);

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

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

105    % repulsive interaction forces
107    if d < rij
        T1 = data.k*(rij - d);
        T2 = data.kappa*(rij - d)*dot((vj -
            vi),tij)*tij;
109    else
        T1 = 0;
111        T2 = 0;
    end
113
    F = (data.A * exp((rij - d)/data.B) + T1)*nij
        + T2;

115
117    data.floor(fi).agents(ai).f = ...
        data.floor(fi).agents(ai).f + F;
119    data.floor(fi-1).agents(aj).f = ...
        data.floor(fi-1).agents(aj).f - F;
    end
121 end
123 end
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
      p = data.floor(fi).agents(ai).p;
11      m = data.floor(fi).agents(ai).m;
      v0 = data.floor(fi).agents(ai).v0;
13      v = data.floor(fi).agents(ai).v;

15

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

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

23

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

```

Listing 2: addDesiredForce.m

```

function data = addWallForce(data)
2 %ADDWALLFORCE adds wall's force contribution to each agent

4 for fi = 1:data.floor_count

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

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

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

18

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

22

24        % calculate force
        if diW < ri
26            T1 = data.k * (ri - diW);

```



```

28         T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
    else
30         T1 = 0;
        T2 = 0;
    end
32     Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;

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

```

Listing 3: addWallForce.m

```

function data = applyForcesAndMove(data)
2 %APPLYFORCESANDMOVE apply current forces to agents and move them using
  %the timestep and current velocity
4
n_velocity_clamps = 0;
6
% loop over all floors higher than exit floor
8 for fi = data.floor_exit:data.floor_count

10     % init logical arrays to indicate agents that change the floor or exit
    % the simulation
12     floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);
14
    % loop over all agents
16     for ai=1:length(data.floor(fi).agents)
        % add current force contributions to velocity
18         v = data.floor(fi).agents(ai).v + data.dt * ...
            data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;

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

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

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

36         % get agent's position

```

```

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
88 end

```

```

82         end
83     else
84         % get rid of disturbing agent
85         fprintf('WARNING: position of an agent is NaN! Deleted this
86             agent.\n')
87         exited(ai) = 1;
88         data.agents_exited = data.agents_exited + 1;
89         data.output.deleted_agents=data.output.deleted_agents+1;
90         newp = [1 1];
91     end
92
93     % update agent's velocity and position
94     data.floor(fi).agents(ai).v = v;
95     data.floor(fi).agents(ai).p = newp;
96
97     % reset forces for next timestep
98     data.floor(fi).agents(ai).f = [0 0];
99
100    % check if agent reached a staircase down and indicate floor change
101    if data.floor(fi).img_stairs_down(round(newp(1)), round(newp(2)))
102        floorchange(ai) = 1;
103    end
104
105    % check if agent reached an exit
106    if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
107        exited(ai) = 1;
108        data.agents_exited = data.agents_exited + 1;
109
110        %
111        fprintf('agent exited from upper loop\n');
112
113        %save current exit nr
114        data.current_exit = data.exit_nr(round(newp(1)),
115            round(newp(2)));
116
117        %
118        fprintf(int2str(data.current_exit));
119
120        %update exit_left
121        data.exit_left(1,data.current_exit) =
122            data.exit_left(1,data.exit_nr(round(newp(1)),
123                round(newp(2)))) - 1;
124
125        %close exit if there is no more free space
126        if data.exit_left(1,data.current_exit) < 1
127
128            %change current exit to wall
129            data.floor(data.floor_exit).img_wall =
130                data.floor(data.floor_exit).img_wall == 1 ...
131                | (data.exit_nr == (data.current_exit));

```

```

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

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

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

134        end
136    end

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

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

146
148
150
152    % loop over all floors lower than exit floor
    for fi = 1:data.floor_exit

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

158
        % loop over all agents
160        for ai=1:length(data.floor(fi).agents)
            % add current force contributions to velocity
162            v = data.floor(fi).agents(ai).v + data.dt * ...
                data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;

164
            % clamp velocity
166            if norm(v) > data.v_max
                v = v / norm(v) * data.v_max;
168                n_velocity_clamps = n_velocity_clamps + 1;
            end

170
            % get agent's new position

```

```

172     newp = data.floor(fi).agents(ai).p + ...
173         v * data.dt / data.meter_per_pixel;
174
175     % 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
183         % 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     % check if agents position is ok
198     % repositioning after 50 times clogging
199     % deleting if agent has a NaN position
200     if ~isnan(newp)
201         if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
202             newp = data.floor(fi).agents(ai).p;
203             v = [0 0];
204             data.floor(fi).agents(ai).clogged =
205                 data.floor(fi).agents(ai).clogged + 1;
206             fprintf('WARNING: clogging agent %i on floor %i (%i).
207                 Position
208                 (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
209             if data.floor(fi).agents(ai).clogged >= 40
210                 nx = rand(1)*2 - 1;
211                 ny = rand(1)*2 - 1;
212                 n = [nx ny];
213                 v = n*data.v_max/2;
214                 fprintf('WARNING: agent %i on floor %i velocity set
215                     random to get out of wall. Position
216                     (%f,%f).\n',ai,fi,newp(1),newp(2))
217
218                 % get agent's new position
219                 newp = data.floor(fi).agents(ai).p + ...
220                     v * data.dt / data.meter_per_pixel;

```

```

218         if isnan(newp)
                % get rid of disturbing agent
                fprintf('WARNING: position of an agent is NaN!
                    Deleted this agent.\n')
220                 exited(ai) = 1;
                data.agents_exited = data.agents_exited +1;
222                 data.output.deleted_agents=data.output.deleted_agents+1;
                newp = [1 1];
224             end
        end
226     end
else
228     % get rid of disturbing agent
    fprintf('WARNING: position of an agent is NaN! Deleted this
        agent.\n')
230     exited(ai) = 1;
    data.agents_exited = data.agents_exited +1;
232     data.output.deleted_agents=data.output.deleted_agents+1;
    newp = [1 1];
234 end

236 % update agent's velocity and position
data.floor(fi).agents(ai).v = v;
238 data.floor(fi).agents(ai).p = newp;

240 % reset forces for next timestep
data.floor(fi).agents(ai).f = [0 0];
242

244 % check if agent reached a staircase up and indicate floor change
if data.floor(fi).img_stairs_up(round(newp(1)), round(newp(2)))
    floorchange(ai) = 1;
246 end

248 % check if agent reached an exit
if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
250     exited(ai) = 1;
    data.agents_exited = data.agents_exited +1;
252

%         fprintf('agent exited from lower loop\n');
254

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

258     %update exit_left
    data.exit_left(1,data.current_exit) =
        data.exit_left(1,data.exit_nr(round(newp(1)),
            round(newp(2)))) - 1;
260

        %close exit if there is no more free space

```

```

262         if data.exit_left(1,data.current_exit) < 1
264             %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));
266             data.floor(data.floor_exit).img_exit =
                 data.floor(data.floor_exit).img_exit == 1 ...
268                 & (data.exit_nr ~= (data.current_exit));

270             %redo initEscapeRoutes and initWallForces with new exit
                and wall parameters
             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 ...
284                                   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 n_velocity_clamps > 0
292     %     fprintf(['WARNING: clamped velocity of %d agents, ' ...
293                 'possible simulation instability.\n'], n_velocity_clamps);
294 % end

```

Listing 4: applyForcesAndMove.m

```

function val = checkForIntersection(data, floor_idx, agent_idx)
2 % check an agent for an intersection with another agent or a wall
% the check is kept as simple as possible
4 %
% arguments:
6 %     data           global data structure
%     floor_idx       which floor to check
8 %     agent_idx       which agent on that floor
%     agent_new_pos    vector: [x,y], desired agent position to check
10 %
% 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
31 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
5
6
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
12 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
37               idx = randi(length(x));
               data.floor(i).agents(cur_agent).p = [y(idx), x(idx)];
39               if checkForIntersection(data, i, cur_agent) == 0
                   tries = -1; % leave the loop
41               end
               tries = tries - 1;
43           end
45           if tries > -1
               %remove the last agent
               data.floor(i).agents = data.floor(i).agents(1:end-1);
47           end
49           end
           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
59     error('no spots to place agents!');
end
61

```

```
end
```

Listing 7: initAgents.m

```
function data = initEscapeRoutes(data)
2 %INITESCAPEROUTES Summary of this function goes here
%   Detailed explanation goes here
4
for i=1:data.floor_count
6
    boundary_data = zeros(size(data.floor(i).img_wall));
8    boundary_data(data.floor(i).img_wall) = 1;

10 if i<data.floor_exit
    boundary_data(data.floor(i).img_stairs_up) = -1;
12
elseif i>data.floor_exit
14    boundary_data(data.floor(i).img_stairs_down) = -1;

16    else
        boundary_data(data.floor(i).img_exit) = -1;
18
end
20 exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    [data.floor(i).img_dir_x, data.floor(i).img_dir_y] = ...
22    getNormalizedGradient(boundary_data, -exit_dist);
end
```

Listing 8: initEscapeRoutes.m

```
function data = initialize(config)
2 % initialize the internal data from the config data
%
4 % arguments:
%   config      data structure from loadConfig()
6 %
% return:
8 %   data      data structure: all internal data used for the main loop
%
10 %           all internal data is stored in pixels NOT in meters

12
data = config;
14
%for convenience
16 data.pixel_per_meter = 1/data.meter_per_pixel;

18 fprintf('Init escape routes...\n');
    data = initEscapeRoutes(data);
20 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
44     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
46         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
48         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
50         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
data.output_file_name = ['output_' data.frame_basename];

62 % prepare video file name
64 data.video_file_name = ['video_' data.frame_basename '.avi'];

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

```

Listing 9: initialize.m

```

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

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

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

```

Listing 10: initWallForces.m

```

1 function config = loadConfig(config_file)
% load the configuration file
3 %
% arguments:
5 % config_file      string, which configuration file to load
%
7
9 % get the path from the config file -> to read the images
config_path = fileparts(config_file);
11 if strcmp(config_path, '') == 1
    config_path = '.';
13 end

15 fid = fopen(config_file);
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 ...
41                                   & img_build(:, :, 3) == 0);

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

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 ...
53                                   & img_build(:, :, 3) == 0);

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

59     config.floor(i).img_stairs_down = (img_build(:, :, 1) == 0 ...
61                                   & 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
        %make a zeroes matrix as big as img_exit
71     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
            %build the exit_left matrix
83            config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));
85
        end
        end
87
        %init the plot image here, because this won't change
89     config.floor(i).img_plot = 5*config.floor(i).img_wall ...
        + 4*config.floor(i).img_stairs_up ...
91     + 3*config.floor(i).img_stairs_down ...
        + 2*config.floor(i).img_exit ...
93     + 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)
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
if floor_idx~=data.floor_count
10     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-');
18 hold off;
20
20 title(sprintf('%i', floor_idx));

```

Listing 12: plotAgentsPerFloor.m

```

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

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

```

Listing 13: plotExitedAgents.m

```

function plotFloor(data, floor_idx)
2
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
6 set(h,
    'position',[0,0.35+0.65/3*(floor_idx-data.floor_exit+1),1,0.65/3-0.005]);
8
8 hold off;
8 % the building image
10 imagesc(data.floor(floor_idx).img_plot);
10 hold on;
12
12 %plot options
14 colormap(data.color_map);
14 axis equal;
16 axis manual; %do not change axis on window resize
18
18 set(h, 'Visible', 'off')
18 % title(sprintf('floor %i', floor_idx))
20
20 % plot agents
22 if ~isempty(data.floor(floor_idx).agents)
22     ang = [linspace(0,2*pi, 10) nan]';
24     rmul = [cos(ang) sin(ang)] * data.pixel_per_meter;
24     draw = cell2mat(arrayfun(@(a) repmat(a,p,length(ang),1) + a.r*rmul, ...
26         data.floor(floor_idx).agents, 'UniformOutput', false)'));
26     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
26     if agents_on_boat(i)<0
        steps=i-2;
28         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
42     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 t99=0;
71 for i=1:steps
72     if agents_left(i)<agents_start*0.99
73         t99=t99+dt;
74     end
75 end
76 if t99~=0
77     t99=t99+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', [1:1:13], 'FontSize', 16)
   plot(time/60, agents_left/agents_start*100, 'LineWidth', 2)
110 axis([0 13 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
118 grid on
   set(gca, 'XTick', [1:1:13], 'FontSize', 16)
120 list = cell(config.floor_count, 1);
   color = hsv(config.floor_count);
122 color(config.floor_exit, :) = [0 0 0];
   for i=1:config.floor_count
124     plot(time/60, agents_per_floor(i,:), 'LineWidth', 2, 'color', color(i,:))
       list{i} = [sprintf('floor %i', i)];
126 end
   legend(list)
128
   axis([0 13 0 1000])
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', [1:1:13], 'FontSize', 16)
   list = cell(config.exit_count/2, 1);
140 color = hsv(config.exit_count/2);
   for i=1:config.exit_count/2
142     plot(time/60, exit_left(i,:), 'LineWidth', 2, 'color', color(i,:))
       list{i} = [sprintf('boat %i / --- %i', i, i+13)];

```

```

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

150 axis([0 13 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) ] );

160 % print out
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)

170 fprintf('t_10: %f\n', t10)
171 fprintf('t_50: %f\n', t50)
172 fprintf('t_90: %f\n', t90)
173 fprintf('t_99: %f\n', t99)
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
    for floor=1:data.floor_count
39         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
        set(0,'CurrentFigure',data.figure_floors);
49         for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
51             plotFloor(data, floor);
        end
53
        if data.save_frames==1
55             % print('-depsc2',sprintf('frames/%s_%04i.eps', ...
%                 data.frame_basename,data.step), data.figure_floors);

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

61
        end
63
        set(0,'CurrentFigure',data.figure_exit);

```

```

65     plotExitedAgents(data);

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

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

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

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

85
    end

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

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

99 end

101 %make video while simulation
    close(vidObj);
103
    % toc of whole simulation
105 data.output.simulation_time = toc(simstart);

107 % save complete simulation
    output = data.output;
109 save('output','output')
    fprintf('Simulation done in %i seconds and saved data to output file.\n',
        data.output.simulation_time);

```

```

111 % save diary
113 diary

```

Listing 16: simulate.m

9.1.2 control exit 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

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

37            % if force has not been calculated yet...
            if aj > ai
39                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
47         d = norm(nij);
49
         % normalized vector pointing from j to i
         nij = nij / d;
51         % tangential direction
         tij = [-nij(2), nij(1)];
53
         % sum of radii
55         rij = (ri + rj);
57
         % repulsive interaction forces
         if d < rij
59             T1 = data.k*(rij - d);
             T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
61         else
             T1 = 0;
63             T2 = 0;
         end
65
         F = (data.A * exp((rij - d)/data.B) + T1)*nij + T2;
67
         data.floor(fi).agents(ai).f = ...
69             data.floor(fi).agents(ai).f + F;
         data.floor(fi).agents(aj).f = ...
71             data.floor(fi).agents(aj).f - F;
         end
73     end
75
    % include agents on stairs!
    if fi > 1
77         % use range tree to get the indices of all agents near agent ai
         if ~isempty(data.floor(fi-1).agents)
79             idx = rangeQuery(tree_lower, pi(1) - r_max, ...
                             pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';
81
             % if there are any agents...
83             if ~isempty(idx)
                 for aj = idx
85                     pj = data.floor(fi-1).agents(aj).p;
                     if data.floor(fi-1).img_stairs_up(round(pj(1)),
87                         round(pj(2)))
                         vj = data.floor(fi-1).agents(aj).v;
89                         rj = data.floor(fi-1).agents(aj).r;

```

```

91         % vector pointing from j to i
          nij = (pi - pj) * data.meter_per_pixel;
93
          % distance of agents
95         d = norm(nij);
97
          % normalized vector pointing from j to i
          nij = nij / d;
99         % tangential direction
          tij = [-nij(2), nij(1)];
101
          % sum of radii
103         rij = (ri + rj);
105
          % repulsive interaction forces
          if d < rij
107             T1 = data.k*(rij - d);
             T2 = data.kappa*(rij - d)*dot((vj -
                 vi),tij)*tij;
109         else
             T1 = 0;
111             T2 = 0;
          end
113
          F = (data.A * exp((rij - d)/data.B) + T1)*nij
              + T2;
115
          data.floor(fi).agents(ai).f = ...
              data.floor(fi).agents(ai).f + F;
117         data.floor(fi-1).agents(aj).f = ...
              data.floor(fi-1).agents(aj).f - F;
119     end
121 end
123 end
125 end
end

```

Listing 17: 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

```



```

11     p = data.floor(fi).agents(ai).p;
12     m = data.floor(fi).agents(ai).m;
13     v0 = data.floor(fi).agents(ai).v0;
14     v = data.floor(fi).agents(ai).v;
15     numbr= data.floor(fi).agents(ai).nr;

16     %control exit

17
18     if data.control_exit==1
19
20         %even agents
21         if numbr==0;
22         % get direction towards nearest exit
23         ex = lerp2(data.floor(fi).img_dir_x_even, p(1), p(2));
24         ey = lerp2(data.floor(fi).img_dir_y_even, p(1), p(2));
25         e = [ex ey];
26         % get force
27         Fi = m * (v0*e - v)/data.tau;

28
29         % add force
30         data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f
31         + Fi;
32     end

33     %odd agents
34     if numbr==1;
35     % get direction towards nearest exit
36     ex = lerp2(data.floor(fi).img_dir_x_odd, p(1), p(2));
37     ey = lerp2(data.floor(fi).img_dir_y_odd, p(1), p(2));
38     e = [ex ey];
39     % get force
40     Fi = m * (v0*e - v)/data.tau;

41
42     % add force
43     data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f
44     + Fi;

45     end

46
47
48     else

49
50
51
52
53     % get direction towards nearest exit
54     ex = lerp2(data.floor(fi).img_dir_x, p(1), p(2));
55     ey = lerp2(data.floor(fi).img_dir_y, p(1), p(2));
56     e = [ex ey];

```

```

59     % get force
    Fi = m * (v0*e - v)/data.tau;
61
    % add force
63     data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
    end
65 end
end

```

Listing 18: addDesiredForce.m

```

1 function data = addWallForce(data)
    %ADDWALLFORCE adds wall's force contribution to each agent
3
    for fi = 1:data.floor_count
5
        for ai=1:length(data.floor(fi).agents)
7            % get agents data
            p = data.floor(fi).agents(ai).p;
9            ri = data.floor(fi).agents(ai).r;
            vi = data.floor(fi).agents(ai).v;
11
            % get direction from nearest wall to agent
13            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));
15
            % get distance to nearest wall
17            diW = lerp2(data.floor(fi).img_wall_dist, p(1), p(2));
19
            % get perpendicular and tangential unit vectors
            niW = [ nx ny];
21            tiW = [-ny nx];
23
            % calculate force
25            if diW < ri
                T1 = data.k * (ri - diW);
27                T2 = data.kappa * (ri - diW) * dot(vi, tiW) * tiW;
            else
29                T1 = 0;
                T2 = 0;
31            end
            Fi = (data.A * exp((ri-diW)/data.B) + T1)*niW - T2;
33
            % add force to agent's current force
35            data.floor(fi).agents(ai).f = data.floor(fi).agents(ai).f + Fi;
        end
37 end

```

Listing 19: addWallForce.m

```

function data = applyForcesAndMove(data)
2 %APPLYFORCESANDMOVE apply current forces to agents and move them using
  %the timestep and current velocity
4
n_velocity_clamps = 0;
6
  % loop over all floors higher than exit floor
8 for fi = data.floor_exit:data.floor_count

    % init logical arrays to indicate agents that change the floor or exit
    % the simulation
12 floorchange = false(length(data.floor(fi).agents),1);
    exited = false(length(data.floor(fi).agents),1);
14

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

20
        % clamp velocity
22 if norm(v) > data.v_max
            v = v / norm(v) * data.v_max;
24            n_velocity_clamps = n_velocity_clamps + 1;
        end

26
        % get agent's new position
28 newp = data.floor(fi).agents(ai).p + ...
            v * data.dt / data.meter_per_pixel;

30
        % if the new position is inside a wall, remove perpendicular
        % component of the agent's velocity
32 if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
            data.floor(fi).agents(ai).r

34

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

38

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

44

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

48

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

```

```

50         v * data.dt / data.meter_per_pixel;
51     end
52
53     % check if agents position is ok
54     % repositioning after 50 times clogging
55     % deleting if agent has a NaN position
56     if ~isnan(newp)
57         if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
58             newp = data.floor(fi).agents(ai).p;
59             v = [0 0];
60             data.floor(fi).agents(ai).clogged =
61                 data.floor(fi).agents(ai).clogged + 1;
62             fprintf('WARNING: clogging agent %i on floor %i (%i).
63                 Position
64                 (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
65             if data.floor(fi).agents(ai).clogged >= 40
66                 nx = rand(1)*2 - 1;
67                 ny = rand(1)*2 - 1;
68                 n = [nx ny];
69                 v = n*data.v_max/2;
70                 fprintf('WARNING: agent %i on floor %i velocity set
71                     random to get out of wall. Position
72                     (%f,%f).\n',ai,fi,newp(1),newp(2))
73
74                 % get agent's new position
75                 newp = data.floor(fi).agents(ai).p + ...
76                     v * data.dt / data.meter_per_pixel;
77                 if isnan(newp)
78                     % get rid of disturbing agent
79                     fprintf('WARNING: position of an agent is NaN!
80                         Deleted this agent.\n')
81                     exited(ai) = 1;
82                     data.agents_exited = data.agents_exited +1;
83                     data.output.deleted_agents=data.output.deleted_agents+1;
84                     newp = [1 1];
85                 end
86             end
87         end
88     else
89         % get rid of disturbing agent
90         fprintf('WARNING: position of an agent is NaN! Deleted this
91             agent.\n')
92         exited(ai) = 1;
93         data.agents_exited = data.agents_exited +1;
94         data.output.deleted_agents=data.output.deleted_agents+1;
95         newp = [1 1];
96     end
97
98     % update agent's velocity and position

```

```

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

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

98

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

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

108    %
        fprintf('agent exited from upper loop\n');

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

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

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

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

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

128        %redo initEscapeRoutes and initWallForces with new exit
            and wall parameters

130        if data.control_exit~=1
            data = initEscapeRoutes(data);

132        %control exits
    else
134        if data.floor(fi).agents(ai).nr==0    %agents with
            number 0 --> only use even exits

```

```

136         data = initEscapeRoutes_even(data);
138
139         else
140             data = initEscapeRoutes_odd(data);
141         end
142     end
143     data = initWallForces(data);
144
145     %             fprintf('new routes from upper loop\n');
146
147     end
148 end
149 end
150
151 % add appropriate agents to next lower floor
152 if fi > data.floor_exit
153     data.floor(fi-1).agents = [data.floor(fi-1).agents
154                               data.floor(fi).agents(floorchange)];
155 end
156
157 % delete these and exited agents
158 data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
159 end
160
161
162
163
164 % loop over all floors lower than exit floor
165 for fi = 1:data.floor_exit
166
167     % init logical arrays to indicate agents that change the floor or exit
168     % the simulation
169     floorchange = false(length(data.floor(fi).agents),1);
170     exited = false(length(data.floor(fi).agents),1);
171
172     % loop over all agents
173     for ai=1:length(data.floor(fi).agents)
174         % add current force contributions to velocity
175         v = data.floor(fi).agents(ai).v + data.dt * ...
176             data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
177
178         % clamp velocity
179         if norm(v) > data.v_max
180             v = v / norm(v) * data.v_max;
181             n_velocity_clamps = n_velocity_clamps + 1;
182         end
183
184         % get agent's new position

```

```

186     newp = data.floor(fi).agents(ai).p + ...
        v * data.dt / data.meter_per_pixel;

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

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

196         % get wall distance gradient (which is of course perpendicular
    % to the nearest wall)
198         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));
200         n = [nx ny];

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

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

210
212     % check if agents position is ok
    % repositioning after 50 times clogging
    % deleting if agent has a NaN position
214     if ~isnan(newp)
        if data.floor(fi).img_wall(round(newp(1)), round(newp(2)))
216             newp = data.floor(fi).agents(ai).p;
            v = [0 0];
218             data.floor(fi).agents(ai).clogged =
                data.floor(fi).agents(ai).clogged + 1;
            fprintf('WARNING: clogging agent %i on floor %i (%i).
                Position
                (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
220             if data.floor(fi).agents(ai).clogged >= 40
                nx = rand(1)*2 - 1;
                ny = rand(1)*2 - 1;
222                 n = [nx ny];
                v = n*data.v_max/2;
224                 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))

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

```

```

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

```



```

276         if data.exit_left(1,data.current_exit) < 1
277
278             %change current exit to wall
279             data.floor(data.floor_exit).img_wall =
280                 data.floor(data.floor_exit).img_wall == 1 ...
281                     | (data.exit_nr == (data.current_exit));
282             data.floor(data.floor_exit).img_exit =
283                 data.floor(data.floor_exit).img_exit == 1 ...
284                     & (data.exit_nr ~= (data.current_exit));
285
286             %redo initEscapeRoutes and initWallForces with new exit
287             and wall parameters
288             if data.control_exit~=1
289                 data = initEscapeRoutes(data);
290
291             %control exits
292             else
293
294                 if data.floor(fi).agents.nr==0; %agent with number ==
295                     0 , only use even numbers
296
297                 data = initEscapeRoutes_even(data);
298
299                 else
300
301                 data = initEscapeRoutes_odd(data);
302
303                 end
304             end
305             data = initWallForces(data);
306
307             fprintf('new routes from lower loop\n');
308
309         end
310
311     end
312
313     % add appropriate agents to next lower floor
314     if fi < data.floor_exit
315         data.floor(fi+1).agents = [data.floor(fi+1).agents ...
316                                     data.floor(fi).agents(floorchange)];
317     end
318
319     % delete these and exited agents
320     data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
321 end
322
323 % if n_velocity_clamps > 0
324 %     fprintf(['WARNING: clamped velocity of %d agents, ' ...

```

```

%           'possible simulation instability.\n'], n_velocity_clamps);
322 % end

```

Listing 20: applyForcesAndMove.m

```

function val = checkForIntersection(data, floor_idx, agent_idx)
2 % check an agent for an intersection with another agent or a wall
% the check is kept as simple as possible
4 %
% arguments:
6 % data           global data structure
% floor_idx       which floor to check
8 % agent_idx      which agent on that floor
% agent_new_pos   vector: [x,y], desired agent position to check
10 %
% return:
12 % 0              for no intersection
% 1              has an intersection with wall
14 % 2              with another agent

16 val = 0;

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

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

```

Listing 21: checkForIntersection.m

```

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

```

Listing 22: 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
    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
35                 %control exit
37                 data.floor(i).agents(cur_agent).nr=logical(round(rand(1)));
39
                    %even-->0
                    %odd-->1
41
                    tries = 10;
43                     while tries > 0
                        % randomly pick a spot and check if it's free
45                         idx = randi(length(x));
                            data.floor(i).agents(cur_agent).p = [y(idx), x(idx)];

```

```

47         if checkForIntersection(data, i, cur_agent) == 0
48             tries = -1; % leave the loop
49         end
50         tries = tries - 1;
51     end
52     if tries > -1
53         %remove the last agent
54         data.floor(i).agents = data.floor(i).agents(1:end-1);
55     end
56     end
57     data.total_agent_count = data.total_agent_count +
58         length(data.floor(i).agents);
59
60     if length(data.floor(i).agents) ~= agent_count
61         fprintf(['WARNING: could only place %d agents on floor %d ' ...
62             'instead of the desired %d.\n'], ...
63             length(data.floor(i).agents), i, agent_count);
64     end
65 end
66 if floors_with_agents==0
67     error('no spots to place agents!');
68 end
69
70 end

```

Listing 23: initAgents.m

```

function data = initEscapeRoutes(data)
2 %INITESCAPEROUTES Summary of this function goes here
3 % Detailed explanation goes here
4
5 for i=1:data.floor_count
6
7     boundary_data = zeros(size(data.floor(i).img_wall));
8     boundary_data(data.floor(i).img_wall) = 1;
9
10
11 if i<data.floor_exit
12     boundary_data(data.floor(i).img_stairs_up) = -1;
13
14 elseif i>data.floor_exit
15     boundary_data(data.floor(i).img_stairs_down) = -1;
16
17 else
18     boundary_data(data.floor(i).img_exit) = -1;
19
20 end
21
22 exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
23 [data.floor(i).img_dir_x, data.floor(i).img_dir_y] = ...

```

```

24         getNormalizedGradient(boundary_data, -exit_dist);
end

```

Listing 24: initEscapeRoutes.m

```

1 function data = initEscapeRoutes_even(data)
2 %INITESCAPEROUTES Summary of this function goes here
3 % Detailed explanation goes here
4 %only even numbered rescue boats are exits
5
6
7 for i=1:data.floor_count
8
9     boundary_data = zeros(size(data.floor(i).img_wall));
10    boundary_data(data.floor(i).img_wall) = 1;
11
12    if i<data.floor_exit
13        boundary_data(data.floor(i).img_stairs_up) = -1;
14
15    elseif i>data.floor_exit
16        boundary_data(data.floor(i).img_stairs_down) = -1;
17
18    else
19
20        temp1=double(mod(data.exit_nr,2)); %matrix in which every number
21        which is even turns to zero, odd turns to one
22        temp2=logical((data.floor(i).img_exit)-(temp1));
23        % temp2=logical((data.floor(i).img_exit)-(temp1));
24        % boundary_data(temp2)=-1; %boundary_data considers
25        only the exits with even numbers --> set -1 where those are
26        temp2=((data.floor(i).img_exit)-(temp1));
27        temp3=logical(mod(temp2,2));
28        boundary_data(temp3)=-1;
29
30
31 end
32
33 fprintf('Init escaperoutes_even...\n');
34 exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
35 [data.floor(i).img_dir_x_even, data.floor(i).img_dir_y_even] = ...
36     getNormalizedGradient(boundary_data, -exit_dist);
37
38
39 end

```

Listing 25: initEscapeRoutes_even.m

```

1 function data = initEscapeRoutes_odd(data)
2 %INITESCAPEROUTES Summary of this function goes here
3 % Detailed explanation goes here
4 %only odd numbered rescue boats are exits
5
6
7 for i=1:data.floor_count
8
9     boundary_data = zeros(size(data.floor(i).img_wall));
10    boundary_data(data.floor(i).img_wall) = 1;
11
12    if i<data.floor_exit
13
14        boundary_data(data.floor(i).img_stairs_up) = -1;
15
16
17    elseif i>data.floor_exit
18        boundary_data(data.floor(i).img_stairs_down) = -1;
19
20
21    else
22
23        temp=logical(mod(data.exit_nr,2)); %matrix in which every number
24        which is even turns to zero, odd turns to one
25        boundary_data(temp) = -1; %boundary_data considers only the
26        exits with odd numbers
27
28    end
29
30    fprintf('Init escaperoutes_odd...\n');
31    exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
32    [data.floor(i).img_dir_x_odd, data.floor(i).img_dir_y_odd] = ...
33        getNormalizedGradient(boundary_data, -exit_dist);
34
35
36 end

```

Listing 26: initEscapeRoutes_odd.m

```

1 function data = initialize(config)
2 % initialize the internal data from the config data
3 %
4 % arguments:
5 % config      data structure from loadConfig()
6 %
7 % return:
8 % data        data structure: all internal data used for the main loop
9 %

```

```

%           all internal data is stored in pixels NOT in meters
11
13 data = config;

15 %for convenience
data.pixel_per_meter = 1/data.meter_per_pixel;
17
fprintf('Init escape routes...\n');
19 if data.control_exit~=1
    data = initEscapeRoutes(data);
21
    %control exits
23
25
27 else

29     data = initEscapeRoutes_even(data);
    data = initEscapeRoutes_odd(data);
31 end

33 fprintf('Init wall forces...\n');
data = initWallForces(data);
35 fprintf('Init agents...\n');
data = initAgents(data);
37
% maximum influence of agents on each other
39
data.r_influence = data.pixel_per_meter * ...
41     fzero(@(r) data.A * exp((2*data.r_max-r)/data.B) - 1e-4, data.r_max);

43 fprintf('Init plots...\n');
%init the plots
45 %exit plot
data.figure_exit=figure;
47 hold on;
axis([0 data.duration 0 data.total_agent_count]);
49 title(sprintf('agents that reached the exit (total agents: %i)',
    data.total_agent_count));

51 % floors plot
data.figure_floors=figure;
53 % figure('units','normalized','outerposition',[0 0 1 1])
data.figure_floors_subplots_w = data.floor_count;
55 data.figure_floors_subplots_h = 4;
for i=1:config.floor_count
57     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
59         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
61         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
63         data.floor(i).building_plot =
            subplot(data.figure_floors_subplots_h,
                    data.figure_floors_subplots_w, [1:config.floor_count]);
    end
65 end

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

73 % prepare output file name
data.output_file_name = ['output_' data.frame_basename];
75
% prepare video file name
77 data.video_file_name = ['video_' data.frame_basename '.avi'];

79 % set deleted_agents to zero
data.output.deleted_agents = 0;

```

Listing 27: initialize.m

```

1 function data = initWallForces(data)
%INITWALLFORCES init wall distance maps and gradient maps for each floor
3
for i=1:data.floor_count
5
    % init boundary data for fast sweeping method
7    boundary_data = zeros(size(data.floor(i).img_wall));
    boundary_data(data.floor(i).img_wall) = -1;
9
    % get wall distance
11    wall_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    data.floor(i).img_wall_dist = wall_dist;
13

```



```

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

```

Listing 28: initWallForces.m

```

1 function config = loadConfig(config_file)
   % load the configuration file
3   %
   % arguments:
5   %   config_file      string, which configuration file to load
   %
7
9   % get the path from the config file -> to read the images
   config_path = fileparts(config_file);
11  if strcmp(config_path, '') == 1
       config_path = '.';
13  end
15  fid = fopen(config_file);
   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 ...
41                                   & img_build(:, :, 3) == 0);

```

```

43     config.floor(i).img_spawn = (img_build(:, :, 1) == 255 ...
44                                 & img_build(:, :, 2) == 0 ...
45                                 & img_build(:, :, 3) == 255);
46
47     %pixel is exit if 1-->0, 3-->0, and if 2 is between 255 and 230 or if no
48     %red or blue
49
50
51     config.floor(i).img_exit = (img_build(:, :, 1) == 0 ...
52                                 & img_build(:, :, 2) ~= 0 ...
53                                 & img_build(:, :, 3) == 0);
54
55
56     config.floor(i).img_stairs_up = (img_build(:, :, 1) == 255 ...
57                                       & img_build(:, :, 2) == 0 ...
58                                       & img_build(:, :, 3) == 0);
59
60     config.floor(i).img_stairs_down = (img_build(:, :, 1) == 0 ...
61                                         & img_build(:, :, 2) == 0 ...
62                                         & img_build(:, :, 3) == 255);
63
64
65     if i == config.floor_exit
66
67         %make the exit_nr matrix where the number of exit is indicated in
68         each
69         %pixel
70
71         %make a zeroes matrix as big as img_exit
72         config.exit_nr=zeros(size(config.floor(config.floor_exit).img_exit));
73
74         %make a zeros vector as long as floor_exit
75         config.exit_left = zeros(1,config.exit_count);
76
77         %loop over all exits
78         for e=1:config.exit_count
79
80             %build the exit_nr matrix
81             config.exit_nr = config.exit_nr + e*( img_build(:, :, 1) == 0
82             & img_build(:, :, 2) == (256-e) & img_build(:, :, 3) == 0 )
83             ;
84
85             %build the exit_left matrix
86             config.exit_left(1,e) = config.(sprintf('exit_%d_nr', e));
87
88         end
89     end
90
91     %init the plot image here, because this won't change
92     config.floor(i).img_plot = 5*config.floor(i).img_wall ...

```

```

91     + 4*config.floor(i).img_stairs_up ...
    + 3*config.floor(i).img_stairs_down ...
    + 2*config.floor(i).img_exit ...
93     + 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 29: 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
if floor_idx~=data.floor_count
10     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 30: plotAgentsPerFloor.m

```

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

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

```

Listing 31: 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 32: 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
23 % check if whole simulation was performed
24 steps=config.duration/dt-1;
for i=1:steps
26     if agents_on_boat(i)<0
        steps=i-2;
28         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
39 % find out t10, t50, t90, t100
40 t10=0;
for i=1:steps
42     if agents_left(i)<agents_start/10
        t10=t10+dt;
44     end
    end
46 if t10~=0
    t10=t10+dt;
48 end

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

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

```

```

70 t99=0;
   for i=1:steps
72     if agents_left(i)<agents_start*0.99
         t99=t99+dt;
74     end
   end
76 if t99~=0
         t99=t99+dt;
78 end

80 t100=0;
   if agents_left==agents_start
82     for i=1:steps
         if agents_left(i)<agents_start
84             t100=t100+dt;
         end
86     end
   end
88
90 % create time axis
   if t100~=0
       time = [0:dt:t100];
92   else
       time = [0:dt:simulation_time_sim];
94   end
   steps = length(time);
96
98 % recalculate agents on boat
   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',[1:1:13],'FontSize',16)
   plot(time/60,agents_left/agents_start*100,'LineWidth', 2)
110 axis([0 13 0 100])
   title(sprintf('rescued agents (of total %i agents)',agents_start));
112 xlabel('time [min]')
   ylabel('rescued agents out of all agents [%]')
114
116 % plot agents_per_floor over time
   f2 = figure;
   hold on
118 grid on
   set(gca,'XTick',[1:1:13],'FontSize',16)

```

```

120 list = cell(config.floor_count,1);
    color = hsv(config.floor_count);
122 color(config.floor_exit,:) = [0 0 0];
    for i=1:config.floor_count
124         plot(time/60,agents_per_floor(i,:), 'LineWidth', 2, 'color', color(i,:))
            list{i} = [sprintf('floor %i',i)];
126     end
    legend(list)
128
    axis([0 13 0 1000])
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', [1:1:13], 'FontSize', 16)
    list = cell(config.exit_count/2,1);
140 color = hsv(config.exit_count/2);
    for i=1:config.exit_count/2
142         plot(time/60,exit_left(i,:), 'LineWidth', 2, 'color', color(i,:))
            list{i} = [sprintf('boat %i / --- %i',i,i+13)];
144     end
    for i=config.exit_count/2+1:config.exit_count
146         plot(time/60,exit_left(i,:), '--', 'LineWidth',
                2, 'color', color(i-config.exit_count/2,:))
    end
148 legend(list)

150 axis([0 13 0 200])
    title('rescue boat capacity');
152 xlabel('time [min]')
    ylabel('free places on rescue boat')
154
    % scale plots up to screen size
156 set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );
    set(f2, 'Position', [0 0 screen_size(3) screen_size(4) ] );
158 set(f3, 'Position', [0 0 screen_size(3) screen_size(4) ] );

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

```

```

170 fprintf('t_10: %f\n', t10)
    fprintf('t_50: %f\n', t50)
172 fprintf('t_90: %f\n', t90)
    fprintf('t_99: %f\n', t99)
174 fprintf('t_100: %f\n', t100)

```

Listing 33: plotFloor.m

```

1 function simulate(config_file)
  % run this to start the simulation
3
  % start recording the matlab output window for debugging reasons
5 diary log

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

11 fprintf('Load config file...\n');
    config = loadConfig(config_file);
13
    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
        for floor=1:data.floor_count
39         data.output.agents_per_floor(floor,data.step) =
            length(data.floor(floor).agents);
        end

```



```

41 % dump exit_left to output
42 data.output.exit_left(:,data.step) = data.exit_left';
43
44 if mod(data.step,data.save_step) == 0
45
46     % do the plotting
47     set(0,'CurrentFigure',data.figure_floors);
48     for floor=1:data.floor_count
49         plotAgentsPerFloor(data, floor);
50         plotFloor(data, floor);
51     end
52
53     if data.save_frames==1
54         print('-depsc2',sprintf('frames/%s_%04i.eps', ...
55 %             data.frame_basename,data.step), data.figure_floors);
56
57 %         make video while simulate
58         currFrame=getframe(data.figure_floors);
59         writeVideo(vidObj,currFrame);
60
61     end
62
63     set(0,'CurrentFigure',data.figure_exit);
64     plotExitedAgents(data);
65
66     if data.agents_exited == data.total_agent_count
67         fprintf('All agents are now saved (or are they?). Time: %.2f
68 %             sec\n', data.time);
69         fprintf('Total Agents: %i\n', data.total_agent_count);
70
71         print('-depsc2',sprintf('frames/exited_agents_%s.eps', ...
72 %             data.frame_basename), data.figure_floors);
73         break;
74     end
75
76     % toc of timestep
77     data.telapsed = toc(tstart);
78     % toc of whole simulation
79     data.output.simulation_time = toc(simstart);
80
81     % save output
82     output = data.output;
83     save(data.output_file_name,'output')
84     fprintf('Frame %i done (took %.3fs; %.3fs out of %.3gs
85 %         simulated).\n', data.step, data.telapsed, data.time,
86 %         data.duration);
87
88 end

```

```

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

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

99 end

101 %make video while simulation
close(vidObj);
103
    % toc of whole simulation
105 data.output.simulation_time = toc(simstart);

107 % save complete simulation
    output = data.output;
109 save('output','output')
    fprintf('Simulation done in %i seconds and saved data to output file.\n',
        data.output.simulation_time);
111
    % save diary
113 diary

```

Listing 34: simulate.m

9.1.3 C code

```

1
#include <mex.h>
3 #include <string.h>

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

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

```

```

17     if (nlhs < 1)
18         return;
19
20     points = (point_t*) mxGetPr(prhs[0]);
21     m = mxGetM(prhs[0]);
22     n = mxGetN(prhs[0]);
23
24     if (m != 2)
25         mexErrMsgTxt("...");
26
27     tree = build_tree(points, n);
28
29     plhs[0] = mxCreateNumericMatrix(tree->first_free + sizeof(int), 1,
30                                     mxUINT8_CLASS, mxREAL);
31     data = (uchar*) mxGetPr(plhs[0]);
32
33     root_index = (int*) data;
34     *root_index = tree->root_index;
35     memcpy(data + sizeof(int), tree->data, tree->first_free);
36
37     free_tree(tree);
38 }

```

Listing 35: createRangeTree.c

```

1  #include "mex.h"
2
3  #include <math.h>
4
5  #if defined __GNUC__ && defined __FAST_MATH__ && !defined __STRICT_ANSI__
6  #define MIN(i, j) fmin(i, j)
7  #define MAX(i, j) fmax(i, j)
8  #define ABS(i)     fabs(i)
9  #else
10 #define MIN(i, j) ((i) < (j) ? (i) : (j))
11 #define MAX(i, j) ((i) > (j) ? (i) : (j))
12 #define ABS(i)     ((i) < 0.0 ? -(i) : (i))
13 #endif
14
15 #define SOLVE_AND_UPDATE      udiff = uxmin - uymin; \
16                               if (ABS(udiff) >= 1.0) \
17                               { \
18                                   up = MIN(uxmin, uymin) + 1.0; \
19                               } \
20                               else \
21                               { \
22                                   up = (uxmin + uymin + sqrt(2.0 - udiff * \
23                                       udiff)) / 2.0; \
24                                   up = MIN(uij, up); \

```

```

25         } \
           err_loc = MAX(ABS(uij - up), err_loc); \
27         u[ij] = up;

29 #define I_STEP(_uxmin, _uymin, _st) if (boundary[ij] == 0.0) \
           { \
31             uij = un; \
              un = u[ij + _st]; \
33             uxmin = _uxmin; \
              uymin = _uymin; \
35             SOLVE_AND_UPDATE \
              ij += _st; \
37         } \
           else \
39         { \
              up = un; \
41             un = u[ij + _st]; \
              ij += _st; \
43         }

45

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

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

54 #define UY_RIGHT u[ij + m]
   #define UY_LEFT  u[ij - m]
55 #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;
       int m2, n2;
65     double up, un, uij, uxmin, uymin, udiff, err_loc;

67     m2 = m - 2;
       n2 = n - 2;
69

       *err = 0.0;
71     err_loc = 0.0;

73     /* first sweep */

```

```

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

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

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

91      for (i = 1; i <= m2; ++i)
          I_STEP_UP(UX_BOTH, UY_BOTH)
93
95      I_STEP_UP(UX_PREV, UY_BOTH)
    }

97  /* 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
105  /* i = m-1, j = n-1 */
    I_STEP_UP(UX_PREV, UY_LEFT)

107
    /* sweep 2 */
109  /* i = 0, j = n-1 */
    ij = (n-1)*m;
111  un = u[ij];
    I_STEP_UP(UX_NEXT, UY_LEFT)
113
    /* i = 1->m2, j = n-1 */
115  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 */
    for (j = n2; j >= 1; --j)
123  {

```

```

125     ij = j*m;
126     un = u[ij];
127     I_STEP_UP(UX_NEXT, UY_BOTH)
128
129     for (i = 1; i <= m2; ++i)
130         I_STEP_UP(UX_BOTH, UY_BOTH)
131
132     I_STEP_UP(UX_PREV, UY_BOTH)
133 }
134
135 /* i = 0, j = 0 */
136 ij = 0;
137 un = u[ij];
138 I_STEP_UP(UX_NEXT, UY_RIGHT)
139
140 /* i = 1->m2, j = 0 */
141 for (i = 1; i <= m2; ++i)
142     I_STEP_UP(UX_BOTH, UY_RIGHT)
143
144 /* i = m-1, j = 0 */
145 I_STEP_UP(UX_PREV, UY_RIGHT)
146
147 /* sweep 3 */
148 /* i = m-1, j = n-1 */
149 ij = m*n - 1;
150 un = u[ij];
151 I_STEP_DOWN(UX_NEXT, UY_LEFT)
152
153 /* i = m2->1, j = n-1 */
154 for (i = m2; i >= 1; --i)
155     I_STEP_DOWN(UX_BOTH, UY_LEFT)
156
157 /* i = 0, j = n-1 */
158 I_STEP_DOWN(UX_PREV, UY_LEFT)
159
160 /* i = m-1->0, j = n2->1 */
161 for (j = n2; j >= 1; --j)
162 {
163     I_STEP_DOWN(UX_NEXT, UY_BOTH)
164
165     for (i = m2; i >= 1; --i)
166         I_STEP_DOWN(UX_BOTH, UY_BOTH)
167
168     I_STEP_DOWN(UX_PREV, UY_BOTH)
169 }
170
171 /* i = m-1, j = 0 */
172 I_STEP_DOWN(UX_NEXT, UY_RIGHT)
173
174 /* i = m2->1, j = 0 */

```

```

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

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

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

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

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

193     /* i = m-1->0, j = 1->n2 */
195     for (j = 1; j <= n2; ++j)
    {
        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)
    }

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

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

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

217     *err = MAX(*err, err_loc);
219 }

221 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
{

```

```

223     double *u, *boundary;
224     double tol, err;
225     int m, n, entries, max_iter, i;

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

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

239

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

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

249     /* Get max iterations */
250     if (nrhs >= 2)
251     {
252         if (!mxIsDouble(prhs[1]) || mxGetM(prhs[1]) != 1 ||
253             mxGetN(prhs[1]) != 1)
254             mexErrMsgTxt("Maximum iteration needs to be positive
255                 integer.");
256         max_iter = (int) *mxGetPr(prhs[1]);
257         if (max_iter <= 0)
258             mexErrMsgTxt("Maximum iteration needs to be positive
259                 integer.");
260     }
261     else
262         max_iter = 20;

263     /* Get tolerance */
264     if (nrhs >= 3)
265     {
266         if (!mxIsDouble(prhs[2]) || mxGetM(prhs[2]) != 1 ||
267             mxGetN(prhs[2]) != 1)
268             mexErrMsgTxt("Tolerance needs to be a positive real number.");
269         tol = *mxGetPr(prhs[2]);

```



```

269         if (tol < 0)
            mexErrMsgTxt("Tolerance needs to be a positive real number.");
271     }
    else
273         tol = 1e-12;

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

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

281
    err = 0.0;
283     i = 0;
    do
285     {
        iteration(u, boundary, m, n, &err);
287         ++i;
    } while (err > tol && i < max_iter);
289 }

```

Listing 36: 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;
    double dxp, dxm, dyp, dym, xns, yns, nrm;
15     int m, n, i, j, nn;

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

23     /* Get inputs */
    if (nrhs < 2)

```

```

25     mexErrMsgTxt("At least 2 input argument needed.");
else if (nrhs > 2)
27     mexErrMsgTxt("At most 2 input argument used.");

29

31     /* Get boundary */
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]);
m = mxGetM(prhs[0]);
37     n = mxGetN(prhs[0]);

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

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

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

53

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

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

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

```

```

73         dym = DIST(i,j);
74
75         if (j < n-1)
76             dyp = INTERIOR(i,j+1) ? DIST(i,j+1) : DIST(i,j);
77         else
78             dyp = DIST(i,j);
79
80         XGRAD(i, j) = (dyp - dym) / 2.0;
81         YGRAD(i, j) = (dyp - dym) / 2.0;
82         nrm = sqrt(XGRAD(i, j)*XGRAD(i, j) + YGRAD(i, j)*YGRAD(i,
83             j));
84         if (nrm > 1e-12)
85         {
86             XGRAD(i, j) /= nrm;
87             YGRAD(i, j) /= nrm;
88         }
89     }
90     else
91     {
92         XGRAD(i, j) = 0.0;
93         YGRAD(i, j) = 0.0;
94     }
95
96     for (j = 0; j < n; ++j)
97     for (i = 0; i < m; ++i)
98         if (!INTERIOR(i, j))
99         {
100             xns = 0.0;
101             yns = 0.0;
102             nn = 0;
103             if (i > 0 && INTERIOR(i-1,j))
104             {
105                 xns += XGRAD(i-1,j);
106                 yns += YGRAD(i-1,j);
107                 ++nn;
108             }
109             if (i < m-1 && INTERIOR(i+1,j))
110             {
111                 xns += XGRAD(i+1,j);
112                 yns += YGRAD(i+1,j);
113                 ++nn;
114             }
115             if (j > 0 && INTERIOR(i,j-1))
116             {
117                 xns += XGRAD(i,j-1);
118                 yns += YGRAD(i,j-1);
119                 ++nn;
120             }
121             if (j < n-1 && INTERIOR(i,j+1))
122             {

```

```

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 37: getNormalizedGradient.c

```

2  #include <mex.h>
3
4  void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
5  {
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;
9
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;

```

```

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

42     wx1 = x - i0;
    wy1 = y - j0;
44     wx0 = 1.0 - wx1;
    wy0 = 1.0 - wy1;
46
    *out = wx0 * (wy0 * d00 + wy1 * d01) + wx1 * (wy0 * d10 + wy1 * d11);
48 }

```

Listing 38: lerp2.c

```

2  #include <mex.h>
   #include <string.h>
4
   #include "tree_build.c"
6  #include "tree_query.c"
   #include "tree_free.c"
8
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
    *prhs[])
10 {
    tree_t *tree;
12     int n, i;
    int *point_idx, *root_idx;
14     range_t *range;
    uchar *data;
16
    if (nlhs != 1)
18         mexErrMsgTxt("...");

20     if (nrhs < 5)
        mexErrMsgTxt("...");
22     else if (nrhs > 5)
        mexErrMsgTxt("...");
24

    data = (uchar*) mxGetPr(prhs[0]);
26

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

```

```

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

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

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

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

44     free_range(range);
45     free(tree);
46 }

```

Listing 39: rangeQuery.c

```

1 #ifndef TREE_H
2 #define TREE_H

4 #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 /* query a range tree */
9 range_t* range_query(tree_t *tree, double x_min, double x_max, double
10 y_min, double y_max);

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

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

16 #endif

```

Listing 40: tree.h

```

1 #ifndef TREE_BUILD_H
2 #define TREE_BUILD_H

4 #include "tree.h"

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);

```

```

8
/* double comparison for qsort */
10 int compare_double(const void *a, const void *b);

12 /* index array sorting functions, sort point index array by point y
    coordinates */
void index_sort_y(const point_t *points, int *point_idx, const int n);
14 void index_quicksort_y(const point_t *points, int *point_idx, int l, int
    r);
int index_partition_y(const point_t *points, int *point_idx, int l, int r);
16
#endif

```

Listing 41: tree_build.h

```

1
#include <assert.h>
3 include <stdio.h>
include <stdlib.h>
5 include <string.h>

7 include "tree_build.h"

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

15     /* get x coordinate values of all points */
    x_vals = (double*) malloc(n * sizeof(double));
17     for (i = 0; i < n; ++i)
        x_vals[i] = points[i].x;
19

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

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

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

```

```

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

/* sort index array by y coordinates of associated points */
43 index_sort_y(points, point_idx, n);
45

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

/* copy point coordinates to tree data */
51 memcpy(tree->data, points, n * sizeof(point_t));
53

/* set first free byte and root index of the tree */
55 tree->first_free = n * sizeof(point_t);
tree->root_index = tree->first_free;
57

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

/* free temporaries */
61 free(x_vals);
63 return tree;
}
65

67 int build_subtree(tree_t *tree, double *x_vals, const int nx, point_t
*points, int *point_idx, const int np)
{
69     int i, j, k, nx_left, np_left, node_size, right_idx;
node_t *node;
71     int *node_point_idx, *point_idx_left, *point_idx_right, node_idx;
uchar *new_data;
73
assert(nx > 0);
75 assert(np > 0);

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

```



```

85     free(tree->data);
      tree->data = new_data;
87 }
      node_idx = tree->first_free;
89 node = (node_t*) &tree->data[node_idx];
      tree->first_free += node_size;

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

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

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

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

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

123     /* fill index arrays */
        j = 0;
125     k = 0;
        for (i = 0; i < np; ++i)
127     {
            if (points[point_idx[i]].x <= node->x_val)
129                 point_idx_left[j++] = point_idx[i];
            else
131                 point_idx_right[k++] = point_idx[i];
        }
133

```

```

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

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

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

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

151 int compare_double(const void *a, const void *b)
153 {
    double ad, bd;
155     ad = *((double*) a);
    bd = *((double*) b);
157     return (ad < bd) ? -1 : (ad > bd) ? 1 : 0;
}

159 void index_sort_y(const point_t *points, int *point_idx, const int n)
161 {
    index_quicksort_y(points, point_idx, 0, n - 1);
163 }

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

169     /* quicksort point indices by point y coordinates, don't touch point
        array itself */
    while (l < r)
171     {
        p = index_partition_y(points, point_idx, l, r);
173         if (r - p > p - l)
        {
            index_quicksort_y(points, point_idx, l, p - 1);
175             l = p + 1;
177         }
        else
179         {

```

```

181         index_quicksort_y(points, point_idx, p + 1, r);
182         r = p - 1;
183     }
184 }
185
186 int index_partition_y(const point_t *points, int *point_idx, int l, int r)
187 {
188     int i, j, tmp;
189     double pivot;
190
191     /* rightmost element is pivot */
192     i = l;
193     j = r - 1;
194     pivot = points[point_idx[r]].y;
195
196     /* quicksort partition */
197     do
198     {
199         while (points[point_idx[i]].y <= pivot && i < r)
200             ++i;
201
202         while (points[point_idx[j]].y >= pivot && j > l)
203             --j;
204
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);
212
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 42: tree_build.c

```

1  #include <stdlib.h>
3
5  #include "tree.h"

```

```

void free_tree(tree_t *tree)
7 {
    free(tree->data);
9 }

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

```

Listing 43: 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, icnreases range capacity if needed */
void range_append(range_t *range, int idx);
8
/* 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 44: tree_query.h

```

1 #include <assert.h>
3 #include <stdio.h>
#include <stdlib.h>
5
#include "tree_query.h"
7
#define LEFT_CHILD_IDX(node_idx, node) (node_idx) + sizeof(node_t) +
    (node)->np * sizeof(int)
9 #define RIGHT_CHILD_IDX(node_idx, node) (node)->right_idx
#define NODE_FROM_IDX(tree, node_idx) (node_t*) &(tree)->data[node_idx];
11
range_t* range_query(tree_t *tree, double x_min, double x_max, double
    y_min, double y_max)
13 {
    int split_node_idx, node_idx;
15     node_t *split_node, *node;
    range_t *range;
17
    /* init range */

```

```

19     range = (range_t*) malloc(sizeof(range_t));
    range->min.x = x_min;
21     range->max.x = x_max;
    range->min.y = y_min;
23     range->max.y = y_max;
    range->n = 0;
25     range->total_size = 16;
    range->point_idx = (int*) malloc(range->total_size * sizeof(int));
27
    /* find split node */
29     split_node_idx = find_split_node(tree, tree->root_idx, range);
    split_node = NODE_FROM_IDX(tree, split_node_idx);
31
    /* if split node is a child */
33     if (split_node->right_idx == -1)
    {
35         range_query_y(tree, split_node_idx, range);
        return range;
37     }

39     /* follow left path of the split node */
    node_idx = LEFT_CHILD_IDX(split_node_idx, split_node);
41     node = NODE_FROM_IDX(tree, node_idx);
    while (node->right_idx != -1)
43     {
        if (range->min.x <= node->x_val)
45         {
            range_query_y(tree, RIGHT_CHILD_IDX(node_idx, node), range);
            node_idx = LEFT_CHILD_IDX(node_idx, node);
47         }
        else
49         {
            node_idx = RIGHT_CHILD_IDX(node_idx, node);
51         }
        node = NODE_FROM_IDX(tree, node_idx);
53     }
    range_query_y(tree, node_idx, range);

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

```

```

69     range_query_y(tree, node_idx, range);

71     return range;
72 }
73
74 void range_append(range_t *range, int idx)
75 {
76     int *new_point_idx;
77     int new_size, i;
78
79     /* just append if there is enough place, otherwise double capacity and
80        append */
81     if (range->n < range->total_size)
82         range->point_idx[range->n++] = idx;
83     else
84     {
85         new_size = range->total_size << 1;
86         new_point_idx = (int*) malloc(new_size * sizeof(int));
87         for (i = 0; i < range->n; ++i)
88             new_point_idx[i] = range->point_idx[i];
89         new_point_idx[range->n++] = idx;
90         free(range->point_idx);
91         range->point_idx = new_point_idx;
92         range->total_size = new_size;
93     }
94 }
95
96 int find_split_node(tree_t *tree, int node_idx, range_t *range)
97 {
98     node_t *node;
99
100     node = (node_t*) &tree->data[node_idx];
101     /* check if this node is the split node */
102     if (range->min.x <= node->x_val && range->max.x > node->x_val)
103         return node_idx;
104
105     /* ...or if it is a child (and therefor the split node) */
106     if (node->right_idx == -1)
107         return node_idx;
108
109     /* otherwise search the split node at the left or right of the current
110        node */
111     if (range->max.x <= node->x_val)
112         return find_split_node(tree, LEFT_CHILD_IDX(node_idx, node),
113                                range);
114     else
115         return find_split_node(tree, RIGHT_CHILD_IDX(node_idx, node),
116                                range);
117 }

```

```

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
123     node = (node_t*) &tree->data[node_idx];
124     points = (point_t*) tree->data;
125     point_idx = (int*) (node + 1);
126
127     /* return if all points are outside the range */
128     if (points[point_idx[0]].y > range->max.y || points[point_idx[node->np
129         - 1]].y < range->min.y)
130         return;
131
132     /* binary search for lower end of the range */
133     y = range->min.y;
134     j = 0;
135     k = node->np - 1;
136     while (j != k)
137     {
138         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]);

```

```
}
```

Listing 45: tree_query.c

```
#ifndef TREE_TYPES_H
2 #define TREE_TYPES_H

4 typedef unsigned char uchar;

6 /* 2D point */
typedef struct
8 {
    double x;
10    double y;
} point_t;

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

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

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

24     /* index of the root node in the data array*/
26     int root_index;
} tree_t;

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

34     /* number of associated points */
36     int np;

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

42 /* range */
typedef struct
44 {
    /* point index list */
```



```

46     int *point_idx;

48     /* number of saved indices */
    int n;

50

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

54     /* minimum range point */
    point_t min;

56     /* maximum range point */
    point_t max;
58     } range_t;
60
    #endif

```

Listing 46: tree_types.h

9.1.4 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
30     idx = rangeQuery(tree, pi(1) - r_max, pi(1) + r_max, ...
31                     pi(2) - r_max, pi(2) + r_max)';
32
33     % loop over agents near agent ai
34     for aj = idx
35
36         % if force has not been calculated yet...
37         if aj > ai
38             pj = data.floor(fi).agents(aj).p;
39             vj = data.floor(fi).agents(aj).v;
40             rj = data.floor(fi).agents(aj).r;
41
42             % vector pointing from j to i
43             nij = (pi - pj) * data.meter_per_pixel;
44
45             % distance of agents
46             d = norm(nij);
47
48             % normalized vector pointing from j to i
49             nij = nij / d;
50             % tangential direction
51             tij = [-nij(2), nij(1)];
52
53             % sum of radii
54             rij = (ri + rj);
55
56             % repulsive interaction forces
57             if d < rij
58                 T1 = data.k*(rij - d);
59                 T2 = data.kappa*(rij - d)*dot((vj - vi),tij)*tij;
60             else
61                 T1 = 0;
62                 T2 = 0;
63             end
64
65             F = (data.A * exp((rij - d)/data.B) + T1)*nij + T2;
66
67             data.floor(fi).agents(ai).f = ...
68                 data.floor(fi).agents(ai).f + F;
69             data.floor(fi).agents(aj).f = ...
70                 data.floor(fi).agents(aj).f - F;
71         end
72     end
73
74     % include agents on stairs!
75     if fi > 1
76         % use range tree to get the indices of all agents near agent ai
77         if ~isempty(data.floor(fi-1).agents)

```

```

79         idx = rangeQuery(tree_lower, pi(1) - r_max, ...
80                             pi(1) + r_max, pi(2) - r_max, pi(2) + r_max)';
81
82         % if there are any agents...
83         if ~isempty(idx)
84             for aj = idx
85                 pj = data.floor(fi-1).agents(aj).p;
86                 if data.floor(fi-1).img_stairs_up(round(pj(1)),
87                                                         round(pj(2)))
88
89                     vj = data.floor(fi-1).agents(aj).v;
90                     rj = data.floor(fi-1).agents(aj).r;
91
92                     % vector pointing from j to i
93                     nij = (pi - pj) * data.meter_per_pixel;
94
95                     % distance of agents
96                     d = norm(nij);
97
98                     % normalized vector pointing from j to i
99                     nij = nij / d;
100                    % tangential direction
101                    tij = [-nij(2), nij(1)];
102
103                    % sum of radii
104                    rij = (ri + rj);
105
106                    % repulsive interaction forces
107                    if d < rij
108                        T1 = data.k*(rij - d);
109                        T2 = data.kappa*(rij - d)*dot((vj -
110                                                         vi),tij)*tij;
111                    else
112                        T1 = 0;
113                        T2 = 0;
114                    end
115
116                    F = (data.A * exp((rij - d)/data.B) + T1)*nij
117                        + T2;
118
119                    data.floor(fi).agents(ai).f = ...
120                        data.floor(fi).agents(ai).f + F;
121                    data.floor(fi-1).agents(aj).f = ...
122                        data.floor(fi-1).agents(aj).f - F;
123                end
124            end
125        end
end

```

```
end
```

Listing 47: addAgentRepulsiveForce.m

```
1 function data = addDesiredForce(data)
2 %ADDDESIREDFORCE add 'desired' force contribution (towards nearest exit or
3 %staircase)
4
5 for fi = 1:data.floor_count
6
7     for ai=1:length(data.floor(fi).agents)
8
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;
14
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];
20
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 48: addDesiredForce.m

```
function data = addWallForce(data)
2 %ADDWALLFORCE adds wall's force contribution to each agent
3
4 for fi = 1:data.floor_count
5
6     for ai=1:length(data.floor(fi).agents)
7         % get agents data
8         p = data.floor(fi).agents(ai).p;
9         ri = data.floor(fi).agents(ai).r;
10        vi = data.floor(fi).agents(ai).v;
11
12        % get direction from nearest wall to agent
13        nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
14        ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
15
16        % get distance to nearest wall
```

```

18     diW = lerp2(data.floor(fi).img_wall_dist, p(1), p(2));
20     % get perpendicular and tangential unit vectors
21     niW = [ nx ny];
22     tiW = [-ny nx];
24
25     % calculate force
26     if diW < ri
27         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 end
end

```

Listing 49: addWallForce.m

```

function data = applyForcesAndMove(data)
2 %APPLYFORCESANDMOVE apply current forces to agents and move them using
  %the timestep and current velocity
4
5 n_velocity_clamps = 0;
6
7 % loop over all floors higher than exit floor
8 for fi = data.floor_exit:data.floor_count
9
10     % init logical arrays to indicate agents that change the floor or exit
11     % the simulation
12     floorchange = false(length(data.floor(fi).agents),1);
13     exited = false(length(data.floor(fi).agents),1);
14
15     % loop over all agents
16     for ai=1:length(data.floor(fi).agents)
17         % add current force contributions to velocity
18         v = data.floor(fi).agents(ai).v + data.dt * ...
19             data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
20
21         % clamp velocity
22         if norm(v) > data.v_max
23             v = v / norm(v) * data.v_max;
24             n_velocity_clamps = n_velocity_clamps + 1;
25         end
26     end
27 end

```

```

28 % get agent's new position
newp = data.floor(fi).agents(ai).p + ...
    v * data.dt / data.meter_per_pixel;

30

32 % if the new position is inside a wall, remove perpendicular
% component of the agent's velocity
34 if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
    data.floor(fi).agents(ai).r

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

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

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

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

50 end

52

54 % check if agents position is ok
% 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)))
58         newp = data.floor(fi).agents(ai).p;
        v = [0 0];
60         data.floor(fi).agents(ai).clogged =
            data.floor(fi).agents(ai).clogged + 1;
        fprintf('WARNING: clogging agent %i on floor %i (%i).
            Position
            (%f,%f).\n',ai,fi,data.floor(fi).agents(ai).clogged,newp(1),newp(2))
62         if data.floor(fi).agents(ai).clogged >= 40
            nx = rand(1)*2 - 1;
64             ny = rand(1)*2 - 1;
            n = [nx ny];
66             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))

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

```

```

72         if isnan(newp)
73             % get rid of disturbing agent
74             fprintf('WARNING: position of an agent is NaN!
              Deleted this agent.\n')
              exited(ai) = 1;
76             data.agents_exited = data.agents_exited +1;
              data.output.deleted_agents=data.output.deleted_agents+1;
78             newp = [1 1];
              end
80         end
              end
82     else
              % get rid of disturbing agent
84             fprintf('WARNING: position of an agent is NaN! Deleted this
              agent.\n')
              exited(ai) = 1;
86             data.agents_exited = data.agents_exited +1;
              data.output.deleted_agents=data.output.deleted_agents+1;
88             newp = [1 1];
              end
90
92     % update agent's velocity and position
              data.floor(fi).agents(ai).v = v;
94             data.floor(fi).agents(ai).p = newp;
96
97     % reset forces for next timestep
              data.floor(fi).agents(ai).f = [0 0];
98
99     % check if agent reached a staircase down and indicate floor change
100     if data.floor(fi).img_stairs_down(round(newp(1)), round(newp(2)))
              floorchange(ai) = 1;
102     end
103
104     % check if agent reached an exit
105     if data.floor(fi).img_exit(round(newp(1)), round(newp(2)))
106         exited(ai) = 1;
              data.agents_exited = data.agents_exited +1;
108
109     %
              fprintf('agent exited from upper loop\n');
110
111     %save current exit nr
              data.current_exit = data.exit_nr(round(newp(1)),
              round(newp(2)));
112
113     %
              fprintf(int2str(data.current_exit));
114
115     %update exit_left
              data.exit_left(1,data.current_exit) =
              data.exit_left(1,data.exit_nr(round(newp(1))),

```

```

118         round(newp(2)))) - 1;
119
120     %close exit if there is no more free space
121     if data.exit_left(1,data.current_exit) < 1
122
123         %change current exit to wall
124         data.floor(data.floor_exit).img_wall =
125             data.floor(data.floor_exit).img_wall == 1 ...
126             | (data.exit_nr == (data.current_exit));
127         data.floor(data.floor_exit).img_exit =
128             data.floor(data.floor_exit).img_exit == 1 ...
129             & (data.exit_nr ~= (data.current_exit));
130
131         %redo initEscapeRoutes and initWallForces with new exit
132         and wall parameters
133         data = initEscapeRoutes(data);
134         data = initWallForces(data);
135
136     %
137         fprintf('new routes from upper loop\n');
138
139     end
140 end
141
142 % add appropriate agents to next lower floor
143 if fi > data.floor_exit
144     data.floor(fi-1).agents = [data.floor(fi-1).agents
145         data.floor(fi).agents(floorchange)];
146 end
147
148 % delete these and exited agents
149 data.floor(fi).agents = data.floor(fi).agents(~(floorchange|exited));
150 end
151
152 % loop over all floors lower than exit floor
153 for fi = 1:data.floor_exit
154
155     % init logical arrays to indicate agents that change the floor or exit
156     % the simulation
157     floorchange = false(length(data.floor(fi).agents),1);
158     exited = false(length(data.floor(fi).agents),1);
159
160     % loop over all agents
161     for ai=1:length(data.floor(fi).agents)
162         % add current force contributions to velocity
163         v = data.floor(fi).agents(ai).v + data.dt * ...

```



```

164         data.floor(fi).agents(ai).f / data.floor(fi).agents(ai).m;
165
166     % clamp velocity
167     if norm(v) > data.v_max
168         v = v / norm(v) * data.v_max;
169         n_velocity_clamps = n_velocity_clamps + 1;
170     end
171
172     % get agent's new position
173     newp = data.floor(fi).agents(ai).p + ...
174         v * data.dt / data.meter_per_pixel;
175
176     % if the new position is inside a wall, remove perpendicular
177     % component of the agent's velocity
178     if lerp2(data.floor(fi).img_wall_dist, newp(1), newp(2)) < ...
179         data.floor(fi).agents(ai).r
180
181         % get agent's position
182         p = data.floor(fi).agents(ai).p;
183
184         % get wall distance gradient (which is of course perpendicular
185         % to the nearest wall)
186         nx = lerp2(data.floor(fi).img_wall_dist_grad_x, p(1), p(2));
187         ny = lerp2(data.floor(fi).img_wall_dist_grad_y, p(1), p(2));
188         n = [nx ny];
189
190         % project out perpendicular component of velocity vector
191         v = v - dot(n,v)/dot(n,n)*n;
192
193         % get agent's new position
194         newp = data.floor(fi).agents(ai).p + ...
195             v * data.dt / data.meter_per_pixel;
196     end
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;

```

```

210         n = [nx ny];
211         v = n*data.v_max/2;
212         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))

214         % get agent's new position
215         newp = data.floor(fi).agents(ai).p + ...
216         v * data.dt / data.meter_per_pixel;
217         if isnan(newp)
218             % get rid of disturbing agent
219             fprintf('WARNING: position of an agent is NaN!
                    Deleted this agent.\n')
220             exited(ai) = 1;
221             data.agents_exited = data.agents_exited +1;
222             data.output.deleted_agents=data.output.deleted_agents+1;
223             newp = [1 1];
224         end
225     end
226 end
227 else
228     % get rid of disturbing agent
229     fprintf('WARNING: position of an agent is NaN! Deleted this
            agent.\n')
230     exited(ai) = 1;
231     data.agents_exited = data.agents_exited +1;
232     data.output.deleted_agents=data.output.deleted_agents+1;
233     newp = [1 1];
234 end

236 % update agent's velocity and position
237 data.floor(fi).agents(ai).v = v;
238 data.floor(fi).agents(ai).p = newp;

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 %         fprintf('agent exited from lower loop\n');
254
255 %save current exit nr

```

```

256         data.current_exit = data.exit_nr(round(newp(1)),
            round(newp(2)));

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

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

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

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

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

276         end

278     end
280     end

282     % add appropriate agents to next lower floor
    if fi < data.floor_exit
        data.floor(fi+1).agents = [data.floor(fi+1).agents ...
284                                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))
292     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;
294     data = initEscapeRoutes(data);
        data = initWallForces(data);
296     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 50: 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)
23         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 51: checkForIntersection.m

```

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

```

Listing 52: 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 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

27       % init agent
       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

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

```

```

        tries = tries - 1;
43     end
    if tries > -1
45         %remove the last agent
        data.floor(i).agents = data.floor(i).agents(1:end-1);
47     end
    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
59     error('no spots to place agents!');
end
61 end
end

```

Listing 53: initAgents.m

```

function data = initEscapeRoutes(data)
2 %INITESCAPEROUTES Summary of this function goes here
% Detailed explanation goes here
4
for i=1:data.floor_count
6
    boundary_data = zeros(size(data.floor(i).img_wall));
8    boundary_data(data.floor(i).img_wall) = 1;

10 if i<data.floor_exit
    boundary_data(data.floor(i).img_stairs_up) = -1;
12
14 elseif i>data.floor_exit
    boundary_data(data.floor(i).img_stairs_down) = -1;

16 else
    boundary_data(data.floor(i).img_exit) = -1;
18
end
20 exit_dist = fastSweeping(boundary_data) * data.meter_per_pixel;
    [data.floor(i).img_dir_x, data.floor(i).img_dir_y] = ...
22     getNormalizedGradient(boundary_data, -exit_dist);
end

```

Listing 54: initEscapeRoutes.m

```

function data = initialize(config)
2 % initialize the internal data from the config data
%
4 % arguments:
%   config      data structure from loadConfig()
6 %
% return:
8 %   data      data structure: all internal data used for the main loop
%
10 %           all internal data is stored in pixels NOT in meters

12
data = config;
14
%for convenience
16 data.pixel_per_meter = 1/data.meter_per_pixel;

18 fprintf('Init escape routes...\n');
data = initEscapeRoutes(data);
20 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
44     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

```

```

46         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]);
48     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]);
50     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]);
52     end
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
data.output_file_name = ['output_' data.frame_basename];
62
% prepare video file name
64 data.video_file_name = ['video_' data.frame_basename '.avi'];

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

```

Listing 55: initialize.m

```

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

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

14    % get normalized wall distance gradient
    [data.floor(i).img_wall_dist_grad_x, ...
16     data.floor(i).img_wall_dist_grad_y] = ...

```



```

    getNormalizedGradient(boundary_data, wall_dist-data.meter_per_pixel);
18 end

```

Listing 56: initWallForces.m

```

1 function config = loadConfig(config_file)
  % load the configuration file
3  %
  % arguments:
5  %   config_file      string, which configuration file to load
  %
7
9  % get the path from the config file -> to read the images
  config_path = fileparts(config_file);
11 if strcmp(config_path, '') == 1
    config_path = '.';
13 end
15 fid = fopen(config_file);
  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);
41
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 ...
53                                   & img_build(:, :, 3) == 0);

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

59
61     config.floor(i).img_stairs_down = (img_build(:, :, 1) == 0 ...
                                           & img_build(:, :, 2) == 0 ...
63                                           & img_build(:, :, 3) == 255);

65     if i == config.floor_exit

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

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

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

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

85         end
87     end

   %init the plot image here, because this won't change
89     config.floor(i).img_plot = 5*config.floor(i).img_wall ...
       + 4*config.floor(i).img_stairs_up ...
91     + 3*config.floor(i).img_stairs_down ...
       + 2*config.floor(i).img_exit ...

```

```

93         + 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
    for i=1:config.open_second_nr
101         config.open_second(i)=config.(sprintf('open_second_%i', i));
    end
103
    % save the "all exits open" configuration
105 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
109 for i=1:config.open_second_nr
    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 57: 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
    if floor_idx~=data.floor_count
10         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 58: plotAgentsPerFloor.m

```

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

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

```

Listing 59: 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 60: 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
26     if agents_on_boat(i)<0
         steps=i-2;
28     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
42     if agents_left(i)<agents_start/10
         t10=t10+dt;
44     end
  end
  if t10~=0
46     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 t99=0;
71 for i=1:steps
72     if agents_left(i)<agents_start*0.99
73         t99=t99+dt;
74     end
75 end
76 if t99~=0
77     t99=t99+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];
93 end
94 steps = length(time);
95
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', [1:1:13], 'FontSize', 16)
plot(time/60, agents_left/agents_start*100, 'LineWidth', 2)
110 axis([0 13 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
118 set(gca, 'XTick', [1:1:13], 'FontSize', 16)
list = cell(config.floor_count, 1);
color = hsv(config.floor_count);
122 color(config.floor_exit, :) = [0 0 0];
for i=1:config.floor_count
124     plot(time/60, agents_per_floor(i,:), 'LineWidth', 2, 'color', color(i,:))
list{i} = [sprintf('floor %i', i)];
126 end
legend(list)
128
axis([0 13 0 1000])
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', [1:1:13], '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,:))
142 list{i} = [sprintf('boat %i / --- %i', i, i+13)];
144 end
for i=config.exit_count/2+1:config.exit_count
146     plot(time/60, exit_left(i,:), '--', 'LineWidth',
2, 'color', color(i-config.exit_count/2,:))

```

```

end
148 legend(list)

150 axis([0 13 0 200])
    title('rescue boat capacity');
152 xlabel('time [min]')
    ylabel('free places on rescue boat')
154
% scale plots up to screen size
156 set(f1, 'Position', [0 0 screen_size(3) screen_size(4) ] );
    set(f2, 'Position', [0 0 screen_size(3) screen_size(4) ] );
158 set(f3, 'Position', [0 0 screen_size(3) screen_size(4) ] );

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

170 fprintf('t_10: %f\n', t10)
    fprintf('t_50: %f\n', t50)
172 fprintf('t_90: %f\n', t90)
    fprintf('t_99: %f\n', t99)
174 fprintf('t_100: %f\n', t100)

```

Listing 61: plotFloor.m

```

1 function simulate(config_file)
    % run this to start the simulation
3
    % start recording the matlab output window for debugging reasons
5 diary log

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

11 fprintf('Load config file...\n');
    config = loadConfig(config_file);
13
    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
    for floor=1:data.floor_count
39         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
        set(0,'CurrentFigure',data.figure_floors);
49         for floor=1:data.floor_count
            plotAgentsPerFloor(data, floor);
51             plotFloor(data, floor);
        end

53         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

```

```

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

71     print('-depsc2', sprintf('frames/exited_agents_%s.eps', ...
                               data.frame_basename), data.figure_floors);
73     break;
    end
75
    % toc of timestep
77     data.telapsed = toc(tstart);
    % toc of whole simulation
79     data.output.simulation_time = toc(simstart);

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

85
    end
87
    % update step
89     data.step = data.step+1;

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

99 end

101 %make video while simulation
    close(vidObj);
103
    % toc of whole simulation
105     data.output.simulation_time = toc(simstart);

107 % save complete simulation
    output = data.output;
109     save('output', 'output')
    fprintf('Simulation done in %i seconds and saved data to output file.\n',
            data.output.simulation_time);
111
    % save diary
113     diary

```

Listing 62: simulate.m