ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Emergency Evacuation of an Airbus A380

Marco Denuder & Dominik Keusch & Laurent Roux

Zurich
May 2010

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Gruppenarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilsmittel verwenden habe, und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Gruppenarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfung ausgefertigt wurde.

Marco Denuder Dominik Keusch Laurent Roux

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.

Marco Denuder Dominik Keusch Laurent Roux

Contents

1 Individual Contributions

The distribution of our work was mainly influenced by the fact that Marco had to join a refresher course at the Swiss Army in April. Therefore, his effort was mostly done during the first part of our project. He started by searching the internet for reliable airplane plans to get to know the seat configuration and afterwards set up a *Matlab* scenario of the whole plane. After several discussions with the group, on how to proceed, he implemented the first model and set the base of our project. During Marco's absence, Laurent and Dominik took over Marco's work and continued the project. As the first model was quite limited in its flexibility, we decided to implement another model, namely the one with static field. Dominik changed the existing airplane model according to the new requirements. He was engaged in searching algorithms to compute a static field of the plane and afterwards implemented one of them in *Matlab*. Laurent then, implemented the new model functions in *Matlab* and changed the old simulation in order to make it work with the static field. In the phase of optimization we collaborated a lot and helped each other. In addition we discussed the results and how to interpret them. Having done the main part of the project, we started writing our report. The individual responsibility on writing the different parts is listed below. Laurent then implemented our writing in LaTeX form.

- Individual Contributions → Dominik
- Introduction and Motivation → Dominik
- Description of the Model
  - General description → Dominik
  - Model 1 → Marco
  - Model 2 → Dominik
- Implementation
  - Model 1 → Marco
  - Model 2 → Laurent
  - Static Field → Dominik
- Simulation Results and Discussion
  - Number of passengers → Laurent
  - Exits variation → Laurent (with static field of Dominik)

  - Obstacles → Laurent
  - Probability → Laurent
  - Discussion → Laurent
- Summary and Outlook → Marco
- References → all
- Implementation in LaTeX → Laurent

2 Introduction and Motivations

The airplane-security is an important issue on aircraft engineering. There are specific laws about how fast an evacuation of an airplane has to occur. Those laws have to be observed; otherwise, the airplane does not get the permission to fly. Additionally, the new airplanes are bigger, carrying more and more people. Therefore, evacuation scenarios are growing in their complexity and it is difficult to estimate, how the evacuation takes place. This simulation should help the engineer to get a better understanding of evacuation situations in airplanes. With certain assumptions taken, it shows the process of evacuation, plus how different parameters influence the evacuation time.

The following report is part of the lecture "Modelling and Simulating Social Systems in *MATLAB*" at ETH Zurich. The course involves different aspects of modelling and simulating social systems, as well as implementing them in a computational environment using Mathworks *MATLAB*.

3 Description of the Model

3.1 General description

The physical appearance of the model is given by the dimensions and seat configurations of the Airbus A380. However, every airline has the opportunity to design its own seat configuration. Thus, it is difficult to find any exact plans of the interior. After some research on the web, we decided to take the interior design according to figure 1 on page 9.

The Airbus A380 has got 2 different decks: the upper deck and the lower deck. They differ in the interior design and the seat configuration, as well as in the possible number of passengers. Therefore, the two decks are treated separately. The most important difference between the two decks is the number of exits. The upper deck has got 8 emergency exits, whereas the lower deck has got 10 of which. In order to obtain a convenient model, there were some assumptions to be taken. These assumptions are listed below. Their purpose is to minimize the complexity of the model, since a model is always a simplification of the reality.

- People take the nearest possible exit: We assume that in case of emergency, people always choose the nearest exit location. In reality, the choice of the exit is much more complex, since it is also influenced by factors such as crowd behaviour, panic, signalisation, etc.
- Airplane model simplified: Our airplane is extremely simplified, as we are limited by the methods of implementation. Therefore, all of our passenger seats commensurate. Also, the space between the corridors is not exact. The width of the lower and upper decks is different, the seats of the crew are neglected, as well as the toilets and structural elements.
- People act with a certain probability: As an assumption, we suggest that people act with a defined probability. Statistically, this probability can be interpreted as various factors, such as intelligence, age, or the passenger's disorientation.
- Upper and lower deck evacuate separately: This means, that there is no connection between the two decks. People can't use the stairs.
- People start evacuating immediately: The passengers do not hesitate or try to organize themselves when the case of emergency happens. They all start the evacuation at the same time and without individual delay.
- All people behave the same way: Our models treat every passenger the same way. They don not differ between age, speed, size, or enthusiasm of the individual. Thus, it can be seen as a statistical average of all people.

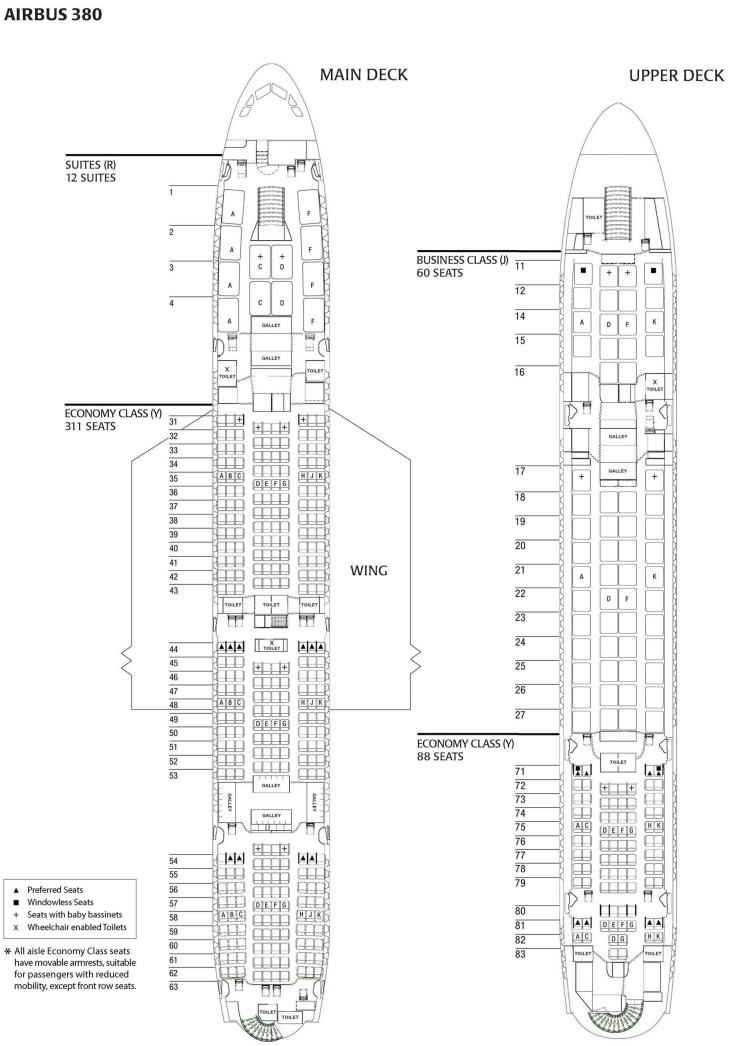

Figure 1: Airbus A380 interior design reference

- People are not climbing over seats: We assume that the passengers are not affected by panic behavior, so they do not climb over seats, but behave respectably. Also, it is quicker to run through the aisle instead of climbing over some seats. Consequently, the assumption is quite rational.

As we were not fully satisfied with the first model, we decided to implement another one. The two models differ slightly, so they are treated separately. Both models use the so-called von Neumann neighbourhood type. This means, that the

four cells orthogonally surrounding a central cell are defined as a neighbour of the central cell.

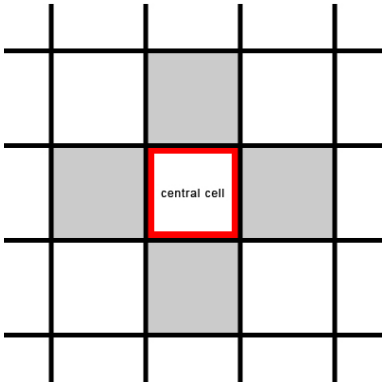

Figure 2: von Neumann neighbourhood: the grey cells are the neighbours of the central cell marked with red

3.2 Model 1

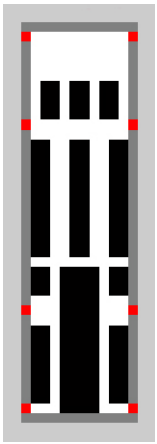

Figure 3: Seat configuration, Model 1

Behavior: At the beginning of every simulation step, a random passenger is chosen. All the action occurs on that specific passenger until the next simulation step chooses another passenger. At first, the program detects the passenger's location. Secondly, the passenger's minimal distance to the nearest exit is being calculated. Afterwards, the passenger moves, based on a specific algorithm (see implementation), and updates

his matrix field, as well as the matrix fields of his surroundings. There are four different functions which lead the passenger to the nearest exit, depending whether the nearest exit is located above or below, as well as on the right or on the left side of the passenger. To include the various factors, which are mentioned earlier, the choice of the passenger's next location comprises a probability factor. Further details are provided in the implementation section.

3.3 Model 2

For this model, we had to slightly change the seat configuration. In fact, we inserted some free space between each seat row, where the passengers can move. This change was necessary for the new implementation with the static field.

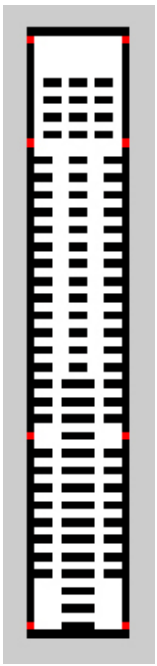

Figure 4: Seat configuration, Model 2

Behaviour: At first, all the passengers stand up, and by doing so, switch to the free space in front of their seat. Now, they all stand on a field. Each field contains a number with the minimum distance to the nearest exit. The only thing the algorithm has to do, is to move the passenger to a field with a smaller number. Mathematically spoken, the passenger moves in the direction of the static field's gradient. This step is done with a certain probability. The probability defines whether to take the best

or the worst opportunity to move. The static field is calculated in a previous step and handed over in a matrix to the main function.

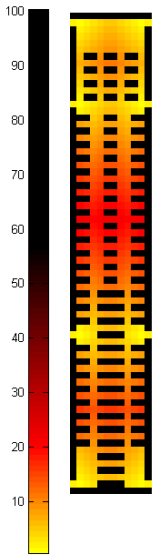

Figure 5: Static field

4 Implementation

4.1 Model 1 - Implementation

After creating a model of both decks, the idea was to create a mathematical image of the model and change it with every iteration step. Due to the fact of having two decks, each function has been written according to the choice of the deck. Some functions have even been written for only one deck. There is one main-function containing several sub-functions to perform the evacuation. All functions will shortly be explained. For further information, check the source code section starting on page 37.

4.1.1 Function *make_upper_level.m* and *make_lower_level.m*

The different items such as walls, exits, free space, occupied space, free seats, and occupied seats, are only distinct by different numbers. The function to create those numbers is called *make_upper_level* respectively *make_lower_level*.

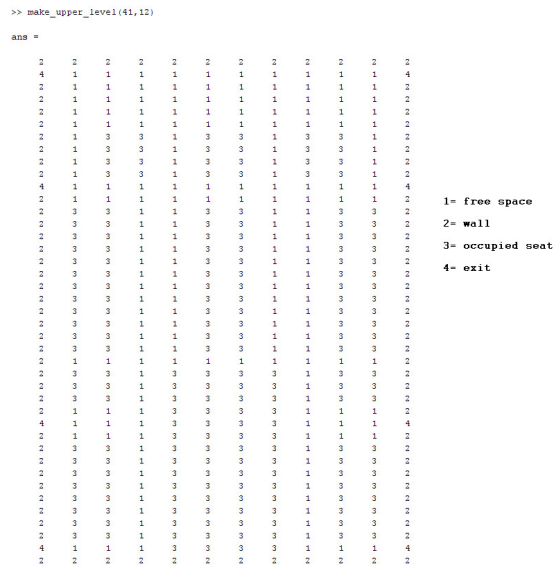

Figure 6: Numerical implementation of the initial situation

The numbers in figure 6 on page 13 are called the coordinate's state.

4.1.2 Function *upper_level_sketch.m* and *lower_level_sketch.m*

Every state has been given a different colour in order to visualize the different states, and to distinguish between passengers, walls, seats and so on. The functions for this step have been named *upper_level_sketch* and *lower_level_sketch*.

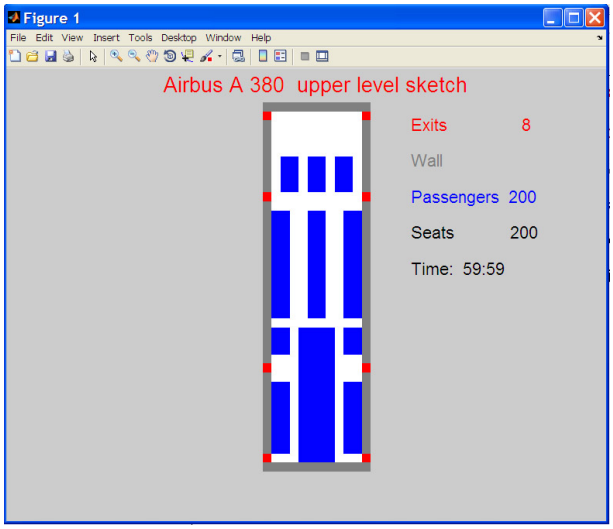

Figure 7: Visualization of the initial states using the *level_sketch function*

4.1.3 Function *ploc.m*

Every cell in the model has its own coordinates separated in length- and width-position. Therefore, it is easy to determine locations and distances between two points. The general idea is, to pick a random passenger, check his location, find out its nearest exit location by calculating the distances between the passenger and all exit-locations, pick the nearest exit, and finally lead him to that exit. How the passenger is led to the nearest exit will be explained later. At first, a short inside on how the passenger's coordinates are calculated. (The coordinates of the exits are calculated analogously). Now, the exit and passenger coordinates have been calculated and can be compared.

```
1 function [passenger_location] = ploc(length_,width,passenger_number,level)
2 % this function calculates the passenger's coordinates
3 passenger_location = ones(passenger_number,2); % default
4 k=1;l=1;
5 for i = 1:length_   % iterate over the whole length
6    for j = 1:width % iterate over the whole width
7         if  level(i,j)==3 % if there is a passenger
```

```
8                passenger_location(k,1) = i; % safe length-coordinate
9                passenger_location(l,2) = j; % safe width-coordinate
10               k = k+1; % choose next passenger's length-coordinate
11               l=l+1;   % choose next passenger's width-coordinate
12           end
13          j=j+1;
14       end
15       i = i+1;
16   end  % stop when all passenger coordinates have been stored
```

4.1.4 Function *min_distance.m*

The function *min_distance.m* measures the distance to the nearest exit of a random passenger as well as the location of his nearest exit by applying the Pythagorean Theorem: nearest line between 1 and 2 in a Cartesian coordinate system equals $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

```
1  delta=1000*ones(1,2); % default
2  k=1;
3       for j = 1:length_(exit_location) % Pythagoras for all exits
4            delta(k,1) = abs(passenger_location(1,1)-exit_location(j,1));
5            delta(k,2) = abs(passenger_location(1,2)-exit_location(j,2));
6            deltanorm(k) = sqrt(delta(k,1)^2+delta(k,2)^2);
7            j=j+1;
8            k=k+1;
9       end
10      [min_dist,index]= min(deltanorm); % choose minimum
11      % calculate nearest exit coordinates
12      nearest_exit_location(1,1) = exit_location(index,1);
13      nearest_exit_location(1,2) = exit_location(index,2);
```

4.1.5 Function *passenger_move.m*

Knowing the target location and the starting location, the passenger is led to the exit with the following function, which depends on several conditions and differs in some aspects considering diverse starting situations: function *passenger_move.m*

1. passenger is stuck on the left side
2. passenger is stuck on the right side
3. passenger is on free space and the nearest exit location is below
4. passenger is on free space and the nearest exit location is above
5. passenger is on a seat in the front row
6. passenger is on a seat and the nearest exit location is on his right

7. passenger is on a seat and the nearest exit location is on his left
8. passenger is on the same height as the exit, exit is on his right
9. passenger is on the same height as the exit, exit is on his left

The next step to take in each situation is listed in the table 1 on page 16

Situation: Passenger is (on)	Priority 1	Priority 2	Priority 3
Stuck on the left side	twice right	-	-
Stuck on the right side	twice left	-	-
Free space / exit is below	Down	Left / right	stay
Free space / exit is above	Up	Left / right	stay
Front seat	Up	-	-
Seat / exit is right	50% right	50% left	stay
Seat / exit is left	50% left	50% right	stay
Same height as exit , exit is left	Left	-	-
Same height as exit , exit is right	Right	-	-

Table 1: Next step algorithm of the function *passenger_move* (for details check source code)

Of course, the passenger can only move, if the space he wants to go to is free. To check the availability of the next location, its state is always being checked. Also, the "old" passenger location's state has to be updated. However, that is not done by this function but with the evacuation function that will be explained later on.

To check the states of the passenger's neighbour cells, and add probabilities, the following approach has been used:

```
function[new_location]= passenger_move(passenger_location,...
        nearest_exit_location,level)

% This function calculates the next step for a passenger who wants
% to reach the nearest exit at all costs!

% all possible neighbour cells a passenger can move to in one step
% meaning: [up;right;down;left;stay]
neighbour=[-1 0; 0 1; 1 0; 0 -1; 0 0];

for i = 1:length(neighbour)
 new_location(i,:) = passenger_location+neighbour(i,:);
 % safe all possible new_locations of the passenger
end

prob = rand; % variable to add probabilities
```

4.1.6 Function *evacuatioin.m*

```
% level = [1,2] visual =[0,1] movie=[0,1] meaning: 0 = no 1 = yes
```

All preceding functions are required to safely lead one random passenger one step nearer to his nearest exit. The main-function *evacuatioin.m* ensures that all passengers eventually reach the exit. Details are listed in the source code section. However, to shortly introduce the function, some main tasks have to be mentioned:

- choosing: the level / visualization (on/off) / record movie (on/off)
- measure the evacuation-time
- remove passenger that has reached the exit
- updating the new states of all fields
- invoke all preceding functions in the right order

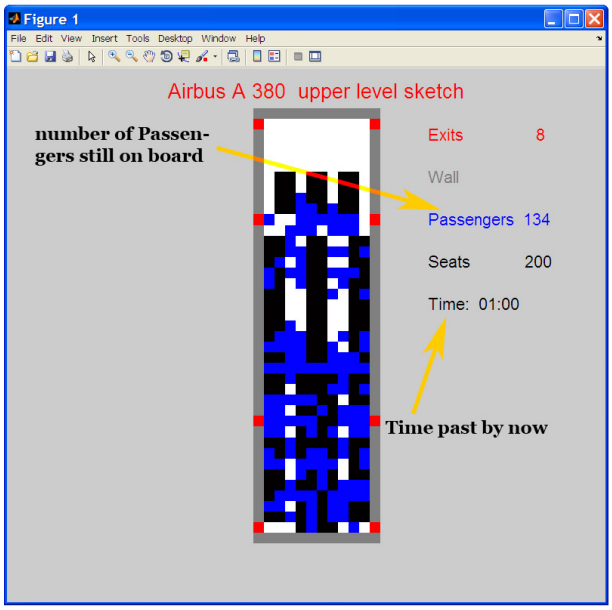

Figure 8: Screenshot of an emergency evacuation simulation

4.2 Model 2 - General description

Model 2 was implemented in *Matlab*. The main function is called *evac_V5.m* as shown on page 47. It takes two input parameters to decide whether the simulation has to visualized and a movie is to be recorded or not. See table 2 on page 18 for the exact values for each case. The function calls several sub functions, called in the *SIMULATION* part of the program.

Movie	Visualize: yes	Visualize: no
Yes	evac_V5(1,1)	evac_V5(0,1)
No	evac_V5(1,0)	evac_V5(0,0)

Table 2: Input parameters for the function *evac_V5.m*

In the *POST PROCESSING* section, time measurement settings are initialized and allocated. The variable *level* was introduced to facilitate a two level simulation in the future. The only requirement would be the mapping of the first, respectively second floor and running the program for each level. The *SIMULATION* section is the heart of the program. First of all the function *initialize.m* is called, which builds the map of the cabin according to the static field. The static field is created externally and needs to be in the same folder as the main program. In the next step, the position of each passenger and exit is evaluated. If visualization is required, the function *field_upper_level_sketch.m* draws the frame of the actual situation in the plane. Finally, the emergency situation starts and the passengers get up with the function *paxGetUp.m*. The for-loop starting at line 65 of prgram 8 calculates the evolution of the simulation for each time step. In the *POST PROCESSING* section one can choose how many passengers move per time step. A check is required to know whether there are still enough passengers in the plane or not, which is done with a simple *if* condition. Should there be less passengers to move in the plane, the number of passengers moving per time step is set to the number of agents remaining in the cabin. The matrix *paxLocations* contains the x and y coordinates of each passenger. The arrangement is purely random. With *paxMove.m* the passenger with coordinate x and y is moved. More details of the function will follow later in the text. After moving the chosen number of agents, the number of remaining passengers, their location and the elapsed time has to be computed. If required, a new frame will be saved and the loop repeated until the plane is empty or the maximal simulation time has been reached.

4.3 Model 2 - Function descriptions

Subsequently, the different functions are extensively and sequentially explained according to the main program.

4.3.1 Function *initialize.m*

In this function, the cabin is built according to the geometric information in the static field (see listing 10 on page 50). To build the interior of the airplane the function *field_make_upper_level.m* is called. It takes the length and width of the plane an positions the seats with passengers, walls and exits. Empty seats could be set as well. In a double *for* loop the the number of passengers and exits are being count. The function returns the number of passengers and exits, length and width of the cabin and the matrix with the cabin information. This matrix contains the location of seats, passengers, walls and exits.

4.3.2 Function *paxLoc.m*

Listing 11 on page 51 shows the function *paxLoc.m* which returns a matrix with the x and y coordinates of each passenger and exit. In the matrix containing the cabin information, passengers are described with a 3 and exits with a 4. This function simply returns the x and y coordinate of each passenger in a random order. The *Matlab* function *randperm(n)* returns a random permutation of the integers 1:n. These values are required to copy the data content of the position matrix into a new randomly sorted matrix, which is returned.

4.3.3 Function *paxGetUp.m*

This function, to be found in listing 13 on page 52, replaces each value in front of a seat (with seated passenger) with a passenger (3) and replaces its original place in the cabin information matrix by a seat (5). It returns the updated cabin information matrix and a count value, which can be used for double check purposes while debugging the program.

4.3.4 Function *paxMove.m*

Listing 9 on page 49 shows the most important of all sub functions. In a first step, the neighbors are being defined. For an airplane evacuation, von Neumann neighbors are useful because the agents are not allowed to jump diagonally over the seats. The *tmp* vector is required to check later on, whether the agent is doing his best move or not. The first *for* loop iterates over all neighbors of the particular passenger and

remembers the value of the static field, if the cell is available. If a cell is a seat, wall or other passenger, the agent is not allowed to move to it. If the cell is good to go, the value of the static field is saved in the *tmp* vector. This is required to choose the best movement for each passenger. If one neighbor is an exit (4), the value of the passenger's position in the cabin information matrix is set to one (free space) and the function is left. In a second step, the probability of the best move is set. In our case it is 75%. If the probability is higher than 75% the passenger moves towards the worst direction, according to the *tmp* vector. If the probability is within the 75% range, the agent does its best possible move, also according to the *tmp* vector. The function returns the updated cabin information matrix.

4.3.5 Function *field_make_upper_level.m*

Listing 15 on page 53 shows the function *paxLoc.m* which sets the correct value according to table 3 on page 20 into the cabin information matrix and returns it.

Number in matrix		Physical meaning	Color
1	=	free space	white
2	=	wall	grey
3	=	passenger	blue
4	=	exit	red
5	=	empty seat	black

Table 3: Nomenclature in cabin information matrix

4.3.6 Function *field_upper_level_sketch.m*

Here, the cabin is drawn according to the geometric information in the cabin information matrix (see listing 12 on page 51). The matrix fields are colored according to the value in table 3 on page 20.

4.4 Static field

The implementation of Model 2 requires a matrix where every cell contains the shortest distance value to its nearest exit. As of now, this matrix is called static field matrix.

Figure 9 on page 21 shows, how the final result should look like. The red cells represent the exits and the black cells stand for the walls and seats. Those black cells are treated like obstacles. So, their distance value approaches infinity. To compute

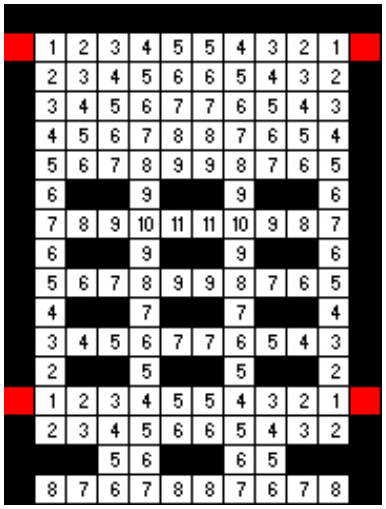

Figure 9: Static field matrix, final result

the static field matrix, we used the Floyd-Warshall algorithm. It is a "graph analysis algorithm for finding shortest paths in a weighted graph" [1].

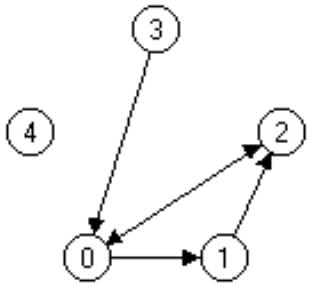

Figure 10: General path-finding problem

At first, we created a new matrix B with dimension $(l \cdot w) \times (l \cdot w)$, whereas l is the length of the airplane and w the width. This matrix is needed to save the distance between the cells (i,k) (i [1..l], k [1..w]) and all the other cells. All cell values are set to 100'000. Then, we took the l x w matrix "upper_level", which contains the plan of the airplane, and inserted each cell's neighbor information into the matrix B. Finally, we applied the Floyd-Warshall algorithm to the matrix B.

```
1
2 for k=1:(l*w)
3     for i=1:(l*w)
4         for j=1:(l*w)
5             B(i,j)=min([B(i,j);(B(i,k)+B(k,j))]);
6         end
7     end
8 end
```

The algorithm iterates over all cells, columns and rows of the matrix B and stores the minimum distance. As a result, we got the matrix B, containing the minimal distance between each pair of cells. The matrix B is a (l*w) x (l*w) matrix. Therefore, we had to reduce it by choosing an exit and listing the distances to the other cells in a l*w matrix. This process was performed with every exit, resulting a matrix C of size $l \cdot w \cdot$(number_of_exits). To obtain the static field, those (number_of_exits) matrices are put together by always taking the minimum value of the cells. The efficiency of our implementation is rather low. The computation of the static field matrix increases with $\mathcal{O}((I \cdot w)^3)$. However, we did not invest more time on enhancing it, because that is not the main part of our project.

5 Simulation Results and Discussion

5.1 Validation

A model validation would be done with measurement data of full scale tests. Since any full scale data is not available, validation is left open as a future step.

5.2 Post processing approach

The analysis was performed with 100 simulations per set of parameters. The mean values and standard deviations are plotted in the diagrams and listed in the tables. The terms *normal conditions* or *normal configurations* are used for simulations without additional obstacles in the cabin using the usual location of the emergency exits. An optimized version was simulated with a different placement of the exits. The number of exits remains the same. In a second approach, obstacles were implemented in the region of the middle exits. The time step is equal to the time each figure remains on the screen, while the simulation is plotted. The probability is explained in paragraph 4.3.4.

5.3 Simulation without obstacles around doors

Figures 11 and 12 on page 24 show the simulation domain without obstacles in normal and optimized configuration.

5.3.1 Simulation with different parameters under normal condition

Figure 13 on page 25 shows the different evacuation times by changing the number of passengers under normal condition. The parameters are the number of passengers per time step moving and their probability to do the best move. The more passengers are moving per time step the lower the evacuation time. With high probability of doing the best possible move, the standard deviation for each measurement decreases.

Plot 14 on page 25 shows the mean evacuation time versus the probability of each passenger doing the best possible move. As expected, the mean evacuation time, as well as the corresponding standard deviation is diametrically opposed to the probability.

5.3.2 Simulation with different parameters and optimized location of emergency exits

Figure 15 on page 26 illustrates the different evacuation times by changing the number of passengers using an optimized arrangement of exits. Again, the more passen-

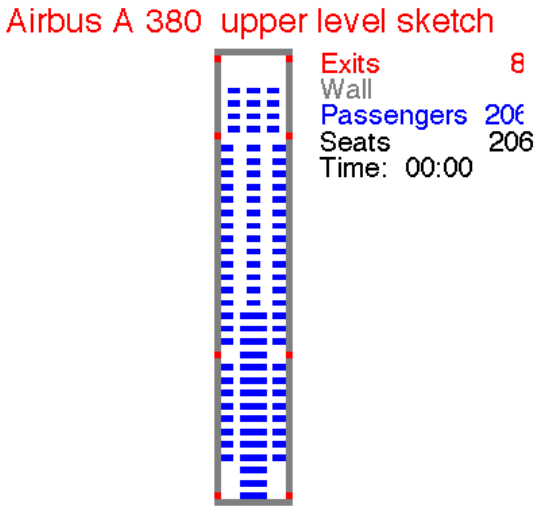

Figure 11: Cabin without obstacles under normal configuration

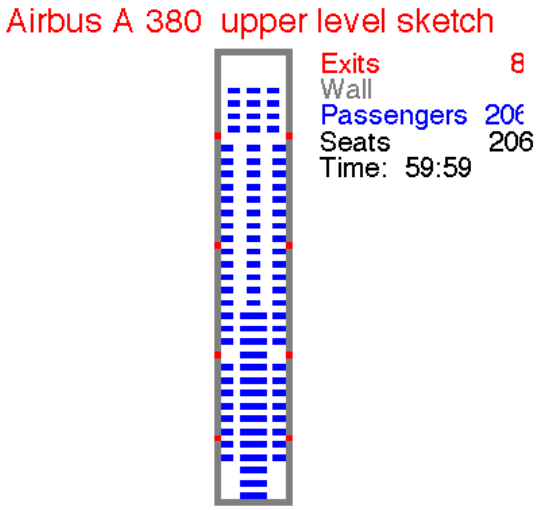

Figure 12: Cabin without obstacles and with optimized exit location

gers moving per time step, the lower the evacuation time. With high probability of choosing the best possible move, the standard deviation for each measurement decreases. With optimized exit locations, the evacuation times are reduced by approx-

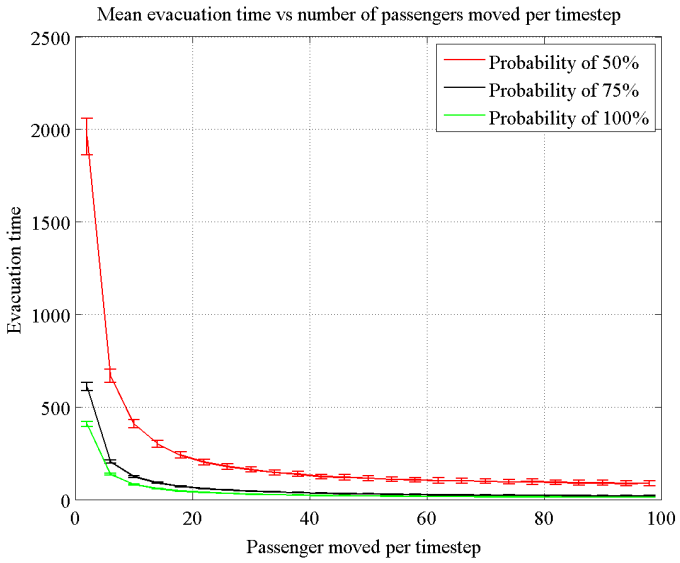

Figure 13: Mean evacuation time vs number of passengers moved per timestep

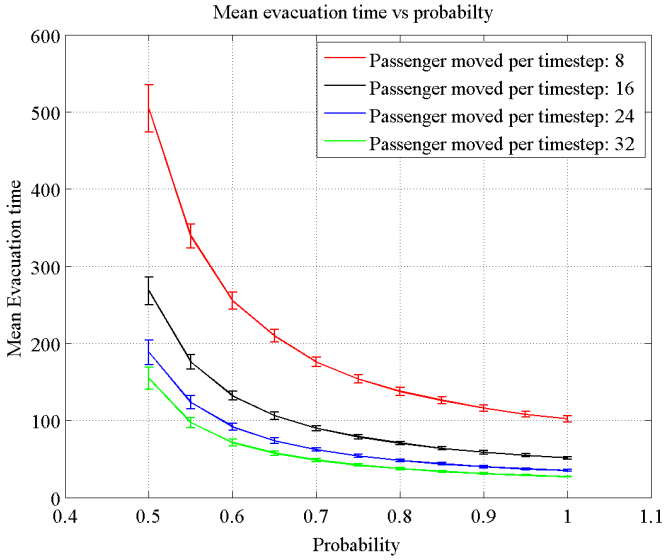

Figure 14: Mean evacuation time vs probability

imately 30 percent. In this case, the optimized exit locations are purely theoretical, because no stability aspects were taken into account.

Figure 16 on page 27 displays the mean evacuation time versus the probability of

each passenger selecting the best possible move. As expected, the mean evacuation time, as well as the corresponding standard deviation are decreasing with increasing probability. Here again the mean evacuation time was reduced by more than 30 percent.

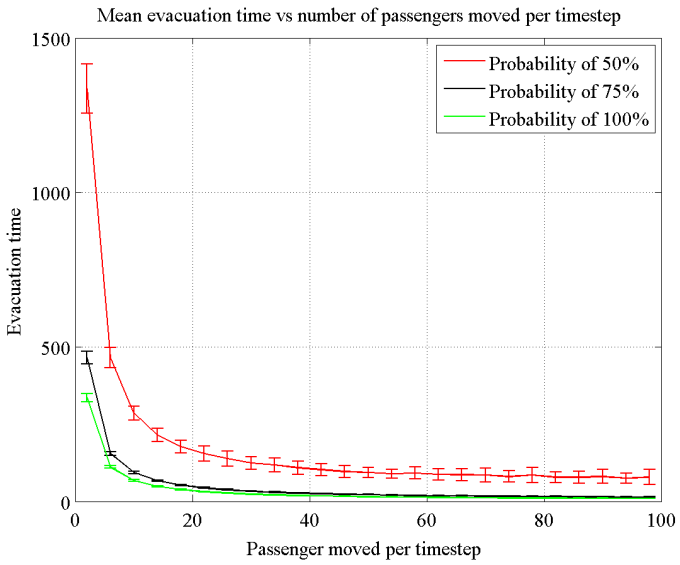

Figure 15: Mean evacuation time vs number of passengers moved per timestep (optimized)

5.4 Simulation with obstacles around doors

Figures 17 and 18 on page 27 indicate both configurations of the simulation domains with obstacles. The obstacles are placed around the four middle doors and are colored grey. The obstacles act like walls. The static field has not been changed. The obstacles were moved to the new locations of the corresponding doors.

5.4.1 Simulation with different parameters and obstacles

Figure 19 on page 28 shows the different evacuation times by altering the number of passengers under normal condition. The parameters are the number of passengers per time step moving and their probability to perform the best move. The more passengers moving per time step, the lower the evacuation time. With high probability of choosing the best possible move, the standard deviation for each measurement decreases.

Plot 20 on page 29 shows the mean evacuation time versus the probability of each passenger to do the best possible move. Yet again, the mean evacuation time, as

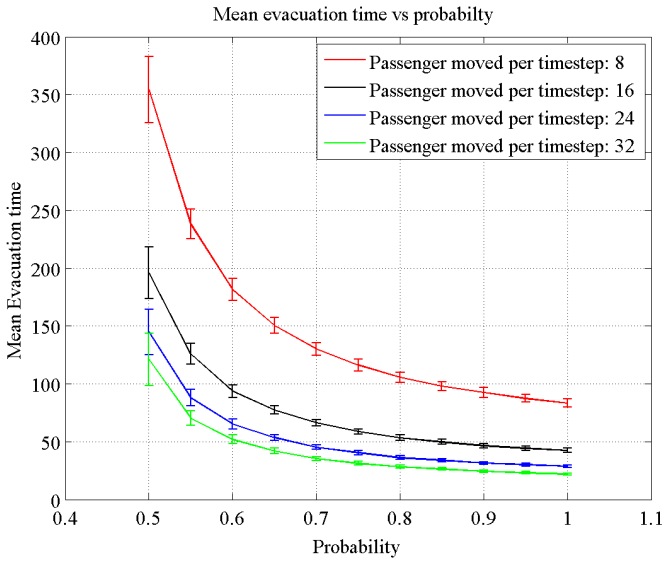

Figure 16: Mean evacuation time vs probability (optimized)

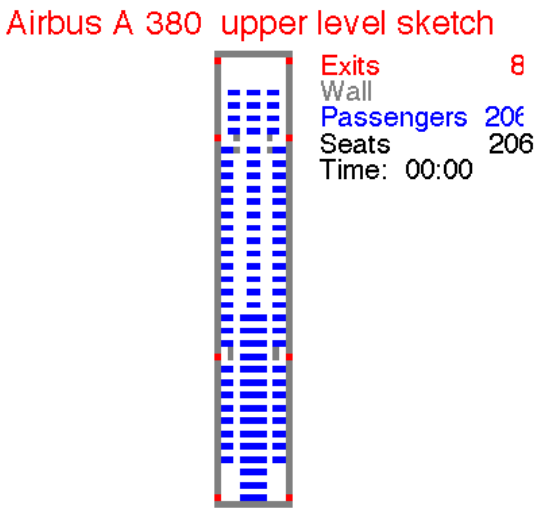

Figure 17: Cabin with obstacles under normal configuration

well as the corresponding standard deviation are constantly declining, the higher the probability is set.

Exits with obstacles result in a longer period of evacuation. The difference to

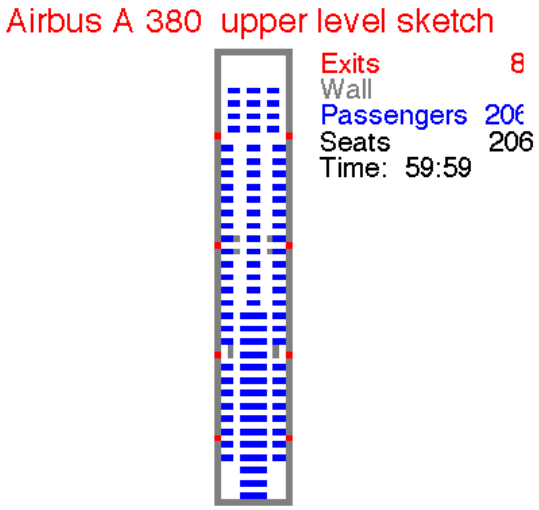

Figure 18: Cabin with obstacles and optimized exit location

normal configuration is more than 60 per cent.

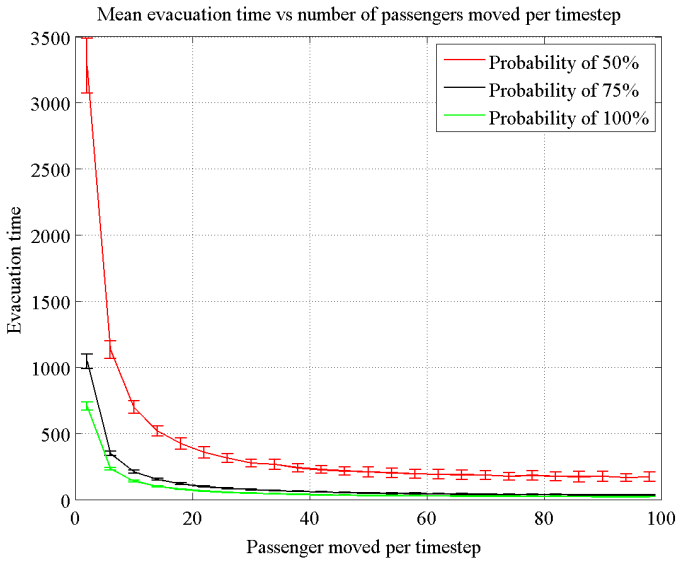

Figure 19: Mean evacuation time vs number of passengers moved per timestep (with obstacles)

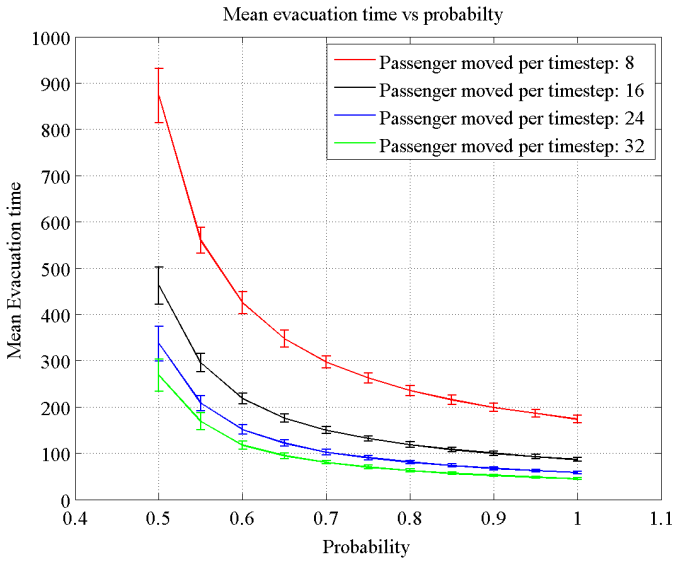

Figure 20: Mean evacuation time vs probability (with obstacles)

5.4.2 Simulation with different parameters, optimized location of emergency exits and obstacles

Figure 21 on page 30 shows the different evacuation times with a modified number of passengers with an optimized arrangement of exits. The exits are not free of obstacles according to figure 18 on page 28. Again the more passengers are moving per time step, the lower the evacuation time. With increased probability of choosing the best possible move the standard deviation for each measurement decreases. With optimized exit locations, the evacuation times dropped by approximately 30 percent with respect to the normal exit configuration with obstacles. The evacuation time lasts about 40 per cent longer than the one without obstacles. In real life, this difference could be deadly, which indicates the major importance of free exits.

Figure 16 on page 27 shows the mean evacuation time versus the probability of each passenger to choose the best possible step. As eagerly awaited, the mean evacuation time, as well as the corresponding standard deviation are again diametrically opposed to the probability. In that particular case, the mean evacuation time plummeted over 30 per cent too.

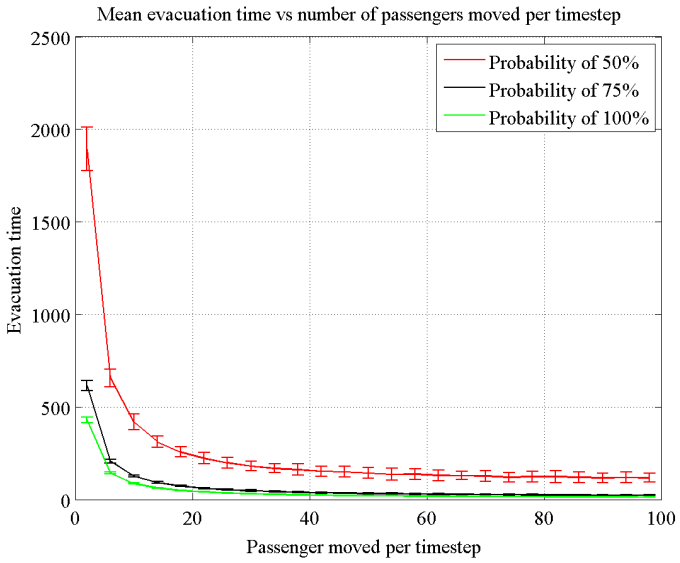

Figure 21: Mean evacuation time vs number of passengers moved per timestep (optimized and with obstacles)

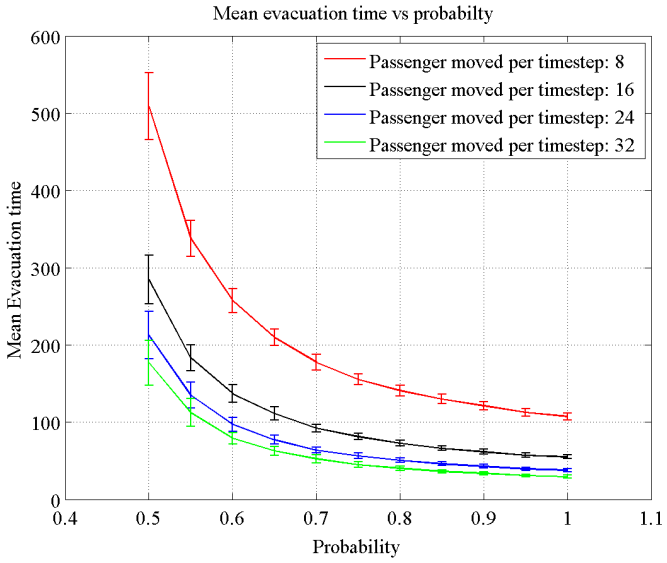

Figure 22: Mean evacuation time vs probability (optimized and with obstacles)

5.5 Discussion

The curves' level-off points in figure 23 and following, on pages 31 to 33, indicate a limit of passengers moving per time step. If more than 25 passengers move per time

step, the evacuation time remains reasonably steady. The reason of that behavior is due to reaching the maximum capability of the exits per time step. The same behavior is observed in the case with obstacles on figure 24 on page 32. The probability of choosing the correct move has a strong impact on the evacuation time. The higher the probability, the lower the evacuation time in both cases, with and without obstacles. The level-off trend in the high probability region exhibits the lower time rate. The results with obstacles look similar. Moreover the evacuation time increases with unexpected obstacles. "Unexpected" refers to the fact, that the static field has not been adapted. Agents follow their optimum path, until they see the unexpected obstacle. Then, they have to surround it. A considerably larger jamming trend can be observed around the exits concerned.

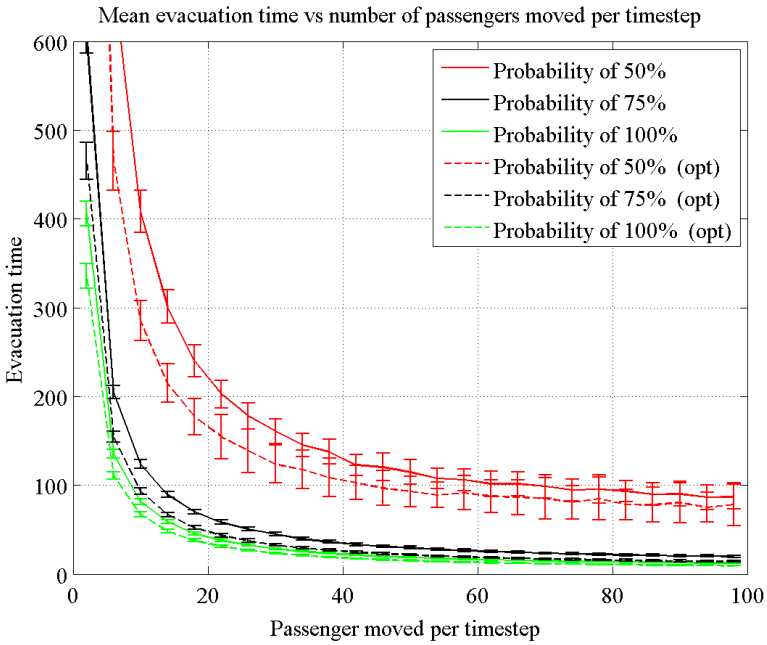

Figure 23: Mean evacuation time vs number of passengers moved per timestep

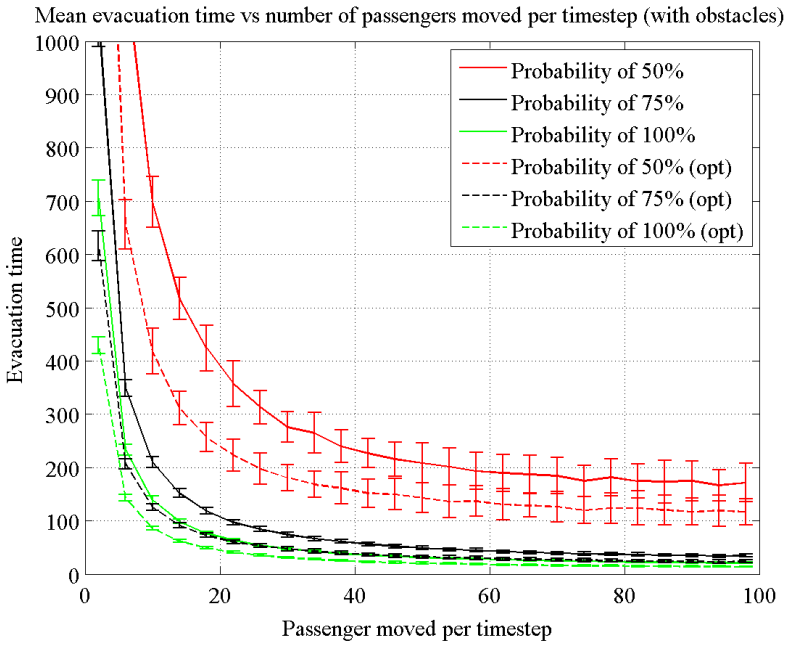

Figure 24: Mean evacuation time vs number of pax moved per timestep (with obstacles)

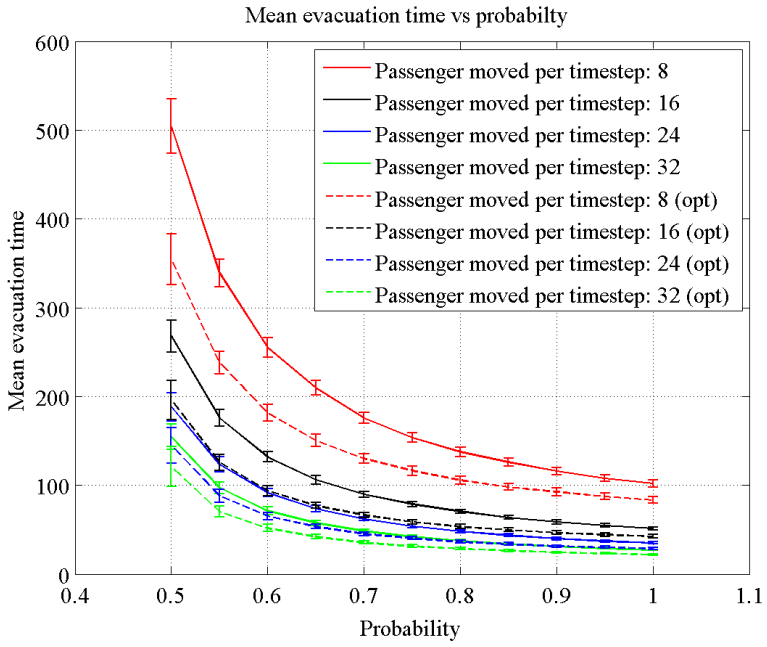

Figure 25: Mean evacuation time vs probability

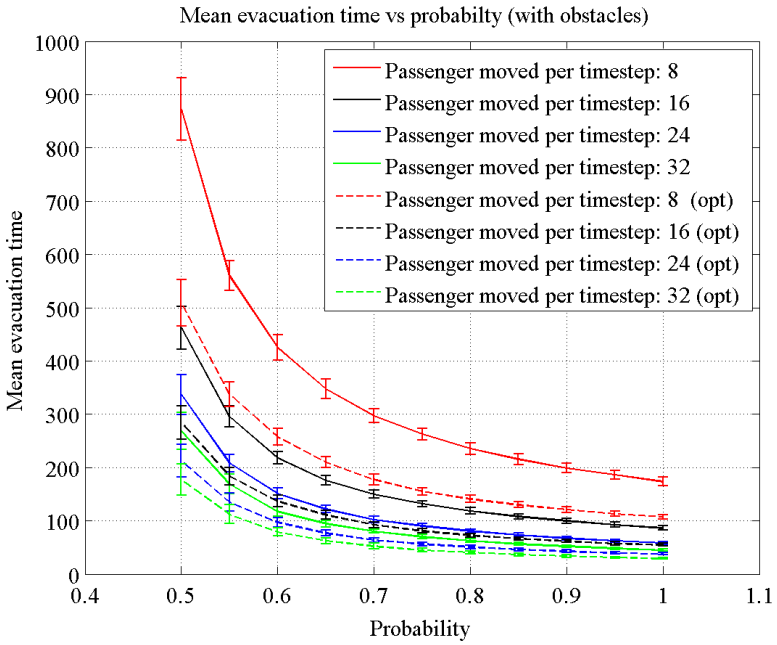

Figure 26: Mean evacuation time vs probability (with obstacles)

6 Summary and Outlook

In this chapter, we will summarize the usability of the two models as well as the simulation results. So as to enhance our models we will propose some improvement suggestions.

6.1 Comparison

Model 1 persuaded with its simplicity. For instance, its simulation time, with only one motion per iteration, is quite as fast as the simulation time of Model 2 using more of which. Conversely, there is no way Model 1 can compete with Model 2 in terms of adaptability. Whereas in Model 2 one easily can alter the probabilities, as well as the number of agents moving per time step, in Model 1 one has to change the probability parameters in the corresponding .m-file. Let alone the fact that in Model 1 only one agent can move at a time. Having said that, the required RAM storage, in addition to the computing speed, are considered more positive using Model 1.

Conclusively, we decided to compute our simulations with Model 2, so there are no necessary changes to be done in the .m-files.

6.2 Simulation results

All in all, we concluded, that an alternative choice of exit locations could, at least theoretically, reduce the evacuation time to a third of its original time. Conversely, the alternative we suggested would lead the passengers to the wings, which is not exactly to be recommended due to explosion considerations.

Moreover, obstacles in the exit areas can extend the evacuation time up to 40 per cent which is extremely poor.

In addition, the number of agents moving per iteration is restricted. Using common sense, it is not hard to recognize, that a moving crowd consisting of 200 people causes a much higher chaos and jamming factor as a group of 10 people. According to our results, the best rate of agents moving per time step is reached at a number of 25 agents.

Finally, in all cases, a high probability augments the evacuation efficiency.

6.3 Outlook

In order to collect appropriate statements about the models' prospective, we summed up the difficulties occurred in the past to conclude possible developments of our consisting model.

Difficulties within the first model:

- Many assumptions had to be made to enable the simulation
- Only one movement per iteration → slow simulation
- Complex description of the passenger's next step → not general valid
- Probability must be changed in the .m file
- Exit locations must be changed in the .m file

Difficulties within the second model:

- Many assumptions had to be made to enable the simulation
- Long calculation time → low numerical efficiency
- Many functions and files

Regarding the difficulties we experienced with our models, we can make following propositions to enhance them:

- Include a function to easily switch the exit locations
- Involve people with disabilities, children, and natural factors
- Enable interaction between the two decks
- Use calculated information for further analysis. For instance, if many people are in front of the nearest exit, choose the second nearest exit.
- Optimize visualization

There are several further proposals to upgrade our model. However, every enhancement is creating a more complex model and reduces its computational skills. All the assumptions we made (see page 8) could be eliminated or improved by other assumptions. Nevertheless, the model will get much more complicated and its efficiency questionable.

7 References

References

[1] www.en.wikipedia.org/wiki/main_page. internet.

[2] www.flightglobal.com/assets/getasset.aspx?itemid=19339. internet.

[3] www.informatik.hu-berlin.de/forschung/gebiete/algorithmenii/lehre/ss09/theo3/skript/thi3-bsp-fw1.pdf. internet.

[4] www.iti.fh-flensburg.de/lang/algorithmen/graph/warshall.htm. internet.

[5] www.singaporeair.com/saa/en_uk/images/company_info/eot/fleet_info/380/a380.jpg. internet.

[6] www.users.ices.utexas.edu/∼carsten/papers/bursteddeklauckschadschneideretal01.pdf. internet.

[7] Reine Byun and Anne Femmer. Modelling pedestrian dynamics using a descrete floor field and agend based simulation. Technical report, Modelling and Simulating Social Systems with MATLAB, 2008.

[8] A. Schadschneider C. Burstedde, K. Klauck and J.Zitzartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A*, pages 507–525, 2001.

8 Programs

8.1 Model 1

Listing 1: Main Program: evacuation.m

```
function [evacuation_time, iterations]=evacuation(level, visual, movie)
% level = [1,2] visual =[0,1] movie=[0,1] 0 = no 1 = yes
% This function shows an evacuation procedure of the Airbus A380 with
% passengers full of panic

% level 1 = upper level
% level 2 = lower level

if level>2
    error('Sorry, unfortunately the Airbus 380 consist only of two decks!');
end

if level<1
    error('Sorry, there is no such deck! choose between deck 1 and 2...');
end

simtime=200000;              % maximal simulation time
evacuation_time = simtime; % default time for break
frame = 1;                   % iteration variable for visual / movie
tic;                         % start counting time
earlier=now;                 % set time to measure
stopwatch=datestr(now-now,'MM:SS'); % set stopwatch to zero
%-----------------------------------UPPER LEVEL-----------------------------------%

if level==1
    % calculate the number of Passengers
    passenger_number = 200;
    % calculate length / width of level / number of exits
    length = 41;
    width  = 12;
    exits  =  8;
    % create the required level and calculate passenger and exit locations
    [upper_level]=make_upper_level(length,width);
    % find coordinates of any passenger
    [passenger_location]=ploc(length,width,passenger_number,upper_level);
    % find coordinates of any exit
    [exit_location]=eloc(length,width,exits,upper_level);
    % visualization of the level
    if visual==true
        upper_level_sketch(upper_level,passenger_number,stopwatch);
        M(frame) = getframe;              % set first visual / movie frame
        frame = frame + 1;                % get ready for next frame
    end

    % Running the simulation

for t=1:simtime

% select a ramdom Passenger in the required deck
randpass = round(rand*(passenger_number-1))+1;
% calculating the nearest exit location of the random Passenger
```

```
[nearest_exit_location]=...
min_distance(passenger_location(randpass,:),exit_location);

% calculating the next location of the Passenger
new_location = passenger_move(passenger_location(randpass,:),...
nearest_exit_location,upper_level);

%------PASSENGER DYNAMICS: all possibilities of altering position --------%

% simulation of a passenger that has reached the exit
if upper_level(new_location(1,1), new_location(1,2)) == 4
   upper_level(passenger_location(randpass,1), ...
   passenger_location(randpass,2)) = 1;       % give free space

 if visual == true
 upper_level(new_location(1,1), new_location(1,2)) = 6; % the point of exit
 upper_level_sketch(upper_level,passenger_number,stopwatch); % update state
 M(frame) = getframe;                                % set first movie frame
 frame = frame + 1;                                  % get ready for next frame
 pause(0.1);                                         % slow simulation for exiting
 upper_level(new_location(1,1), new_location(1,2)) = 4; % give exit free
 end

passenger_location(randpass,:) = [];  % remove the passenger from the deck
passenger_number= passenger_number-1; % adjust passenger number
end

% if passenger was on a seat and went to free space
if (upper_level(new_location(1,1), new_location(1,2)) == 1 && ...
     upper_level(passenger_location(randpass,1),...
     passenger_location(randpass,2))==3)

upper_level(passenger_location(randpass,1),...
passenger_location(randpass,2))=5;        % old position is now a free seat

 if visual == true
 % new location ( former free space) is occupied with passenger
 upper_level(new_location(1,1),new_location(1,2))=6;
 upper_level_sketch(upper_level,passenger_number,stopwatch); % update state
 M(frame) = getframe;                                  % set first movie frame
 frame = frame + 1;                                    % get ready for next frame
 end

% move passenger to next location (length)
passenger_location(randpass,1)= new_location(1,1);
% move passenger to next location (width)
passenger_location(randpass,2)= new_location(1,2);

% if passenger was on a seat and went to a free seat
elseif (upper_level(new_location(1,1), new_location(1,2)) == 5 && ...
         upper_level(passenger_location(randpass,1),...
         passenger_location(randpass,2))==3)

upper_level(passenger_location(randpass,1),...
passenger_location(randpass,2))=5;        % old position is now a free seat

if visual == true
% new location ( former free seat) is occupied with passenger
```

```
upper_level(new_location(1,1),new_location(1,2))=3;
upper_level_sketch(upper_level,passenger_number,stopwatch);  % update state
M(frame) = getframe;                                % set first movie frame
frame = frame + 1;                                  % get ready for next frame
end

% move passenger to next location (length)
passenger_location(randpass,1)= new_location(1,1);
% move passenger to next location (width)
passenger_location(randpass,2)= new_location(1,2);

% if passenger was on free space and went to a free space
elseif (upper_level(new_location(1,1), new_location(1,2)) == 1 && ...
        upper_level(passenger_location(randpass,1),...
        passenger_location(randpass,2))==6)

upper_level(passenger_location(randpass,1),...
passenger_location(randpass,2))=1;        % old position in now free space

if visual == true
% new location ( former free space) is occupied with passenger
upper_level(new_location(1,1),new_location(1,2))=6;
upper_level_sketch(upper_level,passenger_number,stopwatch);  % update state
M(frame) = getframe;                                 % set first movie frame
frame = frame + 1;                                  % get ready for next frame
end

% move passenger to next location (length)
passenger_location(randpass,1)= new_location(1,1);
% move passenger to next location (width)
passenger_location(randpass,2)= new_location(1,2);

% if passenger was on free space and went to a free seat
elseif (upper_level(new_location(1,1), new_location(1,2)) == 5 && ...
        upper_level(passenger_location(randpass,1),...
        passenger_location(randpass,2))==6)

upper_level(passenger_location(randpass,1),...
passenger_location(randpass,2))=1;        % old position = free space

if visual == true
% new location ( former free seat) is occupied with passenger
upper_level(new_location(1,1),new_location(1,2))=3;
upper_level_sketch(upper_level,passenger_number);             % update state
M(frame) = getframe;                                % set first movie frame
frame = frame + 1;                                  % get ready for next frame
end

% move passenger to next location (length)
passenger_location(randpass,1)= new_location(1,1);
% move passenger to next location (width)
passenger_location(randpass,2)= new_location(1,2);
end

   % End of the simulation
     if passenger_number == 0
          evacuation_time = toc;                    % stop time
          iterations = t;                           % stop interations counter
```

```
168             break;
169         end
170             stopwatch=datestr(now-earlier,'MM:SS'); % update stopwatch
171 end    %  end of the (for t = 1:simtime) loop
172
173         if movie ==true
174             movie2avi(M,'evacuation_A380_upper_level.avi'); % save the movie
175         end
176 end    %  end of the (if level ==1) statements
177
178 %-----------------------------------LOWER LEVEL-----------------------------------%
179
180 if level==2
181         % calculate the number of Passengers
182         passenger_number = 358;
183         % calculate length / width of level
184         length = 55;
185         width  = 14;
186         exits  = 10;
187         % create the required level and calculate passenger and exit locations
188         [lower_level]=make_lower_level(length,width);
189         % find coordinates of any passenger
190         [passenger_location]=ploc(length,width,passenger_number,lower_level);
191         % find coordinates of any exit
192         [exit_location]=eloc(length,width,exits,lower_level);
193         % visualization of the level
194         if visual==true
195               lower_level_sketch(lower_level,passenger_number,stopwatch);
196               M(frame) = getframe;                % set first visual / movie frame
197               frame = frame + 1;                  % get ready for next frame
198         end
199
200         % Running the simulation
201
202 for t=1:simtime
203 % select a ramdom Passenger in the required deck
204 randpass = round(rand*(passenger_number-1))+1;
205 % calculating the nearest exit location of the random Passenger
206 [nearest_exit_location]=...
207 min_distance(passenger_location(randpass,:),exit_location);
208 % calculating the next location of the Passenger
209 new_location = passenger_move(passenger_location(randpass,:),...
210 nearest_exit_location,lower_level);
211
212 %------ PASSENGER DYNAMICS: all possibilities of altering position -------%
213
214 % simulation of a passenger that has reached the exit
215 if lower_level(new_location(1,1), new_location(1,2)) == 4
216    lower_level(passenger_location(randpass,1), ...
217    passenger_location(randpass,2)) = 1;     % give free space
218
219 if visual == true
220 % Passenger at the point of exit
221 lower_level(new_location(1,1), new_location(1,2)) = 6;
222 lower_level_sketch(lower_level,passenger_number,stopwatch); % update state
223 M(frame) = getframe;                                % set first movie frame
224 frame = frame + 1;                                  % get ready for next frame
225 pause(0.1);                                         % slow simulation for exiting
```

```
226 lower_level(new_location(1,1), new_location(1,2)) = 4;      % give exit free
227 end
228
229
230 passenger_location(randpass,:) = [];  % remove the passenger from the deck
231 passenger_number= passenger_number-1; % adjust passenger number
232 end
233
234 % if passenger was on a seat and went to free space
235 if (lower_level(new_location(1,1), new_location(1,2)) == 1 && ...
236         lower_level(passenger_location(randpass,1),...
237         passenger_location(randpass,2))==3)
238
239 lower_level(passenger_location(randpass,1),...
240 passenger_location(randpass,2))=5;        % old position is now a free seat
241
242 if visual == true
243 % new location ( former free space) is occupied with passenger
244 lower_level(new_location(1,1),new_location(1,2))=6;
245 lower_level_sketch(lower_level,passenger_number,stopwatch); % update state
246 M(frame) = getframe;                                % set first movie frame
247 frame = frame + 1;                                  % get ready for next frame
248 end
249
250 % move passenger to next location (length)
251 passenger_location(randpass,1)= new_location(1,1);
252 % move passenger to next location (width)
253 passenger_location(randpass,2)= new_location(1,2);
254
255 % if passenger was on a seat and went to a free seat
256 elseif (lower_level(new_location(1,1), new_location(1,2)) == 5 &&...
257         lower_level(passenger_location(randpass,1),...
258         passenger_location(randpass,2))==3)
259
260 lower_level(passenger_location(randpass,1),...
261 passenger_location(randpass,2))=5;     % old position is now a free seat
262
263 if visual == true
264 % new location ( former free seat) is occupied with passenger
265 lower_level(new_location(1,1),new_location(1,2))=3;
266 lower_level_sketch(lower_level,passenger_number,stopwatch); % update state
267 M(frame) = getframe;                                % set first movie frame
268 frame = frame + 1;                                  % get ready for next frame
269 end
270
271 % move passenger to next location (length)
272 passenger_location(randpass,1)= new_location(1,1);
273 % move passenger to next location (width)
274 passenger_location(randpass,2)= new_location(1,2);
275
276 % if passenger was on free space and went to a free space
277 elseif (lower_level(new_location(1,1), new_location(1,2)) == 1 &&...
278         lower_level(passenger_location(randpass,1),...
279         passenger_location(randpass,2))==6)
280
281 lower_level(passenger_location(randpass,1),...
282 passenger_location(randpass,2))=1;        % old position in now free space
283
```

```
284 if visual == true
285 % new location ( former free space) is occupied with passenger
286 lower_level(new_location(1,1),new_location(1,2))=6;
287 lower_level_sketch(lower_level,passenger_number,stopwatch); % update state
288 M(frame) = getframe;                                  % set first movie frame
289 frame = frame + 1;                                    % get ready for next frame
290 end
291
292 % move passenger to next location (length)
293 passenger_location(randpass,1)= new_location(1,1);
294 % move passenger to next location (width)
295 passenger_location(randpass,2)= new_location(1,2);
296
297 % if passenger was on free space and went to a free seat
298 elseif (lower_level(new_location(1,1), new_location(1,2)) == 5 &&...
299         lower_level(passenger_location(randpass,1),...
300         passenger_location(randpass,2))==6)
301
302 lower_level(passenger_location(randpass,1),...
303 passenger_location(randpass,2))=1;        % old position = free space
304
305 if visual == true
306 % new location ( former free seat) is occupied with passenger
307 lower_level(new_location(1,1),new_location(1,2))=3;
308 lower_level_sketch(lower_level,passenger_number,stopwatch); % update state
309 M(frame) = getframe;                                  % set first movie frame
310 frame = frame + 1;                                    % get ready for next frame
311 end
312
313 % move passenger to next location (length)
314 passenger_location(randpass,1)= new_location(1,1);
315 % move passenger to next location (width)
316 passenger_location(randpass,2)= new_location(1,2);
317 end
318
319     % End of the simulation
320       if passenger_number == 0
321            evacuation_time = toc;              % stop time
322            iterations = t;                     % stop iterations counter
323            break;
324       end
325 stopwatch=datestr(now-earlier,'MM:SS');         % update stopwatch
326 end   %  end of the (for t = 1:simtime) loop
327    if movie ==true
328       movie2avi(M,'evacuation_A380_lower_level.avi'); % save movie
329    end
330
331 end   %  end of the (if level ==2) statements
```

Listing 2: Function: supper_level_sketch.m

```
1 function[]=upper_level_sketch(upper_level,passenger_number,stopwatch)
2 % sketch Matrix with different colors for simulation
3 % define different colors for all states:
4
5 % white for free space
6 sketch_colors(1,:) = [1 1 1];
7 % grey for wall
```

```
sketch_colors(2,:) = [.5 .5 .5];
% blue for passengers on seats
sketch_colors(3,:) = [0 0 1];
% red for exits
sketch_colors(4,:) = [1 0 0];
% black for empty seats
sketch_colors(5,:) = [0 0 0];
% blue for passengers on free space
sketch_colors(6,:) = [0 0 1];
clf % clear current figure

x = passenger_number;
time=stopwatch;
% drawing the upper_level_sketch
image(upper_level);
title('Airbus A 380  upper level sketch','color','r','fontsize',20);
colormap(sketch_colors);
text(17,3,'Exits                8','color','r','fontsize',16);
text(17,7,'Wall ','color',[.5 .5 .5],'fontsize',16);
text(17,11,['Passengers  ',int2str(x)],'color',[0 0 1],'fontsize',16);
text(17,15,'Seats          200 ','color',[0 0 0],'fontsize',16);
text(17,19,['Time:  ',time],'color',[0 0 0],'fontsize',16);
axis off;
axis image;
```

Listing 3: Function: make_upper_level.m

```
function[upper_level]= make_upper_level(length,width)

% creating the upper-level and placing the passengers on their seats
% meaning of the numbers:
% 1 = free space
% 2 = wall
% 3 = passenger
% 4 = exit

% create matrix for upper-level
upper_level = ones(length,width);

% create walls
for i = 1:length
    upper_level(i,1) = 2;
    upper_level(i,width) = 2;
end
for j = 1:width
    upper_level(1,j) = 2;
    upper_level(length,j) = 2;
end
% create the 8 exits
upper_level(2,1) = 4;
upper_level(2,width) = 4;
upper_level(11,1) = 4;
upper_level(11,width) = 4;
upper_level(30,1) = 4;
upper_level(30,width) = 4;
upper_level(40,1) = 4;
upper_level(40,width) = 4;
% set the passengers on their seats
```

```
upper_level(7:10,3:4)   = 3;   % business class
upper_level(7:10,6:7)   = 3;
upper_level(7:10,9:10)  = 3;
upper_level(13:24,2:3)  = 3;
upper_level(13:24,6:7)  = 3;
upper_level(13:24,10:11) = 3;

upper_level(26:28,2:3)  = 3;   % economy
upper_level(26:28,10:11)= 3;
upper_level(32:39,2:3)  = 3;
upper_level(32:39,10:11)= 3;
upper_level(26:40,5:8)  = 3;
```

Listing 4: Functioin: eloc.m

```
function[exit_location] = eloc(length, width,exit_number,level)

% this function calculates the exit coordinates

exit_location = ones(exit_number,2);  % default
k=1;l=1;
for i = 1:length
   for j = 1:width

        if  level(i,j)==4
            exit_location(k,1) = i;
            exit_location(l,2) = j;
            k = k+1;
            l=l+1;
        end
      j=j+1;
   end
   i = i+1;
end
```

Listing 5: Function: min_distance.m

```
function[nearest_exit_location]=...
min_distance(passenger_location,exit_location)

% this function measures the distance to the nearest exit of a random
% passenger, as well as the location of his nearest exit

delta=1000*ones(1,2);
k=1;

    for j = 1:length(exit_location)
        delta(k,1) = abs(passenger_location(1,1)-exit_location(j,1));
        delta(k,2) = abs(passenger_location(1,2)-exit_location(j,2));
        deltanorm(k) = sqrt(delta(k,1)^2+delta(k,2)^2);
        j=j+1;
        k=k+1;
    end

    [min_dist,index]= min(deltanorm);
    % calculate nearest exit coordinates
    nearest_exit_location(1,1) = exit_location(index,1);
```

```
21      nearest_exit_location(1,2) = exit_location(index,2);
```

Listing 6: Function: ploc.m

```
1  function [passenger_location] = ploc(length,width,passenger_number,level)
2
3  % this function calculates the passenger's coordinates
4
5  passenger_location = ones(passenger_number,2); % default
6  k=1;l=1;
7  for i = 1:length
8      for j = 1:width
9
10         if  level(i,j)==3
11             passenger_location(k,1) = i;
12             passenger_location(l,2) = j;
13             k = k+1;
14             l=l+1;
15         end
16        j=j+1;
17     end
18     i = i+1;
19 end
```

Listing 7: Function: passenger_move.m

```
1  function[new_location]= ...
2  passenger_move(passenger_location,nearest_exit_location,level)
3
4  % This function calculates the next step for a passenger who wants to reach
5  % the nearest exit at all costs!
6
7  % all possible neighbour cells a passenger can move to in one step
8  % meaning: [up;right;down;left;stay]
9  neighbour=[-1 0; 0 1; 1 0; 0 -1; 0 0];
10
11 for i = 1:length(neighbour)
12  % safe all possible new_locations of the passenger
13  new_location(i,:) = passenger_location+neighbour(i,:);
14 end
15
16 prob = rand; % variable to add probabilities
17
18 % if Passenger is stuck left
19
20     if (level(passenger_location(1,1),passenger_location(1,2))==6 &&...
21         level(new_location(4,1),new_location(4,2))==2 &&...
22         level(new_location(1,1),new_location(1,2))==5 &&...
23         level(new_location(3,1),new_location(3,2))==5 &&...
24         level(passenger_location(1,1),passenger_location(1,2)+2)==1)
25
26     new_location = passenger_location(1,:)+[0,2];  % go twice right
27
28 % if Passenger is stuck right
29
30     elseif (level(passenger_location(1,1),passenger_location(1,2))==6 &&...
31             level(new_location(2,1),new_location(2,2))==2 &&...
```

```
32              level(new_location(1,1),new_location(1,2))==5 &&...
33              level(new_location(3,1),new_location(3,2))==5 &&...
34              level(passenger_location(1,1),passenger_location(1,2)-2)==1)
35
36      new_location = passenger_location(1,:)-[0,2];  % go twice left
37
38  % if Passenger is on free space
39
40  % if the exit is lower and passenger on free space
41      elseif (level(passenger_location(1,1),passenger_location(1,2))==6 &&...
42              (nearest_exit_location(1,1)>passenger_location(1,1) ))
43          if (level(new_location(3,1),new_location(3,2))==1) % if down  free
44                  new_location = new_location(3,:);           % move down
45          % if down is occupied but left is free
46          elseif (level(new_location(4,1),new_location(4,2))==1 && prob>0.5)
47                  new_location = new_location(4,:);           % move left
48          % if left is also occupied and right is free
49          elseif (level(new_location(2,1),new_location(2,2))==1 && prob<0.5)
50                  new_location = new_location(2,:);           % move right
51          else    new_location = new_location(5,:);           % stay
52
53          end
54  % if the exit is higher and passenger on free space
55      elseif (level(passenger_location(1,1),passenger_location(1,2))==6 &&...
56              (nearest_exit_location(1,1)<passenger_location(1,1) ))
57          if (level(new_location(1,1),new_location(1,2))==1) % if up is free
58                  new_location = new_location(1,:);           % move up
59          % if up is occupied but left is free
60          elseif (level(new_location(4,1),new_location(4,2))==1)
61                  new_location = new_location(4,:);           % move left
62          % if left is also occupied and right is free
63          elseif (level(new_location(2,1),new_location(2,2))==1)
64                  new_location = new_location(2,:);           % move right
65          else    new_location = new_location(5,:);           % stay
66
67          end
68  % if Passenger is on a seat
69
70      % if the passenger is on the front row of a seat
71      elseif (level(passenger_location(1,1),passenger_location(1,2))==3 &&...
72              level(new_location(1,1),new_location(1,2))==1)
73           new_location = new_location(1,:);                  % move up
74      % if the passenger is on a seat and the exit is on his right side
75      elseif (level(passenger_location(1,1),passenger_location(1,2))==3 &&...
76              nearest_exit_location(1,2)>passenger_location(1,2))
77          % if next right is free
78          if (level(new_location(2,1),new_location(2,2))==5| ...
79             (level(new_location(2,1),new_location(2,2)))==1 && prob >0.6)
80
81              new_location = new_location(2,:);  % 40% chance to turn right
82          % if next left is free
83          elseif (level(new_location(4,1),new_location(4,2))==5| ...
84                 (level(new_location(4,1),new_location(4,2)))==1 && prob<0.4)
85
86              new_location = new_location(4,:);  % 40% chance to turn left
87          else
88              new_location = new_location(5,:);  % 20% chance to stay
89          end
```

```
90      % if the passenger is on a seat and the exit is on his left side
91      elseif (level(passenger_location(1,1),passenger_location(1,2))==3 &&...
92              nearest_exit_location(1,2)<passenger_location(1,2))
93          % if next left is free
94          if (level(new_location(4,1),new_location(4,2))==5 |...
95             (level(new_location(4,1),new_location(4,2)))==1 && prob >0.6)
96
97              new_location = new_location(4,:);  % 40% chance to turn left
98          % if next right is free
99          elseif (level(new_location(2,1),new_location(2,2))==5 |...
100                (level(new_location(2,1),new_location(2,2)))==1 && prob<0.4)
101
102            new_location = new_location(2,:);  % 40% chance to turn right
103         else
104             new_location = new_location(5,:);  % 20% chance to stay
105         end
106
107     % if Passenger is on the same height as an exit
108
109     % if exit left same hight
110     elseif (nearest_exit_location(1,1)== passenger_location(1,1)&&...
111             nearest_exit_location(1,2)<passenger_location(1,2))
112           new_location = new_location(4,:);     % go left
113
114     % if exit right same hight
115     elseif (nearest_exit_location(1,1)== passenger_location(1,1)&&...
116             nearest_exit_location(1,2)>passenger_location(1,2))
117           new_location = new_location(2,:);     % go right
118   end
```

8.2 Model 2

Listing 8: Main Program: evac_V5.m

```
1  %Evacuation of A380 with a static flow field
2
3  function [countTime]=evac_V5(visual,movie)
4
5
6  %This function requires the static flow field and the full air plane, with
7  %passengers seated on their seats. The imported data are the level to
8  %simulate and the visualisation parameters.
9
10
11 %-------------------------------PRE PROCESSING-------------------------------%
12 load upperdeck-mit-gang-min;
13 static = Min;
14
15
16
17 %Time measurement (chrono)
18 maxSimT      = 100000;                  % maximum simulation time
19 evacTime     = maxSimT;                 % default time for break
20 frame        = 1;                       % iteration variable for visual / movie
21 tic;                                    % start counting time
```

```
startT       = now;                          % set time to measure
stopwatch    = datestr(now-now,'MM:SS');     % set stopwatch to zero
dt           = frame/200;                    % timestep for simulation
paxMovePerTs= 32 ;                           %Numer of pax movements per time step
level = 1;

%-----------------------------------SIMULATION-----------------------------------%

if level==1

    % Counts passengers, exits and calulates width an leng of plane and
    % creates the upper level
    [pax,exits,leng,width,upper_level]=initialize(static);

    % find coordinates of any passenger
    [paxLocation,exitLocation]=paxLoc(leng,width,pax,upper_level);


    % visualization of the level
    if visual==true
        field_upper_level_sketch(upper_level,pax,stopwatch);
        M(frame) = getframe;                 % set first visual / movie frame
        frame = frame + 1;
        pause(1);% get ready for next frame
    end



    % all Pax getting up
    [upper_level,count]=paxGetUp(leng,width,upper_level);
    [paxLocation,exitLocation]=paxLoc(leng,width,pax,upper_level);

    if visual==true
        field_upper_level_sketch(upper_level,pax,stopwatch);
        M(frame) = getframe;                 % set first visual / movie frame
        frame = frame + 1;
        pause(0.1);% get ready for next frame
    end



    % Running the simulation
    for t=1:maxSimT

        %Check length of paxLocation
        sizePaxLocation=size(paxLocation);
        sizePaxLocation=sizePaxLocation(1,1);

        %Check if enough pax on board to move
        if sizePaxLocation < paxMovePerTs
            paxMovePerTs = sizePaxLocation;
        end %if

        for p=1:paxMovePerTs %Numer of pax movements per time step

            x=paxLocation(p,2);
            y=paxLocation(p,1);
            upper_level = paxMove(x,y,upper_level,static);
```

```
80
81          end %for
82
83
84          %Count the passengers and check their new locations
85          [pax]=paxCount(upper_level);
86          [paxLocation,exitLocation]=paxLoc(leng,width,pax,upper_level);
87
88          %Actualize the chrono
89          stopwatch=datestr(now-startT,'MM:SS');
90
91          if visual==true
92                field_upper_level_sketch(upper_level,pax,stopwatch);
93                M(frame) = getframe;                % set first visual / movie frame
94                frame = frame + 1;
95                pause(0.075);% get ready for next frame
96          end %if
97
98
99
100
101
102       if pax <= 0
103            break;
104       end %if
105
106
107
108       end %for t
109
110  %-------------------------------------POST PROCESSING-------------------------------------%
111      if movie == true
112          movie2avi(M,'evacuation_A380_upper_level_070.avi','COMPRESSION','None');
     % save the visualization as movie
113      end
114
115
116
117
118
119
120
121 end %if
```

Listing 9: Function: paxMove.m

```
1  %function to move passenger (x und y sind Koordinaten des Punktes)
2
3  function [level] = paxMove(x,y,level,static)
4
5  %1st Order von Neumann Neighbourhood
6  neigh=[1 0;0 -1;-1 0;0 1];
7
8  %Shuffle of neighbours matrix
9  sizeNeigh=size(neigh);
10 sizeNeigh=sizeNeigh(1,1);
11
12 randNo=randperm(sizeNeigh);
```

```
for k=1:sizeNeigh
    neigh_rand(randNo(k),:)=neigh(k,:);
end %for
neigh = neigh_rand;



tmp=[-100 -100 -100 -100];
quit=0;




%Iteration over neighboures
for k=1:4
    x2 = x + neigh(k,1);
    y2 = y + neigh(k,2);

    %Check what cell is next and remember staic value
    if level(y2,x2) == 1
        tmp(k) = static(y2,x2);
    elseif level(y2,x2) == 4
        level(y,x) = 1;
        quit=1;
    end %if


end %for k


%Move or stay
prob=0.75; %probabiltiy to make correct move
if min(abs(tmp)) ~= 100 && quit~=1
    if rand(1,1) >= prob
        [tmp I] = max(tmp); %I is index of maximal value
        x2 = x + neigh(I,1);
        y2 = y + neigh(I,2);
        level(y2,x2)=3;
        level(y,x)=1;
    else
        [tmp I] = min(abs(tmp)); %I is index of maximal value
        x2 = x + neigh(I,1);
        y2 = y + neigh(I,2);
        level(y2,x2)=3;
        level(y,x)=1;
    end %if
end %if
```

Listing 10: Function: initialize.m

```
%Function to count passengers and exits

function [countP,countE,l,w,A]=initialize(static)


[l,w] = size(static);

A = field_make_upper_level(l,w);
```

```
9
10 countP=0;
11 countE=0;
12 for k = 1:l
13     for m = 1:w
14
15         if A(k,m) == 3
16             countP=countP+1;
17         elseif A(k,m) == 4
18             countE=countE+1;
19         end %if
20
21
22     end %for m
23 end %for k
```

Listing 11: Function: paxLoc.m

```
1  function [paxLoc_rand,exLoc] = paxLoc(length,width,paxNo,level)
2
3  % this function calculates the passenger's coordinates
4
5  paxLoc = ones(paxNo,2);
6  paxLoc_rand=ones(paxNo,2);
7  exLoc = ones(paxNo,2);
8
9  k=1;l=1; %Pax
10 o=1;p=1; %Exits
11
12
13 for i = 1:length
14    for j = 1:width
15        if  level(i,j)==3
16            paxLoc(k,1) = i;
17            paxLoc(l,2) = j;
18            k = k+1;
19            l = l+1;
20        elseif level(i,j)==4
21            exLoc(o,1) = i;
22            exLoc(p,2) = j;
23            o = o+1;
24            p = p+1;
25        end % if
26    end %for j
27 end %for i
28
29 %Shuffle of pax location matrix
30 sizePaxLoc=size(paxLoc);
31 sizePaxLoc=sizePaxLoc(1,1);
32
33 randNo=randperm(sizePaxLoc);
34
35 for k=1:sizePaxLoc
36     paxLoc_rand(randNo(k),:)=paxLoc(k,:);
37 end %for
```

Listing 12: Function: field_upper_level_sketch.m

```
function[]=field_upper_level_sketch(upper_level,passenger_number,stopwatch)
% sketch Matrix with different colors for simulation
% define different colors for all states:

% white for free space
sketch_colors(1,:) = [1 1 1];
% grey for wall
sketch_colors(2,:) = [.5 .5 .5];
% blue for passengers on seats
sketch_colors(3,:) = [0 0 1];
% red for exits
sketch_colors(4,:) = [1 0 0];
% black for empty seats
sketch_colors(5,:) = [0 0 0];
% blue for passengers on free space
sketch_colors(6,:) = [0 0 1];
clf % clear current figure

x = passenger_number;
time=stopwatch;
% drawing the upper_level_sketch
image(upper_level);
title('Airbus A 380  upper level sketch','color','r','fontsize',20);
colormap(sketch_colors);
text(17,3,'Exits                8','color','r','fontsize',16);
text(17,7,'Wall ','color',[.5 .5 .5],'fontsize',16);
text(17,11,['Passengers  ',int2str(x)],'color',[0 0 1],'fontsize',16);
text(17,15,'Seats           206 ','color',[0 0 0],'fontsize',16);
text(17,19,['Time:  ',time],'color',[0 0 0],'fontsize',16);
axis off;
axis image;
```

Listing 13: Function: paxGetUp.m

```
%function to move passenger

function [A,count] = paxGetUp(length,width,level)

A=level;
count=0;
for k = 1:length
    for m = 1:width
        if A(k,m) == 3
            A(k,m) = 5;
            A(k-1,m) = 3;
            count = count + 1;
        end %if
    end %for m
end %for k
```

Listing 14: Function: paxCount.m

```
%Function to count passengers and exits

function [countP]=paxCount(level)
```

```
4
5
6 [l,w] = size(level);
7
8 A = level;
9
10
11 countP=0;
12
13 for k = 1:l
14     for m = 1:w
15
16         if A(k,m) == 3
17             countP=countP+1;
18         end %if
19
20
21     end %for m
22 end %for k
```

Listing 15: Function: field_make_upper_level.m

```
1 function[upper_level]= field_make_upper_level(length,width)
2
3 % creating the upper-level and placing the passengers on their seats
4 % meaning of the numbers:
5 % 1 = free space
6 % 2 = wall
7 % 3 = passenger
8 % 4 = exit
9 % 5 = seat
10
11 % create matrix for upper-level
12 upper_level = ones(length,width);
13
14 % create walls
15 for i = 1:length
16     upper_level(i,1) = 2;
17     upper_level(i,width) = 2;
18 end
19 for j = 1:width
20     upper_level(1,j) = 2;
21     upper_level(length,j) = 2;
22 end
23 % create the 8 exits
24 upper_level(14,1) = 4;
25 upper_level(14,width) = 4;
26 upper_level(31,1) = 4;
27 upper_level(31,width) = 4;
28 upper_level(48,1) = 4;
29 upper_level(48,width) = 4;
30 upper_level(61,1) = 4;
31 upper_level(61,width) = 4;
32 % set the passengers on their seats
33 upper_level(7,3:4)   = 3;
34 upper_level(9,3:4)   = 3;
35 upper_level(11,3:4)   = 3;
36 upper_level(13,3:4)   = 3; % business class
```

```
upper_level(7,6:7)    = 3;
upper_level(9,6:7)    = 3;
upper_level(11,6:7)    = 3;
upper_level(13,6:7)    = 3;
upper_level(7,9:10)   = 3;
upper_level(9,9:10)   = 3;
upper_level(11,9:10)   = 3;
upper_level(13,9:10)   = 3;
for i=16:2:40
upper_level(i,2:3)   = 3;
upper_level(i,6:7)   = 3;
upper_level(i,10:11) = 3;
end


upper_level(42:2:46,2:3)   = 3;    % economy
upper_level(42:2:46,10:11)=  3;
upper_level(50:2:64,2:3)   = 3;
upper_level(50:2:64,10:11)=  3;
upper_level(42:2:70,5:8)   = 3;

% upper_level(46,4) = 2;
```

8.2.1 Static field

Listing 16: Static field: static_field.m

```
a=100000;  %value for walls, seats
passenger_number = 200;

length = 71;
width  = 12;
[upper_level]=field_make_upper_level(length,width); %make the upper level matrix
field_upper_level_sketch(upper_level,passenger_number,0);

A=upper_level;

[l,w]=size(A);

B=a*ones(l*w,l*w);

anzahl_neighbours=4;
neighbour=[0,-1     %define the neighbours
     -1,0
     1,0
     0,1];



%make matrix B
          for nsp=1:(l*w)
               z=floor((nsp+w-1)/w);
               sp=rem(nsp,w);
               if(sp==0)
                    sp=w;
```

```
29                end
30
31                for neigh=1:anzahl_neighbours
32                    if((z+neighbour(neigh,1))~=0 && (z+neighbour(neigh,1))...
33                                        ~=(l+1) && (sp+neighbour(neigh,2))~=0 &&...
34                                        (sp+neighbour(neigh,2))...
35                                        ~=(w+1)) %%überprüfen ob index in gültigem Bereich
36                    if(A(z+neighbour(neigh,1),sp+neighbour(neigh,2))==1)
37                        B(nsp,(z+neighbour(neigh,1)-1)*w+(sp+neighbour(neigh,2)))=1;
38                    end
39                    end
40
41                end
42            end
43
44
45
46
47  %FLOYD-WARSHALL ALGORITHM
48  for k=1:(l*w)
49      for i=1:(l*w)
50          for j=1:(l*w)
51              B(i,j)=min([B(i,j);(B(i,k)+B(k,j))]);
52          end
53      end
54  end
55
56
57
58  ex=[2 1 %define exits
59      2 w
60      14 1
61      14 w
62      48 1
63      48 w
64      70 1
65      70 w];
66
67
68  %iterate through all the exits and take the corresponding values
69  for exit=1:8
70      nsp2=0;
71      x=ex(exit,1);
72      y=ex(exit,2);
73      pos=(x-1)*w+y;
74  for z2=1:l
75      for sp2=1:w
76          nsp2=nsp2+1;
77          C(z2,sp2,exit)=B(pos,nsp2);
78      end
79  end
80
81  end
82
83  C
```