Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Train Boarding Platform Simulation

Daniel Graf & Matthias Krebs

Zurich
December 2010

Eigenständigkeitserklärung

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

Daniel Graf Matthias Krebs

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.

Daniel Graf Matthias Krebs

Contents

1 Individual contributions

As we met and exchanged ideas several times per week, both of us have contributed to most parts of this project. We contributed equally to the model description, development, testing and final documentation. A specific separation can be made here:

Matthias Krebs was in charge of finding and implementing an appropriate forces model. He invested a lot of time to get the agents moving around smoothly and keeping always an appropriate distance to all obstacles and trains.

Daniel Graf designed the different test cases and executed them during several days on the remote workstations of D-ITET. He also did the statistical analysis of the collected data.

2 Introduction and Motivations

As our ways to the ETH include about two hours of travelling by train every day, we decided to simulate a specific situation, that we come across twice a day. When trying to board the train on a crowded platform, one can observe a special kind of bottleneck problem.

We often discussed different techniques to find and enter a free door as quickly as possible, so we wanted to simulate and analyze them using Mathworks *MATLAB* as our term project of the course *Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB*.

At first it was essential to take a look at the different factors influencing the behavior of a passenger:

- A single passenger normally just wants to get in as fast as possible. But if the next free door is too far away he might reconsider his choice. If possible, he decides for a door that takes him a little longer to get in, but shorter to walk to. This problem of the optimal doordecision was our main interest and is covered in full detail in the results section. It is of special interest to separate one's personal optimum from the global one. For the train crew it is important that *every* last passenger gets in as soon as possible, which does not have to correlate with the personal door decisions.

- Some travellers might have used this train so often before, that they are able to predict where the doors will arrive when the train stands still. That way they can reduce their queuing time and have a higher probability to discover an unassigned seat.

- Although it is hard to guess from outside the train, it is important to know how many seats are still empty in the preferred coach. If the platform is very crowded, passengers might need to take another door, which forces the train to wait a lot longer.
- In bad weather there is an additional factor to consider. The passengers want to wait under the roof of the platform just until short before the doors of the train open and the boarding can begin.
- There are sometimes groups, like forms or gym clubs, that prolong the boarding process additionally, as they have reserved seats in one coach and therefore want to enter all together through the same door.

The goals of this project are as follows:

- Simulate a big crowd of passengers in a well-proportioned scenario like in Zurich or Sargans.
- Find a model that describes the behavior of a passenger in terms of movement and deciding for a door.
- Study the effects of a variation of the different simulation parameters.

3 Description of the Model

3.1 Model Overview

We intended to find a suitable model that is general enough to handle a lot of train-specific additions but also precise enough to simulate decisions and movements of individual passengers.

Many general models use analytical approaches like in [3] and are then able to solve wave equations exactly. But most of them are limited to really simple scenarios, like a semicircular crowd in [3]. As our setup with many doors and obstacles would be much too complex to be solved exactly we decided to do a time discrete simulation.

In previous work cellular automatons have also been used quite often, like in [1]. But as we wanted to have precise information about every agent (for distance measuring, obstacle interaction etc.), we decided to use an agent-based approach.

In order to have enough resolution to build doors, obstacles and let the agents move around them, a cellular automaton would have needed at least a level of detail of about one square meter per cell. With a simulation size of $2000m^2$ and a neighborhood diameter of $10m$ a cellular automaton would approximately need as much computation power as an agent based simulation with around 450 agents (asymptotic runtime of the agent based approach $\mathcal{O}(N^2)$, with N the number of the agents). So we preferred the agent based approach, because of its higher precision.

The door decision parameters are inspired from [2]. It gave us a really clean game theoretical approach that has all needed possibilities for us to build on (door availability, door familiarity and additional conditions). It is also claimed in this paper that iterating the decision process has led quickly to a Nash equilibrium. The scenario used there with two doors at opposed sides of a square was much simpler than our boarding platforms.

But also in our simulations with up to 20 doors of 7 different types the decision simulation stabilized quickly. As always when using iterative simplifications, there are probably some border cases, where the simulation would not stabilize. We think that optimal strategies for this game would probably be non-pure strategies. As all agents decide simultaneously it is really unlikely that a optimal strategy would not need any probabilistic decisions.

For simulating the moving behaviour of the agents we found a nice approach in [4] that allowed us to represent all interactions between agents, trains, obstacles and doors.

We have designed large parts of our model in a train-specific way. But many ideas could easily be adopted to other traffic or crowd simulations.

3.2 Trains

A train is a set of wagons. There are three types of wagons, which are different in the way passengers can board. The three types are: First class, second class and bistro wagon. Whereas the first class passengers only board on first class wagons, the second class passengers enter either a second class or bistro wagon. There is also a difference in capacity of the wagons. Second class wagons accept more passengers than a first class or bistro wagon.

Further, we considered in our model, that a train usually arrives later than the passengers do. So the train will move into the station while the passengers are already waiting on the platform.

3.3 Agents

The passengers are the agents in our simulation. They are separated points, each of it with specific properties and a behavioral pattern. The properties are their mass and maximum velocity. Their behavior is more complex and mainly defined by their affinity to class, their mode to choose a door, a limit and frequency of reconsidering the chosen door, etc. (See *section 3.3.2*)

3.3.1 Agent States

During the simulation, an agent possibly changes between three kinds of states: *deboarding, moving, boarding*. For a *deboarding* agent one only has to check whether he can deboard and for a *boarded* agent the he does not act in any way anymore. The situation for a *moving* agent is more complex. In the *moving* state, the agent will consider to change for another train entrance (within the limitation that it is the same train, same class, etc., see again *section 3.3.2*). The agent also has to be moved as well as it has to be checked whether the agent can enter through a door.

3.3.2 Door Selection

The probably most interesting thing about the modeled situation is the question, on which factors the agents base their decision for a specific door. Obviously, this is an individual optimization problem, where the agents try to optimize their conditions. These conditions can be described by several parameters. In the current case, the passenger's distance to go, the number of other passengers with the same intention or the desire for a free seat in the wagon could be these parameters. To use these parameters all together, it is necessary to normalize them, so that you can compare them.

A proper description of an *Exit Selection Model* can be found in [2]. In this paper about evacuation in a fire emergency, they propose to use factors like *estimated evacuation time* (sum of *estimated moving time* and *estimated queuing time*) as well as further factors like familiarity and visibility of the exits and the conditions at the exits. They further propose to separate the exits in some preference groups (depending on the further factors) and so to decide only between the possibilities with the highest preference.

In our model, we implemented these ideas as follows: Mainly, an agent has to reach his final goal, so he has to consider only the doors that lead to the defined destination. In other words, a person A who wants to take the train A only enters a door of train A (we also assume that the person strictly respects the class of the wagons) and a person B who left any train A will not board on train A again, but it will either leave the platform through the subway or board on any other train B. This first selection represents the recommended separation into preference groups.

For the actual door decision, we designed some different functions. We also considered that the order, frequency and number of times an agent can make its decision can influence the result.

The *patience factor* (used by [2]), that prefers the current strategy with a factor of $0 \leq p \leq 1$ over other strategies, is also considered.

The functions to evaluate the door's quality are the following:

Random A possible way to choose for a door is by chance. As one will see in the results this will neither lead to a small final boarding time nor does this mode describe a natural behavior of passengers, so we will not discuss this mode further.

Walk A realistic assumption might be that a passenger always minimizes its way to the door. So if r_i represents the agent i's current position and v_i its velocity and b_k means the position of the door e_k, the agent i's strategy s_i is

$$s_i = \min\left(\frac{d(e_k; r_i)}{v_i}\right) = \min\left(\frac{\|r_i - b_k\|}{v_i}\right) \tag{1}$$

Queuing As another natural behaviour, we considered that an agent chooses always the door where the least amount of other agents are heading to. So if f_k describes the frequency agents can pass the door e_k and $\lambda_i(e_k, s_{-i}, r_i)$ is the number of all other agents heading to the door e_k that are closer to it than agent i (s_{-i} are the strategies of all agents without agent i), then its strategy s_i is

$$s_i = \min\left(\frac{\lambda_i(e_k, s_{-i}, r_i)}{f_k}\right) \tag{2}$$

Sum The logical conclusion is that the natural behavior is a mix of the two strategies *walk* and *queue*. Whereas busy people rather walk a longer distance to reach a door with less other people, a lazy agent rather decides for the closer door without considering the number of people already queuing there. So if $\mu_i \in [0, 1]$ describes the laziness of agent i, its strategy s_i is

$$s_i = \min \left(\mu_k \frac{\|r_i - b_k\|}{v_i} + (1 - \mu_k) \frac{\lambda_i(e_k, s_{-i}, r_i)}{f_k} \right) \tag{3}$$

3.3.3 Groups

While travelling on train, people are often formed in groups. A group is a set of agents that strictly decides for the same strategy. The strategy is either defined to be constant, chosen by the majority of the group or by a group leader. Despite of the difference that group members will not consider the members of the same group by doing their decision, the strategies are similar to the individual agent's strategies.

3.4 Doors

Obviously, a door is a defined area where you change from one part to another part of space. In our simulation the passengers on platform can enter a train or subway respectively do it the inverse way.

Like the agents, the doors have some properties. So each door is determined to be a first or second class wagon entrance respectively a subway entry. Further, there are a frequency and limit of the agent that can pass the door.

3.5 Obstacles

To limit the space where an agent is allowed to move, we included some obstacles in our model. The obstacles describe as well *real* obstacles on the platform (like waiting huts, subways, poles, etc.) as also the borders of the platform. An obstacle is defined by its position, size and period of time it is active. The last parameter is thought to be used to hold the agents back in a defined waiting area until a specific moment.

3.6 Dynamics

We based the movement of the agents on a model already used by [4], who chose it according to a homework from the lecture *Simulations using Particles* by Prof. Petros Koumoutsakos. In this model, the agents are moving like particle in a potential field, so the acceleration of an agent is described by the acting force on it, divided by its mass.

In the model used for our simulation, we based the movement of an agent on the referred model. The force acting on an agent is given by the surrounding (See *section 3.6*). Based on this force, we calculate the actual acceleration.

$$d\vec{v}(t) = \frac{\vec{F_{res}}(t)}{m_{agent}} \tag{4}$$

But the persons will not orbit over the platform like planets, but rather find their way as a smooth line towards their goal. This means that there has to be a limitation of their speed or a friction force, which lets the agent move in a more natural way. So the current velocity is limited to the maximum velocity of the agent.

$$\vec{v}_{new}(t) = \vec{v}(t-dt) + d\vec{v}(t)dt \tag{5}$$

$$\vec{v}(t) = \min\left\{ v_{agent,max}\frac{\vec{v}_{new}(t)}{\|\vec{v}_{new}(t)\|_2}, \vec{v}_{new}(t) \right\} \tag{6}$$

3.7 Forces

There are three kinds of *forces* acting on an agent. The sum of the influence of the doors, obstacles and other agents yields the resulting *force*.

3.7.1 Doors

The main direction of the agent's movement has to be toward its goal, which is always a door (to be precise, this door is door k which is the best strategy s_i for agent i). Therefore, there is a vector $\vec{e}^a_{i,D}(s_i, r_i, b_k)$ that directs from the agent i's position r_i to the position of the door b_k.

$$\vec{e}^a_{i,D}(s_i, r_i, b_k) = \frac{\vec{b_k} - \vec{r_i}}{\left\|\vec{b_k} - \vec{r_i}\right\|} \tag{7}$$

In the case of queuing in front of the door, the agents don't need to approach it up to the point where they have the door's exact position. The area within the space of the door has to be kept empty for eventual agents leaving the door or the next agent boarding it. So we add another vector $\vec{e}^r_{i,D}(s_i, r_i, b_k)$ directing the inverse direction. Its amount is proportional to the inverse of the distance $\left\|\vec{b_k} - \vec{r_i}\right\|$ between agent i and door k and the door range factor d_D.

$$\vec{e}^r_{i,D}(s_i, r_i, b_k) = -\frac{d_D}{\left\|\vec{b_k} - \vec{r_i}\right\|} \cdot \frac{\vec{b_k} - \vec{r_i}}{\left\|\vec{b_k} - \vec{r_i}\right\|} \tag{8}$$

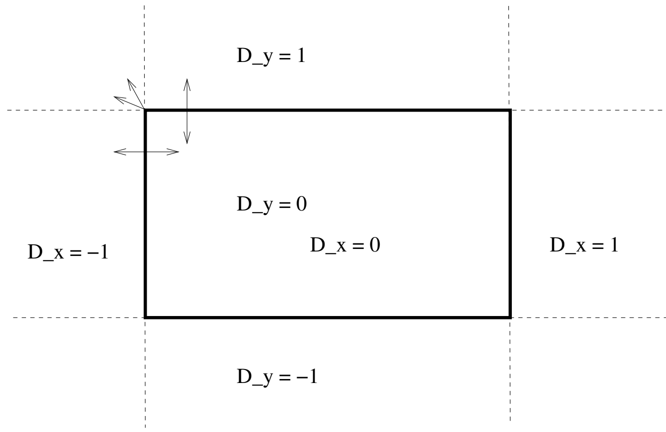

Figure 1: At a rectangular obstacle (bold), there are nine sectors (dashed). An agent is always retractet by the nearest point of the obstacle (arrows).

Now, the force yielding by the door is

$$\vec{F}_{i,D}(s_i, r_i, b_k) = k_D \cdot \left(\vec{e}^{\,a}_{i,D}(s_i, r_i, b_k) + \vec{e}^{\,r}_{i,D}(s_i, r_i, b_k)\right) \tag{9}$$

where k_D is the factor that describes the general strength of the door forces.

There will be an equilibrium point where the force becomes zero if $d_D = \left\|\vec{b_k} - \vec{r_i}\right\|$.

3.7.2 Obstacles

There are several ways for modelling obstacles. In the simulation from [4] it is proposed to model a wall or any other obstacle as a set of fixed point. Then a retraction force between each agent and each obstacle point can be calculated.

To decrease the amount of calculation we decided to introduce another model where an obstacle is represented by a rectangle. Then, an agent is always retracted either by the side respectively edge of the rectangle, which is closest to the agent. This model is valid for agents inside as well as outside the obstacle. To actually decide which side or edge of the obstacle has to be considered for the retraction, the space around each obstacle is split in eight sectors outside and one sector inside the obstacle. The force will be as shown in *figure 1*.

The vector $\vec{l}_{i,j}(\bar{\bar{D}}_{i,j})$ from the agent i's position $\vec{r_i}$ to the point of the obstacle which is the nearest to it, can be calculated as

$$\vec{l}_{i,j}(\bar{\bar{D}}_{i,j}) = \left(\bar{\bar{D}}^{PD}_{i,j}\vec{o_j} + \bar{\bar{D}}_{i,j}\vec{d}_{O,j}\right) - \bar{\bar{D}}^{PD}_{i,j}\vec{r_i} \tag{10}$$

where $\vec{o_j}$ is the center point of obstacle j, $\vec{d}_{O,j}$ the dimension of the rectangle. $\bar{\bar{D}}_{i,j}$ is a matrix determined for each sector around an obstacle as follows (Compare also *figure 1*).

$$\bar{\bar{D}}_{i,j} := \begin{bmatrix} D_x & 0 \\ 0 & D_y \end{bmatrix} \tag{11}$$

with

$$D_x = \begin{cases} -1 & \text{if } r_x < o_{j,x} - d_{O,j,x} \\ 1 & \text{if } r_x > o_{j,x} + d_{O,j,x} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

and

$$D_y = \begin{cases} -1 & \text{if } r_y < o_{j,y} - d_{O,j,y} \\ 1 & \text{if } r_y > o_{j,y} + d_{O,j,y} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where $o_{j,y}$, $o_{j,y}$, $d_{O,j,y}$, $d_{O,j,y}$ are the components of o_j respectively d_{O_j}. $\bar{\bar{D}}_{i,j}^{PD}$ is the same matrix than $\bar{\bar{D}}_{i,j}$ but with the elementwise absolute values.

Finally we find the force $\vec{F}_{i,O}(r_i, o_k, d_k)$ acting on agent i caused by obstacle j

$$\vec{F}_{i,O}(r_i, o_k, d_k) = -k_O \cdot \vec{l}(\bar{\bar{D}}_{i,j}) \cdot \frac{1}{\left\|\vec{l}(\bar{\bar{D}}_{i,j})\right\|} \tag{14}$$

where k_O represents the general strength of the obstacle forces.

3.7.3 Agents

Agents should keep a certain distance between them, so that the queuing procedure becomes realistic. Therefore we have to implement a retraction force between them that is proportional to the inverse distance between each pair of agents.

With some investigations, we concluded that it makes sense if the agents behave similar to atoms in a crystal lattice (compare *figure 2*).

So the agents in front of the door compose a realistic crowd. Therefore, the force $\vec{F}_{i,A}(r_i, r_h)$ acting on agent i caused by agent h is calculated as

$$\vec{F}_{i,A}(r_i, r_h) = \left(\frac{1}{\|\vec{r}_h - \vec{r}_i\|^2} - \frac{d_A}{\|\vec{r}_h - \vec{r}_i\|^3} \right) \cdot \frac{\vec{r}_h - \vec{r}_i}{\|\vec{r}_h - \vec{r}_i\|} \tag{15}$$

where d_A is the agent's required space.

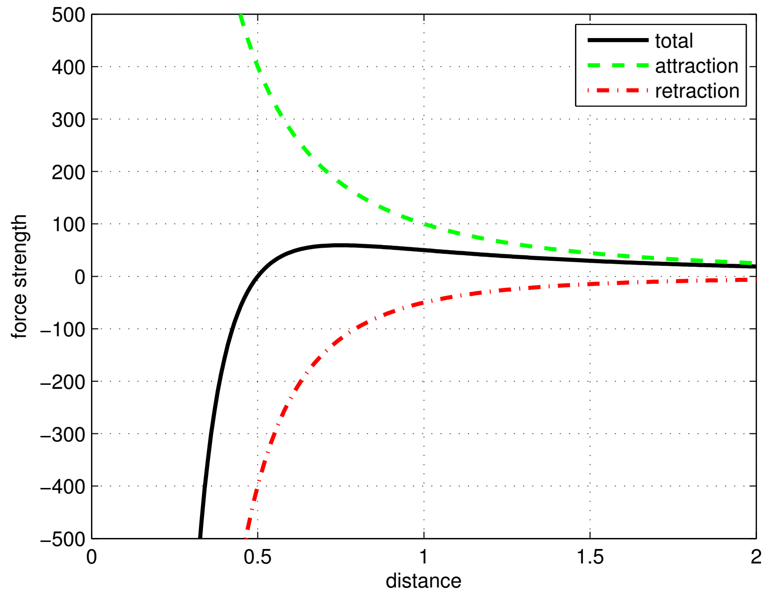

Figure 2: Qualitative diagram of the force between agents, similar to a crystal lattice.

3.8 Simulated Situations

Finally, the introduced objects have to be placed to simulate a specific situation on a train station. The position of the train as well as the number and class of its wagons have to be defined. Further, the subways and any obstacles on the platform have to be placed. In a final step, agents and all their personal properties need to be set.

In our simulation, we analyzed the situation where the train consists of SBB EuroCity wagons. Each train has two first class, a bistro and three second class wagons.

3.8.1 Two Trains

Our first situation represents a common situation at Zurich HB. There are two trains parallel at the same platform. There are entrances to the subway as well as some obstacles (piles) on the platform. There are travellers changing from one train to the other, some are leaving through the subway and others enter one of the trains.

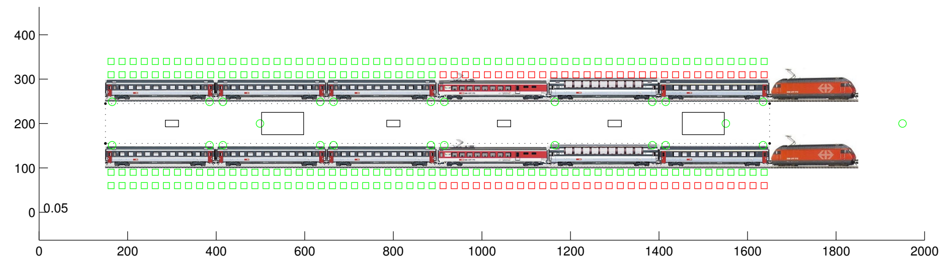

Figure 3: Two Trains Situation. The dashed line defines the waiting area for the passengers. The small green circles mark the doors. The small squares along the train represent the available seats in the train by their color.

3.8.2 One Train

The other situation we implemented in our simulation represents any station where a single train arrives. Some passengers leave the train and exit through the subway. Some other passengers are waiting anywhere on the platform and are going to enter the train as soon as the outcoming passengers finished deboarding.

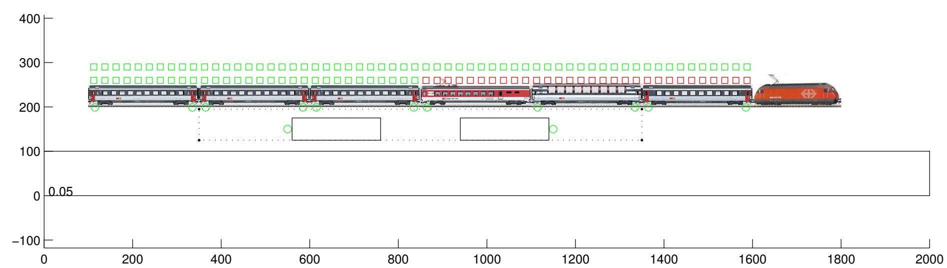

Figure 4: One Train Situation.

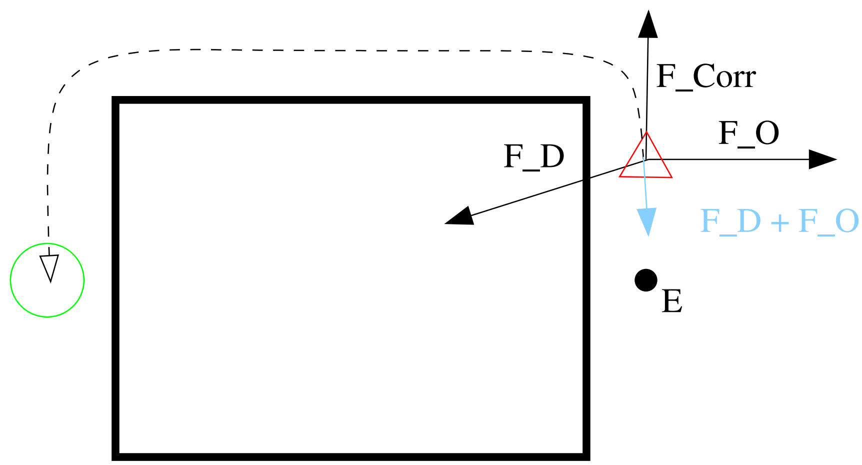

Figure 5: The agent (red triangle) has to move to the door (green circle) which is hidden behind the obstacle. If there are only the forces $\vec{F}_D$ and $\vec{F}_O$, the agent will end up in the equilibrium point E. The correction force $\vec{F}_{Corr}$ leads the agent around the obstacle.

4 Implementation

This chapter first alludes on some specific add-ons that had to be made to guarantee the model's proper functioning. Afterwards it should give a small overview about the created Matlab-Files and their functionality.

4.1 Model Add-ons

4.1.1 Forces Correction

As the agents sometimes ended up in a dead-end on their way to their goal, we had to improve our force-model by a correction force. If an agent is close to an obstacle, and the sum of the door force $\vec{F}_D$ and obstacle force $\vec{F}_O$ becomes very small, this correction force leads the agents around the obstacles as shown in *figures 5 and 6*. In the first of the two discussed situations, the door is "hidden" behind the obstacle. The resulting force $\vec{F}_D + \vec{F}_O$ leads to a dead-end equilibrium point on the right side of the obstacle. Therefore the correction force has to lead the agent around the closest edge of the obstacle.

The mathematical description of the force $\vec{F}_{Corr}$ is

$$\vec{F}_{Corr} = \text{sign}(\vec{F}_O \diamond \vec{F}_D) \cdot k_C \cdot \frac{\vec{o}_O}{\|\vec{o}_O\|} \tag{16}$$

where $(\vec{F}_O \diamond \vec{F}_D)$ is a two dimensional vector product

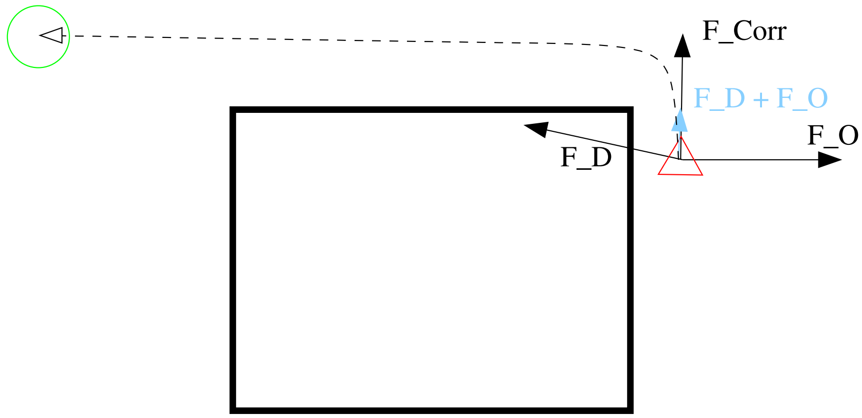

Figure 6: Different to *figure 5* the door is not directly behind the obstacle, but the straight way is blocked, too. As the forces $\vec{F}_D$ and $\vec{F}_O$ would already lead the agent the right way around the obstacle, but are not powerful enough, the correction force $\vec{F}_{Corr}$ will lead the agent around the obstacle's edge.

$$\begin{bmatrix} a \\ b \end{bmatrix} \diamond \begin{bmatrix} c \\ d \end{bmatrix} = ad - bc \tag{17}$$

that is also used by *Prof. Dr. C. Glocker* in the lecture *Mechanik III.*
$\vec{o}_O$ is a vector orthgonal to the obstacle retraction force vector $\vec{F}_O$, actually it is 90 degrees turned clockwise.

$$\vec{o}_O = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \vec{F}_O \tag{18}$$

In the example of *figure 5*, $(\vec{F}_O \diamond \vec{F}_D)$ is negative and $\vec{o}_O$ is directed to south, so $\vec{F}_{Corr}$ is a vector with direction north and its value is the correction force factor k_C.

In the situation in *figure 6*, the door is not directly behind any object (in fact, this door represents a train entrance, whereas the door in *figure 5* is always a subway exit). In this case the correction force has to act the other way round. Because the vector product $(\vec{F}_O \diamond \vec{F}_D)$ will have the inverse sign than in the first example, the correction force for agents attracted by a train's door is

$$\vec{F}_{Corr} = -\text{sign}(\vec{F}_O \diamond \vec{F}_D) \cdot k_C \cdot \frac{\vec{o}_O}{\|\vec{o}_O\|} \tag{19}$$

4.2 Initialization

We split the initialization into several files. To simulate a specific testcase, the parameters have to be set in *run_testcase.m*. This file calls all the specific initialization files before the actual simulation starts.

init_globals This file defines a lot of constants that define the matrices that are used during the simulation.

init_szenario As we defined two scenarios, there are two of these files (one for the one train situation, the other for the two train situation). In these files, all the parameters for the agents, doors and obstacles are set, referring to the given specifications. There are some specific files to distribute the agents properly on the platform and to define the groups. By setting the agent's initial positions randomly, one has to ensure that they are not set inside of any obstacle. The group initial file sets all members of the same group close together on the platform.

init_statistics To collect the data during the simulation, a lot of numbers have to be stored into matrices. These matrices are initialized in this file.

init_style This file defines what the simulation should display and which data should be stored where. It is possible to display either a map with the agent's position or some specific curves. The simulation can also run without displaying anything.

4.3 Simulation

The *simulation.m* file includes the time iteration for the simulation. The order of actualizing the agents' state follows a random permutation. Mainly, the following procedures are done in every iteration step. They will be explained in the following sections.

- Update the agent's **state**: Check whether an agent is currently boarding or deboarding.
- Update **strategy**: Check whether the agent should change his door decision.
- Calculate **forces**: Determine the movement for every agent.
- **Move** agents: Set the new position for every agent according to the calculated force.

- **Plot** the current situation.
- **Save** data: Actualize statistics and save data and/or pictures.

4.4 Update state

One parameter of the doors is the time, until the next agent can pass it. If a door is available, the *agent_update.m* file checks first whether there is an agent inside that wants to deboard. If so, the agent's state will change to *moving*. This means that the agent is now on the platform and interacts with its surrounding.

If there are no more deboarder left, it is checked, whether there is an agent close enough to the door to enter it. If so, the agents state changes to *boarded* and the agent will be set on a seat in the coach. If the agent occupies the last available seat of the coach, the doors' activity will turn to inactive, so that no other agents will try to enter this coach.

4.4.1 Look for a seat

The files *agent_seat_search.m* and *coach_seat_search.m* are used to check where the agent takes a seat inside the train. These files will only be valid for the two introduced situations *One Train Model* and *Two Train Model*, because the two doors of every coach have to be named explicitly. The first file checks whether there is any space left on the chosen coach. The second file than finds the first available compartment for the agent.

4.5 Update strategy

In the initialization file a door decision frequency gets specified. The file *doordecision_frequency.m* then determines how often the agent can change its mind during the current step. The door decision frequency can be any positive number $|f| \in \mathbf{R}$.

The strategy will only be checked for *moving* agents. Their limit of allowed redecisions must not be reached. Then the *place in queue* as well as the *remaining distance* for each door will be calculated for agent i. The decision for the best door will be made by using the agent's kind of *door selection* (See *section 3.3.2*). The number of available redecisions will have decreased. If agent i is member of a group, the whole group's strategy will be the chosen door.

4.6 Calculate forces

The file *calculate_forces.m* calculates the force acting on every agent as it is described in *section 3.7*. As a result, the force acting on agent i will be stored so the next procedure will determine the agent's movement.

4.7 Move agent

Also the moving procedure follows the rules described in the description of the model (see section *section 3.6*) The steps walked by the agents are calculated using the explicit Euler formula.

4.8 Plot

As the scene has changed during the recent time step, the current situation gets displayed. There are two plotting modes and an off-switch:

- Plot map
- Plot graph
- no plot

Plot map shows the situation at the station with the agents as dots on the platform (*figure 7*). This will be very convenient to observe the behaviour of the full crowd of the agents.

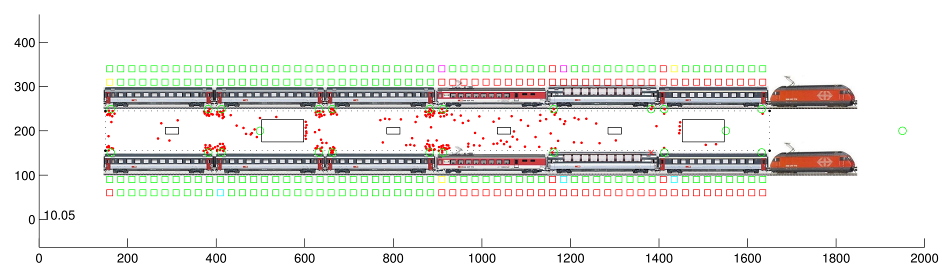

Figure 7: In the *mapping mode*, the agents are displayed as red points (violet if they are member of a group). The doors are marked by a green circle (respectively red cross if an agent has recently used it). Obstacles are displayed with black lines (dotted if it is inactive). The train consists of images by MAERKLIN model railway coaches. The scale is represented in decimeter, as the image of the trains can not be scaled to less than one Matlab plot unit per pixel.

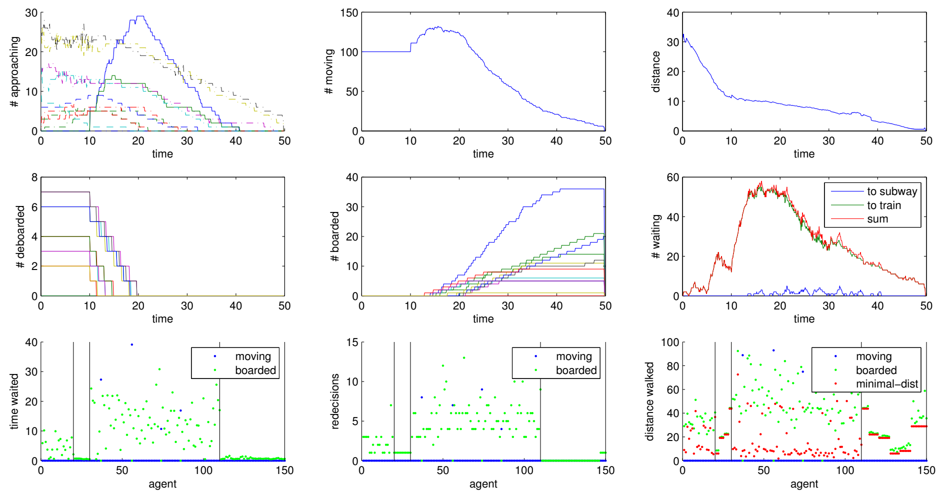

Figure 8: In the *graph mode*, these nine subplots will be displayed during the simulation.

Plot graph displays six time diagrams and three agent diagrams. As an example we take *figure 8* that represents a standard situation from the *One Train Model*. See *section 5.1.2* for a detailed explanation and interpretation of these plots.

- **Approaching** shows how many agents are heading to which door. The solid line represents the subway. The stability of the Nash equilibrium can be itentified by the lines' variation.

- **Moving** shows how many agents are on state *moving*, i.e. the number of agents walking on the platform. It can easily be identified, that the agents start deboarding at $t_0 = 10s$.

- **Distance** indicates the mean remaining distance to the agents' chosen door.

- **(De-)boarded** plots the number of agents that already (de-)borded at this door. The capacity of the door can be recognized by the frequency the curves rise.

- **Waiting** shows the number of agents queuing at a door. The plot differs between agents boarding on a train or leaving through a subway.

- **Time waited** shows for every single agent the time he had to queue. The agents are sorted from left to right (*first class boarding, f.c. deboarding, s.c. boarding, s.c. deboarding*). As long as an agent is still moving, its dot will be plotted in blue. Boarded agents are plotted in green. The first class agents had to queue less time. As most agents boarded within 50 seconds, there are almost no blue dots left.

- **Redecisions** is the recent number of how many time each single agent preferred a new door. Again, the value for first class agents is smaller.
- **Distance walked** shows for every single agent the distance it actually walked (blue/green) as well as the linear distance from the agents start position to its final goal. Of course the actual distance has to be at least as big as the linear distance. The deboarding agents' distances are almost equal to their minimal distances.

In case of interest, every plot can easily be plotted in a single plot by calling the specific *plot_saved_$\cdots$.m* file.

4.9 Save Data

The *save_data.m* file is responsible to store all statistics values of any interests. Most of these values can also be identified on the *Graph Plot*.

5 Simulation Results and Discussion

5.1 General Results

Before analyzing the effect of single parts of the simulation we noticed some interesting properties of our simulation in general. The graphs in this section have been calculated using our default setup that is described in *section 5.2*.

5.1.1 Observations on the map

Beginning The simulation starts 10 seconds before the train reaches its final position and opens its doors. At that time the waiting passengers are randomly distributed over the platform. *Figure 9* shows this situation as a 2d-plot.

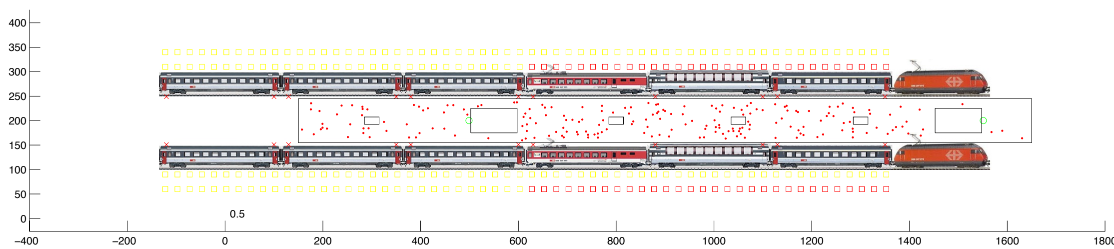

Figure 9: Situation at the beginning of the simulation. The yellow squares mark compartments that are already used by two passengers. The red ones are already full, which represents, that there are fewer seats in the first class and bistro coaches. All passengers are distributed randomly over the waiting area but outside of all the obstacles.

Door opening During the first 10 seconds the passengers start approaching a door, although all doors are still moving. Then the doors open and first all leaving and changing passenger get off the train. As no agent is able to board yet, they start forming semi-circular crowds around the doors, which is visible in *figure 10*.

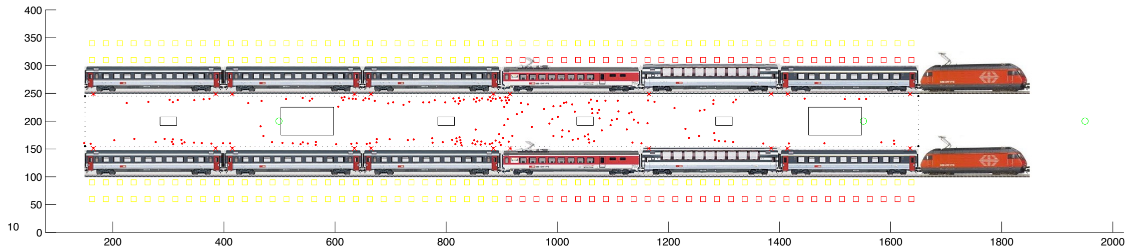

Figure 10: Train station after 10 seconds of simulation time. Around the most popular and most central doors semi-circular crowds start to get formed.

Boarding After 30 seconds, most of the agents have reached their favorite door and some of them already boarded. Around all popular doors the passengers build semi-circular waiting crowds and are in balance between moving closer to the door and not getting to close to any other agent or getting pressed to the train. *Figure 11* also shows how the compartments inside the train are getting occupied.

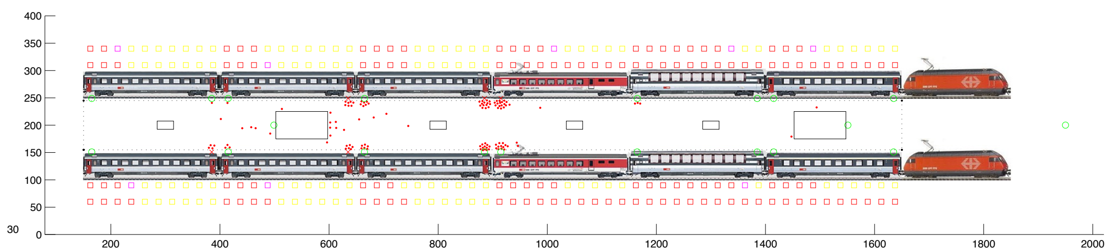

Figure 11: Train station after 30 seconds of simulation time. People are waiting in front of their favorite door and the amount of free seats has decreased on all coaches (red compartments symbolize no more free seats)

Final state 65 seconds after the train arrived the last agent enters the train. When the simulation ends after 90 seconds, all agents have found a seat. It is important to remark that both bistro coaches and one second class coach have been filled up to the last seat. That means that some agents had to redecide for another door at another coach, which caused the train to wait longer. In the end the compartment allocation looks like in *figure 12* .

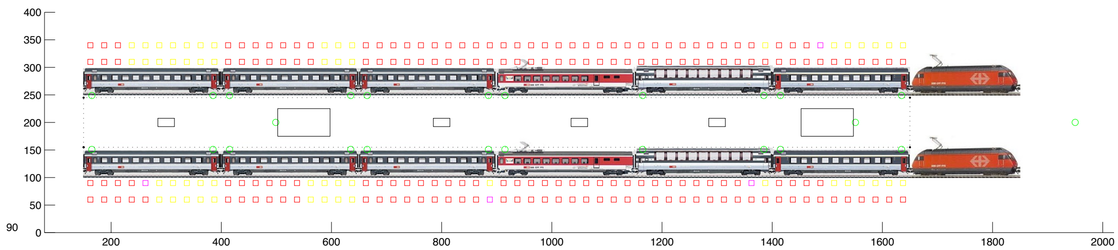

Figure 12: End of simulation after 90 seconds. All passengers boarded and some coaches are full.

5.1.2 Insights out of the graphs

The graph view explained in *figure 8* provides a comprehensive overview of all the recorded data. The following paragraphs show some of these observations.

Approaching agents *Figure 13* contains a lot of information about how the agents decide for a door. During the first few seconds the curves are quite unstable which means that the door decision has not stabilized yet and the Nash equilibrium has not been found yet. But after the doors opened the curves get quite steady and the number of approaching agents decreases almost linearly.

The highest peak belongs to the leftmost exit which is located more central than the other two exit and therefore is approached by most of the second-class passengers. About 10 seconds after the door opening this curve starts to decrease again, which means that already more passengers are entering the exit than there are new agents approaching, who just got off the train. About 45 seconds after start of the simulation, there are already no more leaving agents on the platform.

Something interesting happens after 46 seconds or 47 seconds respectively. There are only four doors, where there are still agents approaching. The two light green lines belong to the rightmost second class door of each train, whereas the dark green lines correspond to the left doors of the two bistro coaches. The bistro coach is filled up, so all the waiting agents there have to redecide. Of course all of them decide for the door, that is just 5 meters away and still has some empty seats. In the figure it is visible how the agents swap from the dark to the light green lines. A few seconds later one of the two second class coaches is full too. The one on the opposite train had enough free seats to accommodate all waiting agents.

Figure 14 contains a detailed view of the agent distributions between these four most popular doors: the one on the left of the bistro-coach and right-most door of the second-class coaches. Here it is clearly visible, that the Nash equilibrium is not very stable at all. There are up the three agents, who redecide from one door to another, when meanwhile three other agents decide to do the opposite. And after just a few time steps they decide back. This confirms our assumption from the model overview (see *section 3.1*), that an optimal door-decision-strategy would need to be a mixed strategy and therefore could not be found using the iterative algorithm proposed in [2].

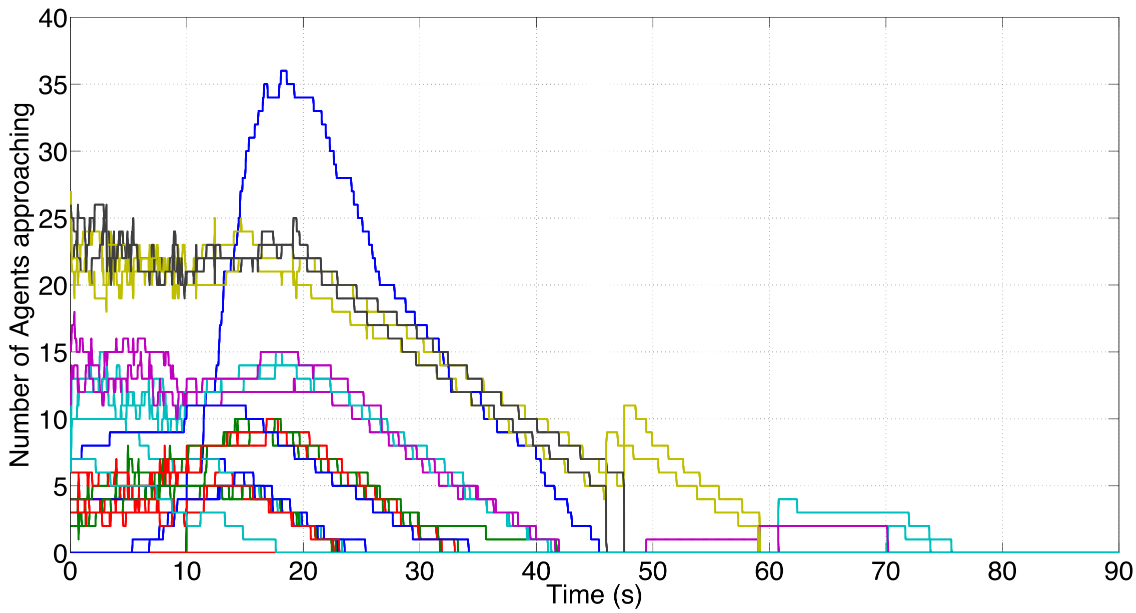

Figure 13: Course of the agents' door approaching behaviour

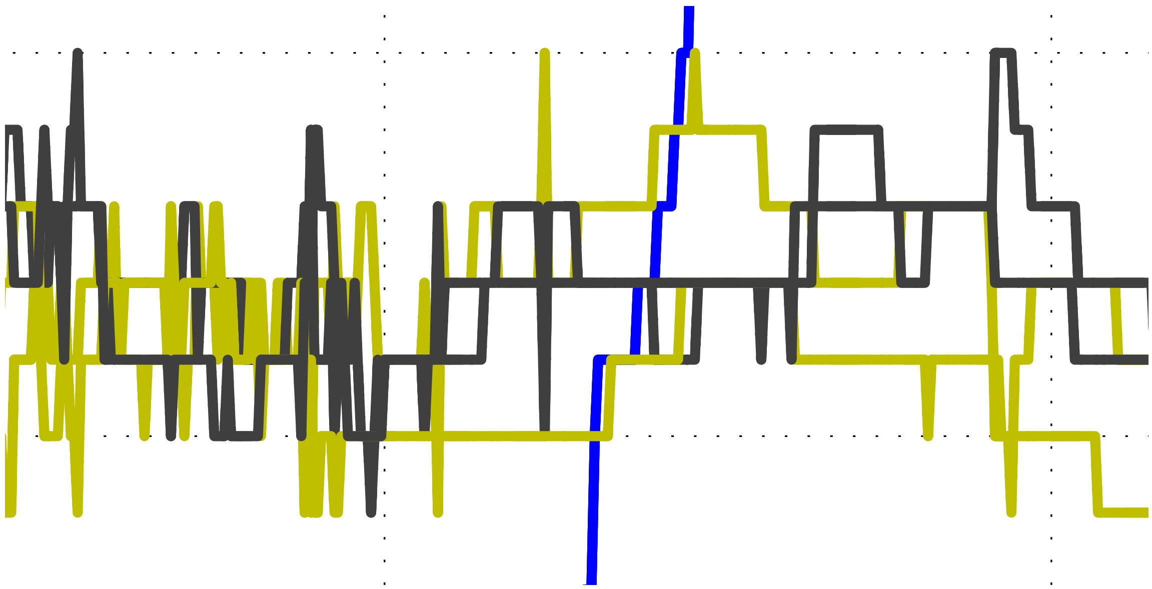

Figure 14: Detail view of the approaching agents to the two left-most doors of the bistro coaches (dark green) and the two right-most doors of the second class coaches (light green) in the time interval $[10s, 15s]$.

Moving and waiting agents *Figure 15* displays how many agents are on the platform. This amount is constant until the doors open. Then it increases until more agents board than get off the train and finally it decreases in a hyperbolic way until the last agent boards after about 70 seconds. After 45 seconds about 90% of the agents have already boarded. So a large part of the time that the train has to stay at the station is caused by only a few agents that either have to walk too long or redecide too late.

The number of waiting agents as shown in *figure 16* consists mainly of the boarding agents. The peak is at about 20 seconds, when most boarding agents arrived at their preferred door, but are still unable to enter the train, because deboarding has not finished yet. Afterward it decreases almost linearly, as agents can board homogeneously. For the exiting agents there are some short queues in front of the subway entrances from about 20 to 40 seconds.

Figure 17 shows the waiting time of each agent separately. The agents are grouped by class and mode. One can see that the leaving agents almost never have to queue. As the density of first class passengers per door is much lower, they also have to wait less than second class travellers. The passengers that change the train also have to wait less for a couple of different reasons:

One reason is, that they spend the first ten seconds or even longer inside the train, which we do not count as waiting because we are only interested in how long the agents queue before they can board the train. When they get off and walk to the opposite train, the queue there will already have shortened a bit. Another reason is, that the leaving doors are equally distributed, which means that some agents get off somewhere near the end of the train, where almost no passengers queue on the opposite side.

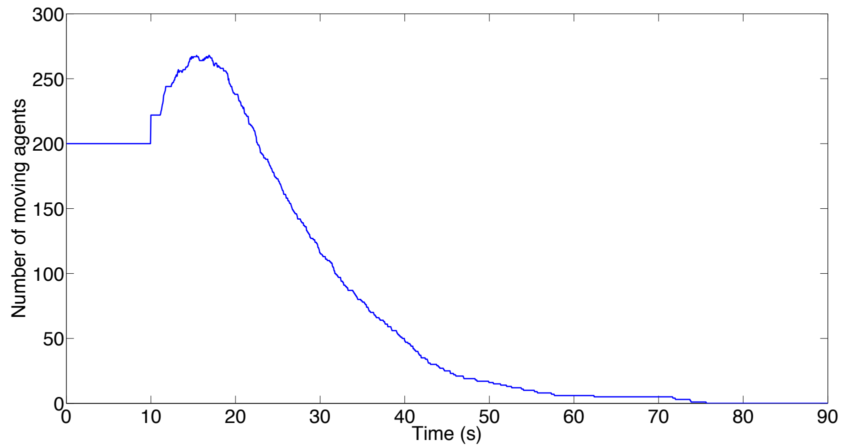

Figure 15: Agents situated on the platform

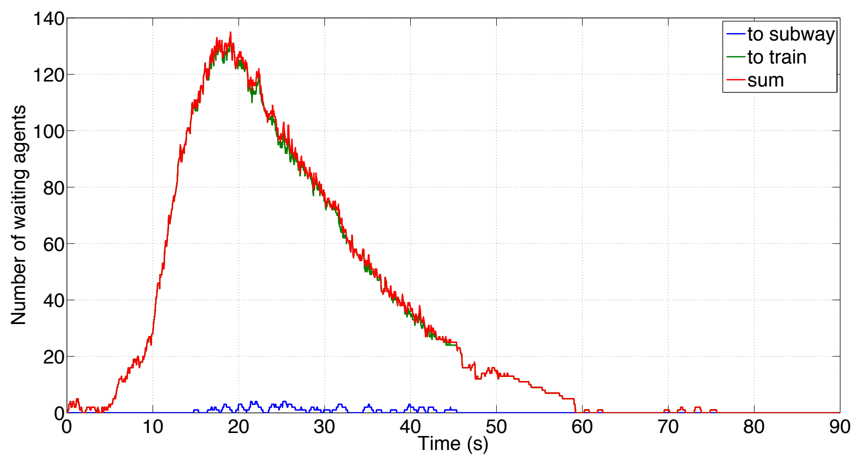

Figure 16: Agents that have to wait just in front of their preferred door

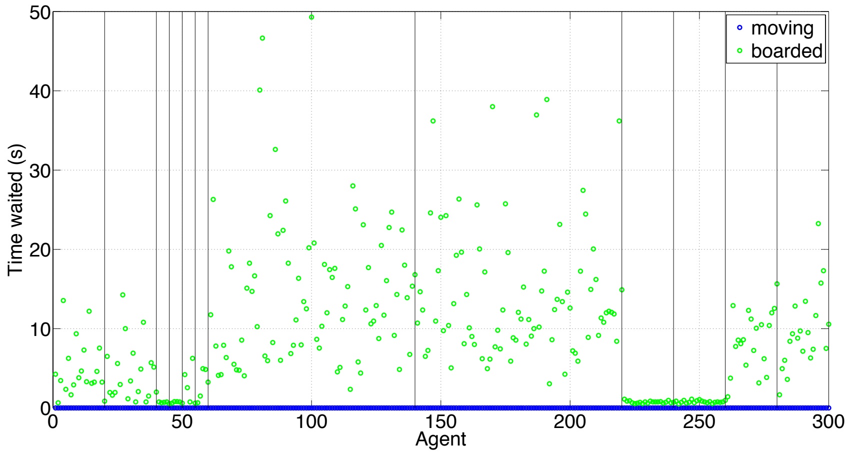

Figure 17: Waiting time per agent. The vertical lines separate the 12 types of agents: [1 - 6] first class; [7 - 12] second class; [odd] train one; [even] train two; [1,2,7,8] boarding agents; [3,4,9,10] deboarding agents; [5,6,11,12] changing agents

Boarding and deboarding agents These two graphs (*figure 18* and *figure 19*) just show what we expected. The doors let people in and out at almost regular time steps.

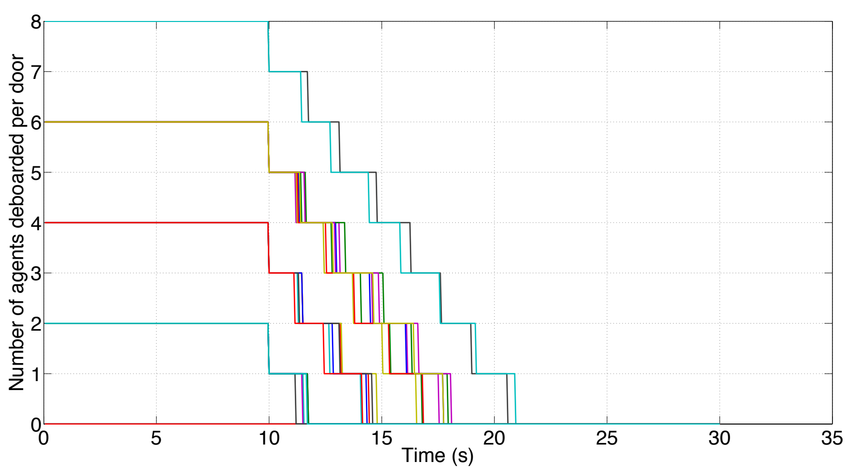

Figure 18: Agents that are waiting behind a door in order to get off the train

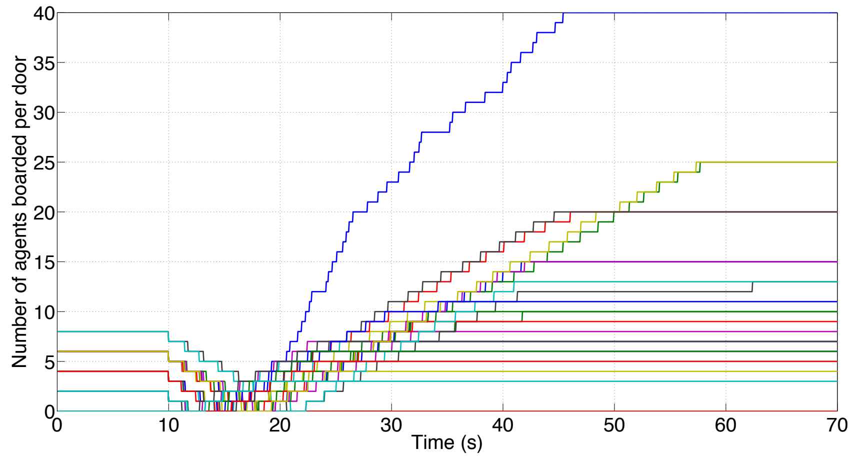

Figure 19: One graph per door, describing the number of agents inside this door, not considering the agents that stay inside permanently

Door redecisions Similar to the distribution of the waiting time, *figure 19* shows that almost all redecisions are done by boarding agents. It is no surprise that this value correlates with the waiting time, as it is much harder to decide, if there are a lot of other passengers queueing in front of an agent, who thinks about redeciding for a door that is further away.

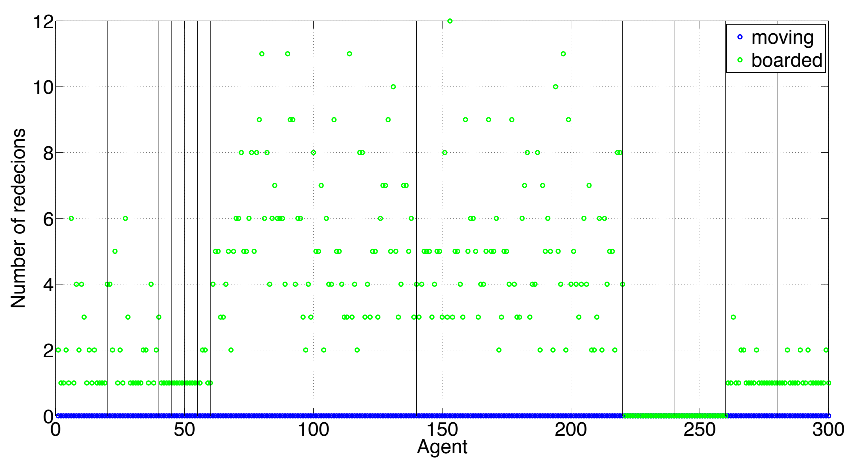

Figure 20: Number of redecisions per agent

Average distance The plot of the average distance between an agent and his chosen door (*figure 21*) reflects the different stages of the simulation. At first the distance falls linearly, which means that all agents move at full speed toward their chosen door. Then, as the first agents start to queue and new agents deboard, the distance decreases less and less. The two small peaks at about 46s and 47s reflect the two bistro coaches getting filled up. But as the next free door is just 5 meters away the peaks are quite small. It gets worse, when the second class coach has reached its maximum capacity and all of the still moving agents have to walk another twenty meters.

In the agent distribution in *figure 22* one can see, that again the boarding agent have the biggest walking overhead. Deboarding agents are able to walk almost straight to their exit. Most changing first class agents can walk directly to their preferred door too. This is caused by the topology of the train station, where the left subway obstacle blocks primarily second class travellers.

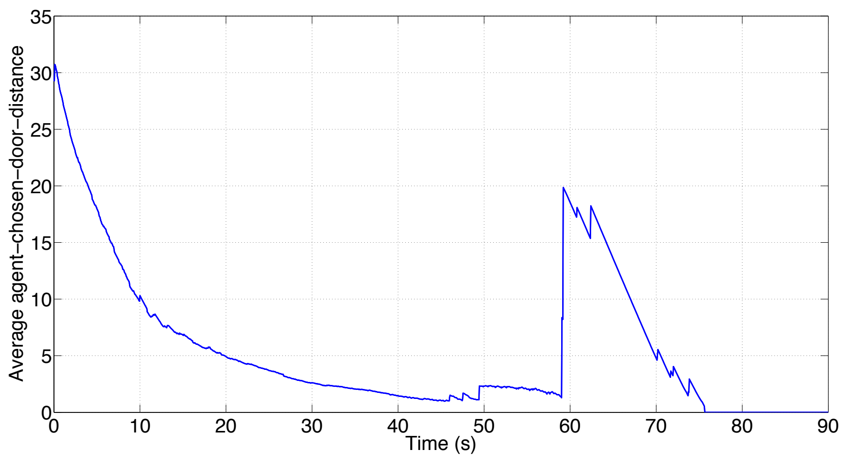

Figure 21: Average distance between all agents and their chosen door

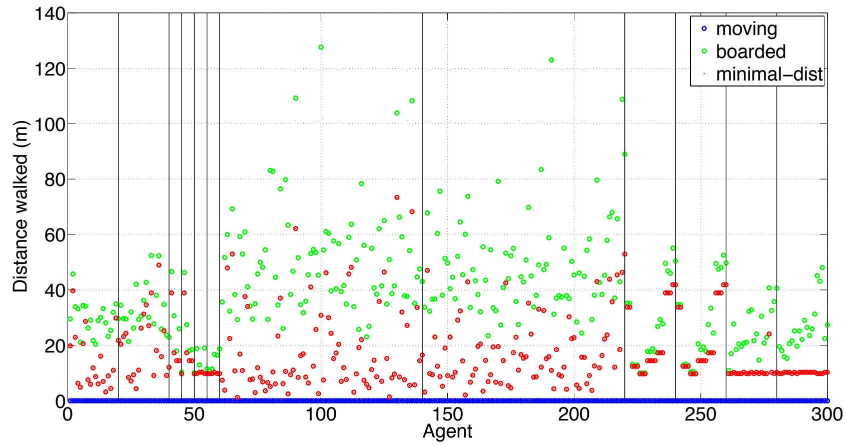

Figure 22: Comparison between linear line distance and the walked distance

Simulation Errors In some cases, where we varied our default parameter values, sometimes imprecisions in our simulation have occured. It could happen, that a few agents get stuck somewhere in a dead spot where the resulting force is almost equal to zero. At first this could happen when the agent approached an obstacle that was in line with him and his chosen door. Than the retraction of the obstacle erased all the attraction of the door at a certain point. Because of that we introduced a correcting force that eliminated almost all of this situations (see *section 4.1.1*).

Another source of errors are inappropriately high time steps or velocities. Our implementation does not include an explicit obstacle collision detection. Therefore the obstacle retraction is calibrated high enough, such that an agent with default velocity can not move so far in a default time step, that he would cross any obstacle or train border.

5.2 Simulation Variables and Result Indicators

In order to study the influence of many simulation parameters on the boarding process, we specified a base case and a series of other test cases to compare it to *Table 1* lists all parameters that can easily be varied in the initialization file. The values written in *italics* marks the default value. The abbreviations in square brackets for each parameter are often used in the code and in the appendix .

The default values are our assumptions about a regular, crowded train station. In the following sections we varied each group of parameters separately and compared the results to the base case presented above.

Szenario	
Number of Trains	one train [OT] *two trains [TT]*
Amount of agents	
People on Platform [PoP]	few (50 passengers per train) *many (100 passengers per train)* too many (200 passengers per train)
People deboarding [Pd]	few (25 passengers per train) *many (50 passengers per train)* too many (100 passengers per train)
People already seated [Pas]	few (50 passengers per train) *many (200 passengers per train)* too many (300 passengers per train)
People attributes and behaviour	
Class ratio (first class share) [FCR]	*0.2*
Maximum velocity Distribution (mean and variance)	$1\frac{m}{s}$ $1.5\frac{m}{s}$ $\mathit{1.5 \pm 0.5}\frac{m}{s}$ *(default)* $2.5 \pm 1.5\frac{m}{s}$
Group ratio	*no groups* 20% grouped 50% grouped
Door decision parameters	
Door decision mode [DDM]	walk *sum* queue wait random
Lazyness coefficient [LC]	$\{0, 0.1, ...\mathit{0.5}, ...0.9, 1.0\}$
Door decision frequency [DSF]	1 decisions per second *20 decisions per second* 100 decisions per second
Limitation of the amount of door decisions [DL]	$\{1, 10, \infty(\mathit{default})\}$
Patience factor for door redecision	$\{0.5, \mathit{0.9}, 1.0\}$
Time	
Time until waiting area opens [TW] and time until doors open [TD]	$1s/1s$ (instant) *5s/10s (default)* $10s/20s$ (late)
Simulation Duration	*90s*

Table 1: different simulation parameters

Statistical measurements To compare the results of the different test setups, we defined a number of statistical values, which we calculated in every run of the simulation. Of course this is only a small subset of all the data, which was recorded during each simulation. We have chosen the values, which we think are most relevant in terms of how a train company could measure the quality and efficiency of their train stations.

All the values have been calculated considering only the agents that boarded a train during the simulation time, including the ones changing from one train to another. This is important because we are only interested in how long it takes until the train is able to depart and do not care, if some agents still are on the way out. Additionally, as exit doors have a much higher capacity than train doors, it is important that these doors are not considered when studying the distribution of the agents over the doors or the average waiting times.

So these are the evaluated values:

- maximal boarding time (latest time that any agent entered a train) [maximum]
- boarding time [average / standard deviation]
- covered distance (the distance an agent walked from its initial position or his leaving door) [average / standard deviation]
- waiting time (the amount of time the agent spent waiting close to his chosen door) [average / standard deviation]
- number of redecisions [average / standard deviation]
- distribution of boarding agents per door [standard deviation]

5.3 Test Series

Each following subsection presents a detailed analysis of one of the specified parameters. The statistical measurements of all test cases used for this section can be found in the appendix. The full test results, including the matlab workspace at the end of each simulation, are available on the homepage (see *section 8.1*).

5.3.1 Number of Agents

At first we varied the number of agents per train. The diagrams in *figure 23* show, that all measurements increase the more agents are added. It is not surprising, that more people result in longer ways, more redecisions, longer waiting queues and so on. When comparing the train scenarios, it is at first not clear, why the agents need to walk, redecide and wait less, if they are in the two train situation. But it gets reasonable, if we consider, that there are only in this scenario additional changing agents. They have to wait less and can just walk across the platform, without trying to decide for a door while the train is still moving.

5.3.2 Door Decision Modes

It was our main goal to study different door decision algorithms. So we started first with just the five different methods described in *section 3.3.2* with default parameters. At first it is important to remark, that the values for the *random*-mode are not quite comparable (see *figure 24*). There the agents board so slowly, that after 90 seconds most agents are still on the platform and therefore not considered in these diagrams.

In most categories the *queue*-mode seems to be the most promising one, especially if we look at the maximal boarding time. Although the difference to the walk and the sum approach are not that big regarding the average boarding time, the train would be able to leave about 13 seconds earlier if all passengers would chose their door only based on the length of the queues. This method also distributes the agents much more uniformly over the doors than the other modes. The only downside of this mode is the big number of door decisions. This probably makes it really hard to apply this mode, if we would try it in reality.

The *sum*-mode is the algorithm, that we think describes best, what human passengers do. They normally look around and decide for the nearest door. Only if it is to crowded there, they will take the longer way to the next door. At first sight, it would make sense, if this was the optimal method. So why is it not optimal to minimize the sum of the walking and the queueing time?

- This would only be optimal from the point of view of a single agent and not in the global scale. For many agents this method seems in fact to be faster than the *queue*-mode, as the mean boarding time is just one second higher with a higher deviation.
- If a door A is much closer at the beginning than a door B, this does not imply, that an agent can board earlier there, even if both doors have no queues at all. Then during the entrance of the train and the initial deboarding, so from about 15 to 20 seconds, no boarding is possible anyways. This gives agents that move to doors further away the chance to catch up, while the others just queue before the door.
- Last and probably most important is the fact, that with this method some central coaches get filled up completely and the agents queuing there have to

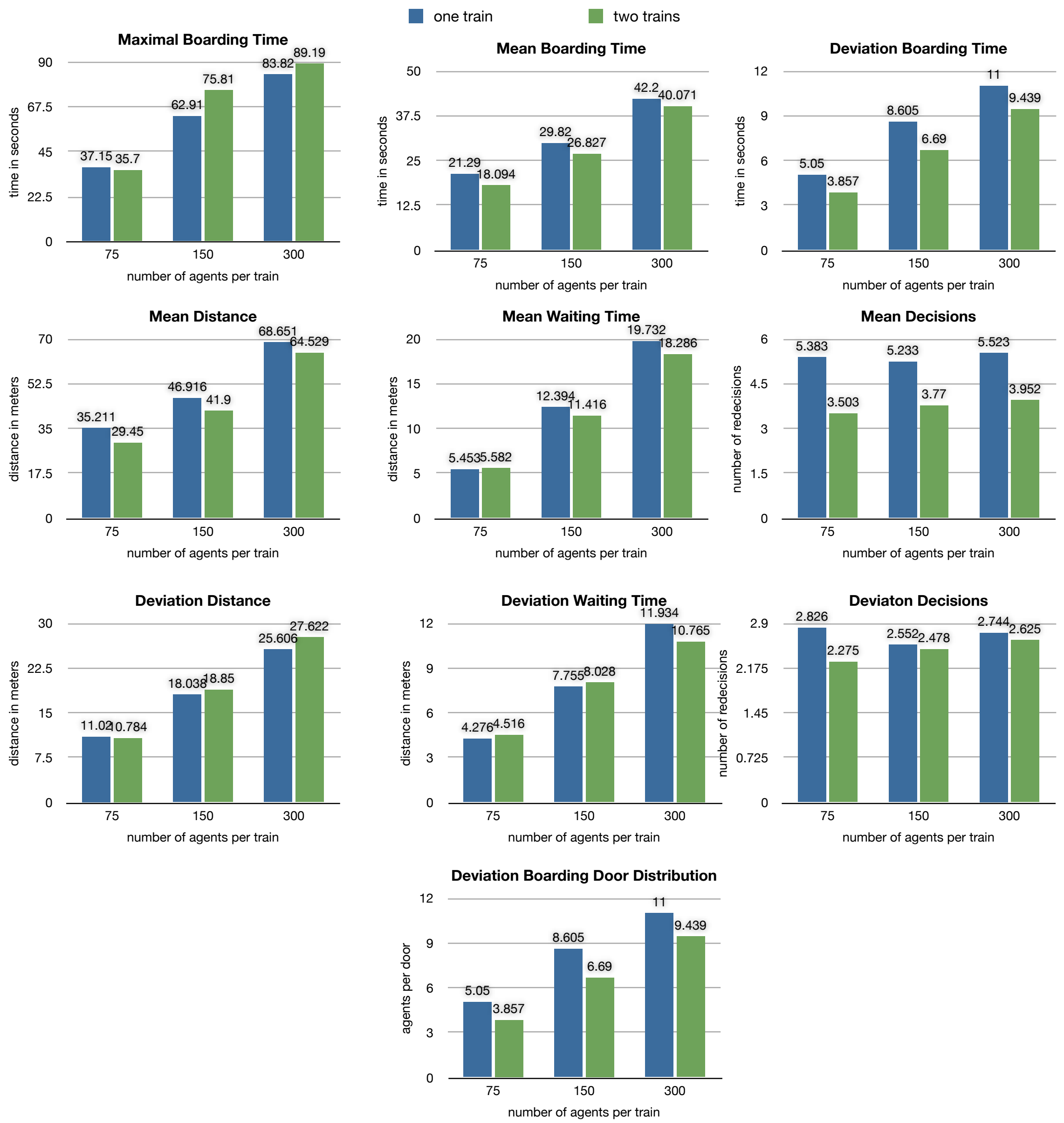

Figure 23: Variation of the number of agents that are moving through the train station.

walk to another door. Unfortunately the agents do not see this event coming and all of them stay queuing there until the last seat is taken. The *queue*-mode adapts better to this issue, as the agents choose the doors more equally and therefore no coach gets filled up completely.

For the *walk*-method the same arguments as for the *sum*-methods apply, just in a intensified way. Here it strikes the eyes, how minimal the number of redecisions is. The *walk*-mode is the only algorithm, which can choose based on the initial position of the agent and does not have to take the other agents into consideration. The only two situations, in which such an agent has to decide again, is when either his chosen coach is full or when he has to make a large detour around an obstacle or a crowd and he comes across a new nearest door.

The *wait*-method seems to do a very good job in his main concern: minimizing the waiting time. The downside of this approach is the much longer distance and the huge number of redecisions. As said before the waiting time for the *random*-algorithm is not really comparable here, as most of the agents don't even get near the door before the simulation stops.

5.3.3 Laziness Coefficient Optimisation

As we have seen some remarkable differences between the *queue*-, *sum*- and *walk*-modes, we wanted a more precise separation and simulated with different agent laziness coefficient in 0.1-steps. So laziness 0 corresponds to the *queue*-mode, 0.5 is exactly the same as the *sum*-mode and laziness 1.0 means minimizing the walking time.

For both, the maximal boarding time and the average boarding time, a laziness factor of 0.1 is optimal. 0.1 is even better than 0.0, because then agents might change their minds, if another door, that was not their chosen door but is closer, gets free. This is of special importance at the end, when only a few agents are left, hence the large gap in the plot of the maximal boarding time (see *figure 25*).

This leads to quite an astonishing "paradox": The more an agent tries to minimize his walking time, the more he actually has to walk. This can be explained by the fact that, with the default parameters and the *lazy*-mode, no coach gets filled, which saves a few agents to walk some additional, unscheduled extra-distance to another coach.

5.3.4 Door Decision Frequency

In order to see whether the simulation depends on the frequency of the door decisions, we varied the number of iterations per seconds of the door decision process (see *figure 26*). The effect seems to be visible, but not statistically significant.

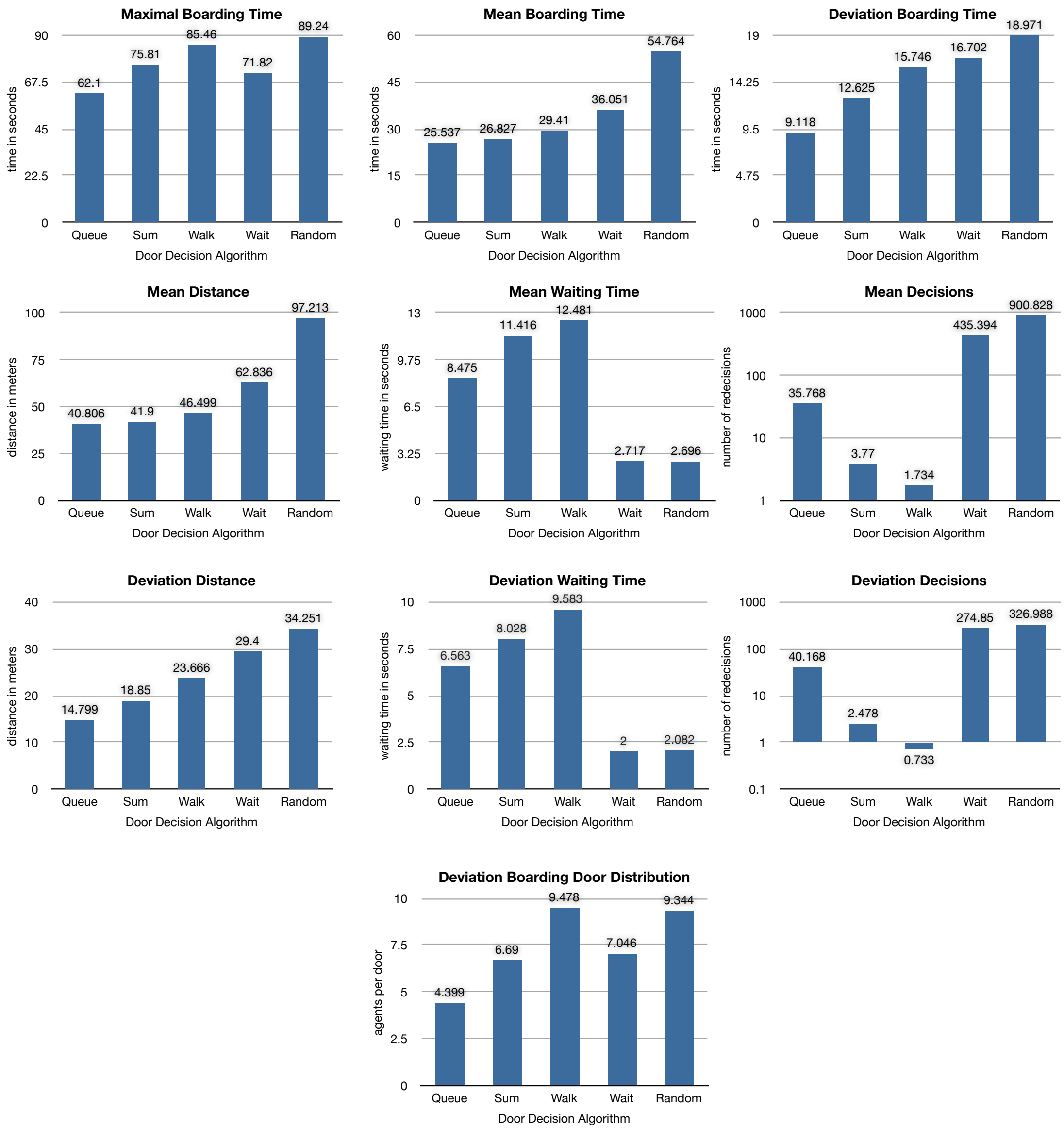

Figure 24: Influence of the different door decision modes

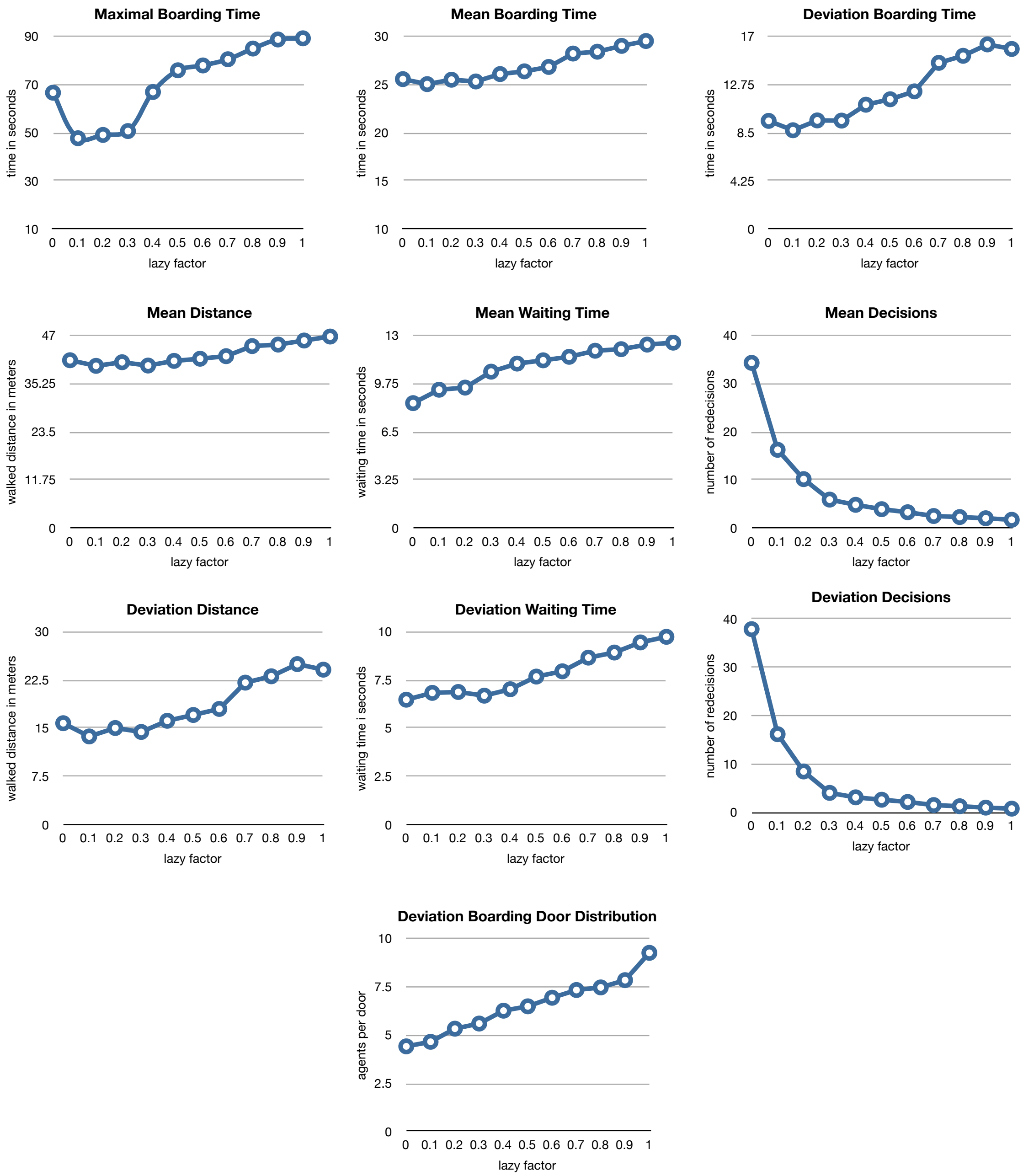

Figure 25: Variation of the laziness coefficient

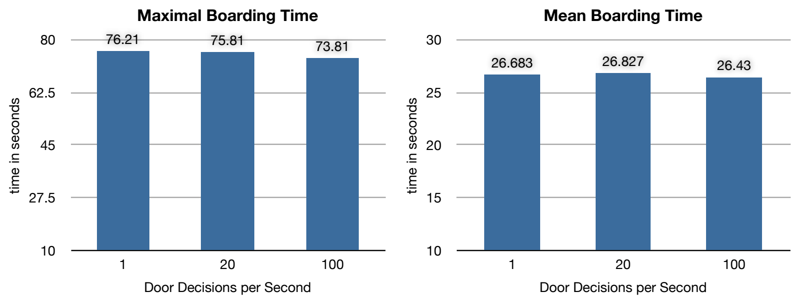

Figure 26: Boarding times with different update frequencies of the door decision process

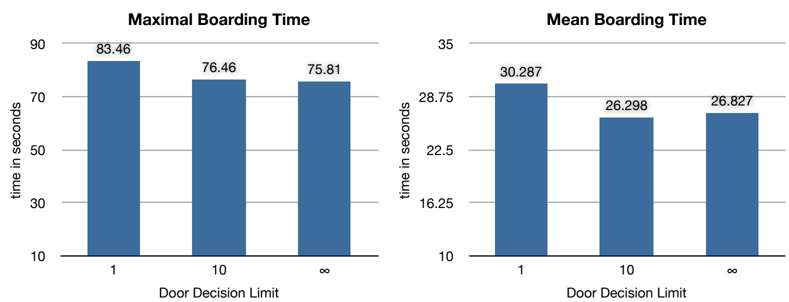

Figure 27: Boarding times with different limitations of the number of door decisions per agent

5.3.5 Door Decision Limit

We also limited the number of times an agent is allowed to change his mind and approach a different door. *Figure 27* shows, that it is important for the agents to have more than just one chance to decide for a door. The difference between 10 and unlimited decisions is not significant.

5.3.6 Patience

A variation of the patience factor as in *figure 28* seems not to have a significant influence on the boarding times. A small patience factor of 0.5, which means that agents redecide only if another door is expected to let them in twice as fast, seems to be a little bit faster, if we look at the mean boarding time.

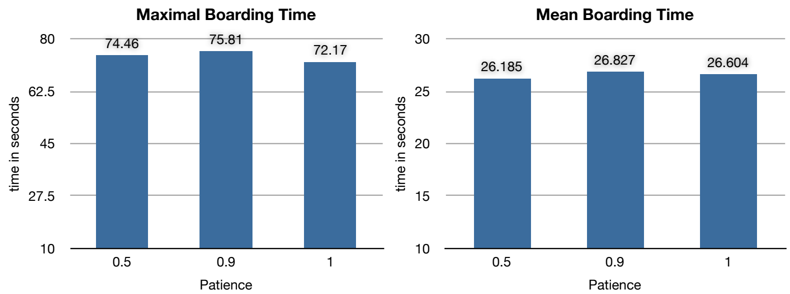

Figure 28: Different patience factors. 0.5 means that the agent compares all the estimated times to reach another door with half of his current one.

5.3.7 Velocity

Higher velocities reduce the final boarding time drastically (see *figure 29*). However the average boarding time does not decrease that much. This is the case, because during the first 15 seconds no agent can board anyway, so a higher velocity only helps the agents to get closer to the door before boarding starts. When we distribute the maximal velocity per agent randomly around $1.5\frac{m}{s}$ with standard deviation $0.5\frac{m}{s}$ this seems to be a little bit faster compared to the situation, where every agent has the same maximal velocity of $1.5\frac{m}{s}$. But the difference is not significant.

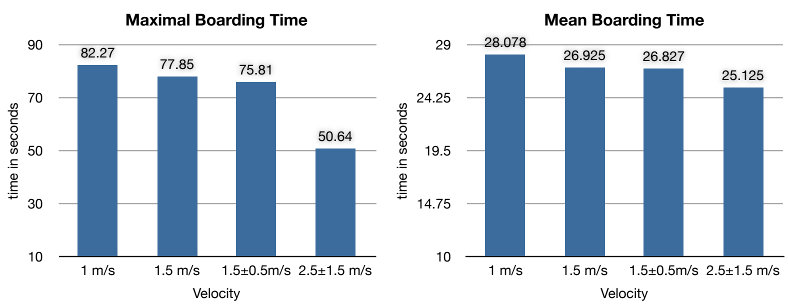

Figure 29: Variation of the maximal velocity of the agents

5.3.8 Groups

As expected having a lot of groups on the platform increases the mean boarding time and the average distance (see *figure 30*). The different amount of redecisions might be caused by the fact, that in a group all agents have to redecide together. But again, the influence seems to be quite small and the differences not statistically significant, considering that only five simulations of each group have been made.

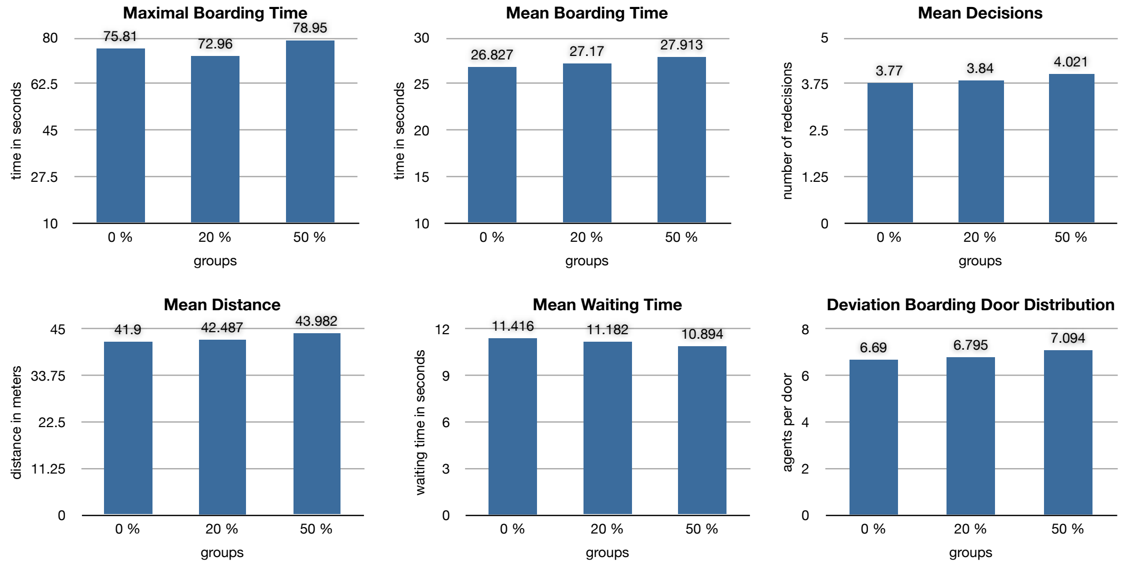

Figure 30: Different group ratios. Each group consists of 10 agents.

5.3.9 Simulation Start relative to the Door Opening

When varying the point in time when the waiting area opens and when the doors of the train open, one can notice some significant differences (see *figure 31*).

There is a complex reason, why the maximal boarding time for the default start is bigger than for the two other starts. If the train arrives quicker, then the agents have no time to group around the still driving doors. Therefore there are more agents that decide for the doors at the end of the train. These doors are not considered that often in the default case because they are too far away at the beginning.

With the third case, where the train still drives for 20 seconds, the opposite happens. The agents pile up in front of the door, but are not fast enough to follow the train. The agents build a kind of tractrix behind the train until their shortfall gets too big and they redecide for the next door arriving. Because this process takes longer in this case, the agents stay more evenly distributed on the boarding platform and therefore also chose their boarding door more evenly. The duration of this initial train-following-process also correlates with the number of door decisions and the covered distance.

If we take a look at the mean boarding time we see, that a longer time period with closed doors increases it by about 7 seconds for every additional 10 seconds. This means that the additional ten seconds of preparation on the platform helps the agents to reduce the boarding time by three seconds.

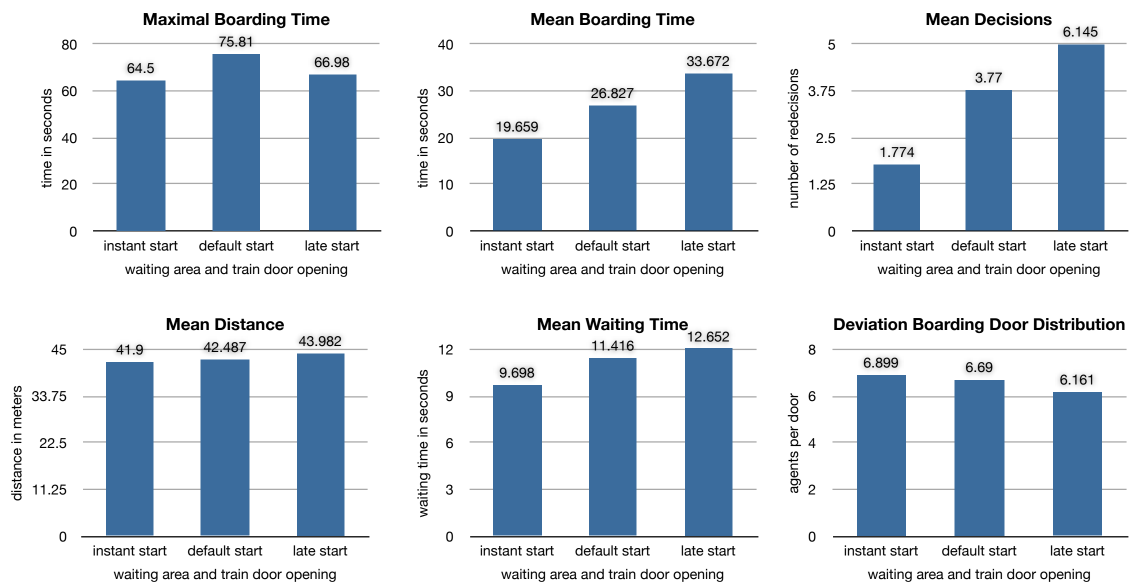

Figure 31: Variation of the moment where the simulation starts relative to the train-entrance

5.3.10 Waiting Area

In our default case the waiting area is quite big and sprawls over almost the entire platform. In this test case we compared the base case to a smaller waiting area that only covers the space between the two central subway exits. We did this comparison for both train stations (see *figure 32*). The mean boarding time increases with the smaller area, because they are not able spread as quickly over the platform as in the base case. It also results in a longer way for the average agent.

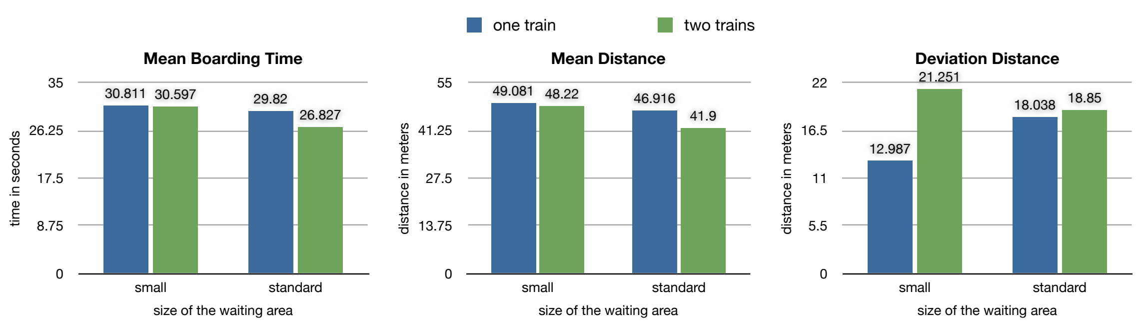

Figure 32: Two sizes of the waiting area simulated in both train stations

6 Summary and Outlook

Conclusions The main aim of our project was to find an appropriate model that is able to simulate big crowds moving in a train station. We intended to find some specific parameters that affect the boarding behaviour of the passengers and with that an optimization of the minimal boarding time and to maximize distribution of the passengers in the different coaches.

The model we implemented was able to simulate the intended situations (i.e. Zurich, Sargans) on a platform quite well. The model is able to include various scenarios, like different exits and obstacles.

With the introduced decision modes for the passengers, we were able to analyze the effect of different decision parameters on the resulting final situation. Therefore, a strategy, which mainly consists of taking the door with the smallest queue, leads to a minimal final boarding time. All others characteristics as mean distance walked, mean time waited and the uniform distribution over all coaches are optimized by this strategy. Other factors like the frequency of decision making, limitations of maximum redecisions as well as the patience factor had a rather small influence.

Discussion We calculated five simulations in 36 different test cases. This already took several hours of computation. But in order to get more meaningful results there would be many more simulations needed. Our test suite was big enough to separate many differences clearly, but in some unclear cases more data samples would have been helpful.

There are also some imprecisions in our model. If the scenario is quite complex the force model does not imply a way from the agents position towards his chosen door. Another problem is, that if a crowd gets bigger and bigger the agent density increases to an unrealistic high level. That way it can happen, that an agent experiences a force so heavy, that he gets "pressed through the wall" and is unable to get out again. An additional obstacle-agent-comparison might be helpful there.

All in all our model and our implementation fulfills our expectations and was able to provide some interesting results.

Outlook There is always room for possible extensions:

- An extended door decision mode, which also takes the number of free seats behind the door into account, could help to optimize the boarding time even further.
- If one would like to simulate more complex scenarios, like complete train stations, with many more trains and obstacles, it would be essential to use much

more object oriented design in the implementation. That way every door would be linked to a coach and every coach would be part of a train. This would make the whole setup process much easier.

- In areas with more obstacles the force model would need to be extended with some sort of shortest-path detection like Dijkstra's algorithm. The combination of such a graph algorithm with the existing model of attracting and repulsive forces could get quite tricky though.
- To make the boarding process more realistic, it would be necessary to simulate the interior of the coaches also with freely moving agents. One could take into account, that boarding an already full coach is still possible, as long going as through the coach at some slower speed is not impossible.
- Another interesting observation is, that passengers might behave completely different in some special situations. For instance, when the train is on the verge of leaving and an agent wants to board, but just realizes that his door is broken, the agent will start running towards the next door. All agent are just able to run for some seconds, so they have to decide wisely when it is dramatic enough to run.
- Our model does not care much about the beginning of the simulation. We just initialize the agents all over the platform and then give them some time to array before the train arrives. It would be interesting to see, where agents, who arrive at the platform several minutes before the entrance of the train, would end up.
- To verify our calculated data some measurement of realistic behaviour would be needed. That way one could investigate the ”real laziness factor”, which would probably be located somewhere above our optimal 0.1.
- Finally it would be challenging to reason about possible ways to persuade real travellers to behave more queue-mode-like. We think that a wise positioning of the subway exits is very important. If the train stations were already built with the distribution of the passengers in mind, the laziness of the agent would not be that important anymore.

7 References

References

[1] Marco Denuder, Dominik Keusch, and Laurent Roux. Emergency evacuation of an airbus a380. *Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB*, May 2010.

[2] Harri Ehtamo, Simo Heliövaara, Simo Hostikka, and Timo Korhonen. Modeling evacuees' exit selection with best response dynamics. *Fire Safety Journal*, 41(5):309–319, July 2006.

[3] Dirk Helbing, Anders Johansson, Joachim Mathiesen, Mogens H. Jensen, and Alex Hansen. Analytical approach to continuous and intermittent bottleneck flows. *Physical Review Letters*, PRL 97(168001), October 2006.

[4] Daniel Zünd and Simon Schmid. Evacuation bottleneck. *Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB*, May 2010.

8 Appendix

8.1 Link to further material

This report, the full matlab source code, a video of the basic test case and all the test case results used for the analysis section are ready to download under this URL: http://n.ethz.ch/∼grafdan/train/

8.2 Sourcecode

This chapter lists the main source code files of this simulation. To run the simulation yourself use these commands:

```
% first edit init_style.m to configure the plotting and output modes
% then specify the testcase id  (2 is base case with two trains)
Testcase = 2
% and start the simulation
run_testcase
% or use the test suite command to run all testcases and save movies,
    statistics and workspace snapshots to the result folder
run_testsuite
```

8.2.1 Starting points

Listing 1: run_testsuite.m

```
test_case_count = 36;
sample_count = 5;

for Testcase = 1:test_case_count
    datestr(now)
    Testcase

    % create file header
    value_names = {'final_boarding_time mean_boarding_time
        std_dev_boarding_time   mean_distance std_dev_distance
        mean_waiting_time std_dev_waiting_time  mean_decisions
        std_dev_decisions std_dev_boarded_per_door  unboarded'};
    dlmwrite(strcat('results/textfiles/', int2str(Testcase), '.txt'),strcat(
        'Testcase Nr. ', int2str(Testcase)), 'delimiter', '');
    dlmwrite(strcat('results/textfiles/', int2str(Testcase), '.txt'),
        value_names(1), 'delimiter', '','-append');

    for isample = 1:sample_count
```

```
        isample
        %clear('global')
        run_testcase
        save(strcat('results/workspace/', int2str(Testcase), '_', int2str(
            isample), '.mat'));


        % collect single value results
        moving_agents = (agent(:,agentSTATE) ≠ agentSTATEmoving)
        boarding_agents = (agent(:,agentMODE) ≠ agent_mode_enter_subway)
        selected_agents = (boarding_agents & moving_agents)

        final_boarding_time = max(stat_moving_time(selected_agents,
            stat_movEND))
        mean_boarding_time = mean(stat_moving_time(selected_agents,
            stat_movEND) - stat_moving_time(selected_agents,stat_movSTART))
        stddev_boarding_time = std(stat_moving_time(selected_agents,
            stat_movEND) - stat_moving_time(selected_agents,stat_movSTART))
        mean_distance = mean(stat_sum_distance(selected_agents,1))
        stddev_distance = std(stat_sum_distance(selected_agents,1))
        mean_waiting_time = mean(stat_sum_waiting(selected_agents,1))
        stddev_waiting_time = std(stat_sum_waiting(selected_agents,1))
        mean_decision = mean(stat_sum_decision(selected_agents,1))
        stddev_decision = std(stat_sum_decision(selected_agents,1))
        stddev_boarded_per_door = std(stat_boarded_per_door(step,door(:,
            doorMODE) ≠ agent_mode_enter_subway))
        unboarded = sum(agent(:,agentSTATE) == agentSTATEmoving)

        dlmwrite(strcat('results/textfiles/', int2str(Testcase), '.txt'), [
            final_boarding_time, mean_boarding_time, stddev_boarding_time,
            mean_distance, stddev_distance, mean_waiting_time,
            stddev_waiting_time, mean_decision, stddev_decision,
            stddev_boarded_per_door, unboarded], 'delimiter', '\t','-append'
            );

    end
end
quit
```

Listing 2: run_testcase

```
% Start simulation here!
% Arrangment for simulation


init_globals;
init_main;
init_style;

```

```
% -----------------------------
% -----Standard-values----------

% Szenario
SZENARIO = TWO_TRAINS;                        % [TT]

% Crowdness
PART_FC = 0.2;                              % [FCR]
AGENTS_OP = MANY_AGENTS_OP;                 % [PoP]
AGENTS_D = MANY_AGENTS_D;                   % [Pd]
AGENTS_SEATED = MANY_AGENTS_SEATED;         % [Pas]

WAITING_AREA = BIG;                         % [WA]

% Behaviour
DOOR_DECISION_MODE = MIN_SUM;               % [DDM]
    LAZINESS = 0.5;                         % [LC]
PATIENCE = 0.9;                             % [P]
DECISION_STEPFREQ = 1;                      % [DSF]
DECISION_LIMIT = agentDECTIMESinfinite;     % [DL]
VELOCITY = 1.5;                             % [VD]
    VELOCITY_VAR = 0.5;
GROUPING = 0;                               % [G]
    GROUP_SIZE = 10;                        % [GS]

% Time
AREA_DELAY = 5;                             % [TW]
DOORS_DELAY = 10;                           % [TD]

% Simulation Stability
TIMESTEP = 0.05;                            % [TS]
FORCES_COEFF = FC_STANDARD;
TIMEMAX = 90;

switch Testcase
    case 1
        SZENARIO = ONE_TRAIN;                       % [OT]
    case 2

    case 3
        SZENARIO = ONE_TRAIN;                     % [OT]
        AGENTS_OP = FEW_AGENTS_OP;                % [PoP]
        AGENTS_D = FEW_AGENTS_D;                  % [Pd]
        AGENTS_SEATED = FEW_AGENTS_SEATED;        % [Pas]
    case 4
        AGENTS_OP = FEW_AGENTS_OP;                % [PoP]
        AGENTS_D = FEW_AGENTS_D;                  % [Pd]
        AGENTS_SEATED = FEW_AGENTS_SEATED;        % [Pas]
    case 5
        SZENARIO = ONE_TRAIN;                     % [OT]
```

```
            AGENTS_OP = TOOMANY_AGENTS_OP;          % [PoP]
            AGENTS_D = TOOMANY_AGENTS_D;            % [Pd]
            AGENTS_SEATED = FEW_AGENTS_SEATED;      % [Pas]
        case 6
            AGENTS_OP = TOOMANY_AGENTS_OP;          % [PoP]
            AGENTS_D = TOOMANY_AGENTS_D;            % [Pd]
            AGENTS_SEATED = FEW_AGENTS_SEATED;      % [Pas]
        case 7
            DOOR_DECISION_MODE = MIN_WALK;          % [DDM]
        case 8
            DOOR_DECISION_MODE = MIN_QUEUE;         % [DDM]
        case 9
            DOOR_DECISION_MODE = MIN_WAIT;          % [DDM]
        case 10
            DOOR_DECISION_MODE = RANDOM;            % [DDM]
        case 11
            LAZINESS = 0.0;                         % [LC]
        case 12
            LAZINESS = 0.1;                         % [LC]
        case 13
            LAZINESS = 0.2;                         % [LC]
        case 14
            LAZINESS = 0.3;                         % [LC]
        case 15
            LAZINESS = 0.4;                         % [LC]
        case 16
            LAZINESS = 0.5;                         % [LC]
        case 17
            LAZINESS = 0.6;                         % [LC]
        case 18
            LAZINESS = 0.7;                         % [LC]
        case 19
            LAZINESS = 0.8;                         % [LC]
        case 20
            LAZINESS = 0.9;                         % [LC]
        case 21
            LAZINESS = 1.0;                         % [LC]
        case 22
            DECISION_STEPFREQ = 1*TIMESTEP;         % [DSF]
        case 23
            DECISION_STEPFREQ = 100*TIMESTEP;       % [DSF]
        case 24
            DECISION_LIMIT = 1;                     % [DL]
        case 25
            DECISION_LIMIT = 10;                    % [DL]
        case 26
            PATIENCE = 1;                          % [P]
        case 27
            PATIENCE = 0.5;                           % [P]
        case 28
```

```
        VELOCITY = 1;                                % [VD]
        VELOCITY_VAR = 0;
    case 29
        VELOCITY = 1.5;                                % [VD]
        VELOCITY_VAR = 0;
    case 30
        VELOCITY = 2.5;                                % [VD]
        VELOCITY_VAR = 1.5;
    case 31
        GROUPING = 0.2;                                  % [G]
    case 32
        GROUPING = 0.5;                                  % [G]
    case 33
        AREA_DELAY = 1;                                % [TW]
        DOORS_DELAY = 1;                              % [TD]
    case 34
        AREA_DELAY = 10;                                % [TW]
        DOORS_DELAY = 20;                              % [TD]
    case 35
        WAITING_AREA = SMALL;                                % [WA]
        SZENARIO = ONE_TRAIN;                              % [OT]
    case 36
        WAITING_AREA = SMALL;                                % [WA]
    otherwise
        'unknown testcase id'
        return;

end

switch SZENARIO
    case ONE_TRAIN
        init_szenario_one_train;
    case TWO_TRAINS
        init_szenario_two_trains;
end
```

8.2.2 Initializations

Listing 3: init_globals.m

```
% In this script, constants (valid for every scene) are defined

%set people (position, goal), doors (size, frequency, capacity), obstacles
%(rectangle position, size, inside/outisde, active/inactive),

% To identify the column of the "people" Matrix,
% those indices are represented by these variables
```

```
agentXPOS = 1;      % 1st column: x-position in meters
agentYPOS = 2;      % 2nd col.: y-position in meters
agentXVEL = 3;      % x-velocity in meters
agentYVEL = 4;      % y-velocity in meters
agentXFORCE = 5;    % Force acting on agent
agentYFORCE = 6;    % ditto
agentMODE = 7;      % Mode: Defines the mode of possible doordecision
agentSTATE = 8;     % current state
   agentSTATEdeboarding = -1;
   agentSTATEmoving = 0;
   agentSTATEboarded = 1;
agentLDOOR = 9;     % Leaving door
agentCDOOR = 10;     % current chosen door for bording
agentMAXV = 11;      % Maximal velocity
agentPATIENT = 12;  % privilege factor for current door
agentLAZY = 13;     % balance between movingtime (lazy) and queuetime (1-lazy)
agentDMODE = 14;    % Mode of deciding for leaving door
   agentDMODEsum_lazy = 1;  % minimum sum of walk(lazy) + queue(1-lazy)
   agentDMODEsum = 2;  % agent decides for minimum sum of walk+queue
   agentDMODEwalk = 3; % agent decides for minimum walk
   agentDMODEqueue = 4;    % agent decides for minimum queue
   agentDMODEwait = 5; % minimum difference between walk and queue
   agentDMODErandom = 6;   % agent chooses randomly
agentDECTIMES = 15; % Max. times of redecision
   agentDECTIMESnone = 0;
   agentDECTIMESinfinite = -1;
agentGROUP = 16;    % 0 is independent
   agentGROUPnone = 0;
% Amount of columns for agent
agentCOLCOUNT = agentGROUP;

% columns of "door" Matrix represent:
doorXPOS = 1;        % 1st column: x-position in meters
doorYPOS = 2;        % y-position
doorMODE = 3;         % identifies a certain "group" of doors.
                          % can only be entered by people with same mode
doorSTATE = 4;       % current time left, til next agent can enter
doorMEANFREQ = 5;    % mean frequency of people entering
doorVARFREQ = 6;     % variation of frequency
doorACTIVITY = 7;     % state of the door (gets set to inactive if coach full)
   doorINACTIVE = 0;
   doorACTIVE = 1;
doorAGENT = 8;       % amount of people enterred the door
                          % (negativ, while people still debording)
   doorAGENTbord = 1;
   doorAGENTdebord = -1;
% Amount of columns for door
doorCOLCOUNT = doorAGENT;
```

```

% columns of "obstacle" Matrix represents:
 obstacleXCENTER = 1;
 obstacleYCENTER = 2;
 obstacleWIDTH = 3;
 obstacleHEIGHT = 4;
 obstacleSTART = 5;     % time value, when obstacle starts to be activated
 obstacleEND = 6;       % time value, when obstacle stops being activated
 obstacleRANGE = 7;     % distance in meters where the retracting force has
     abs = 1
 obstaclePASSABLE = 8;  % should agents be able to move trough the obstacle
     borders

 % Amount of columns for obstacle
 obstacleCOLCOUNT = obstaclePASSABLE;

 % Direction iteration arrays (East, North, West, South)
 xdir = [1,0,-1,0];
 ydir = [0,1,0,-1];

 % Plotting Modes
 plotMAPview = 1;
 plotGRAPHview = 2;
 plotDEFAULT = 3;

 % Video Recording
 videoOFF = 0;
 videoON = 1;

 % Data Export
 data_export_OFF = 0;
 data_export_ON = 1;

 % time when last person boarded
 final_boarding_time = 0;

 % train entrance velocity
 trainVELOCITY = 3;

 % simulation modes
 simulationMODEtest = 0;
 simulationMODEonetrain = 1;
 simulationMODEtwotrains = 2;
```

Listing 4: init_main.m

```
% Szenario
ONE_TRAIN = 1;
TWO_TRAINS = 2;
```

```

% Crowdness
FEW_AGENTS_OP = 50;
MANY_AGENTS_OP = 100;
TOOMANY_AGENTS_OP = 200;
FEW_AGENTS_D = 25;
MANY_AGENTS_D = 50;
TOOMANY_AGENTS_D = 100;
FEW_AGENTS_SEATED = 50;
MANY_AGENTS_SEATED = 200;
TOOMANY_AGENTS_SEATED = 300;

SMALL_AREA_OT = [50, 5];
SMALL_AREA_TT = [50, 5];
SMALL = [SMALL_AREA_OT; SMALL_AREA_TT];
BIG_AREA_OT = [100,7];
BIG_AREA_TT = [150,9];
BIG = [BIG_AREA_OT; BIG_AREA_TT];

% Behaviour (Agents)
MIN_WALK = agentDMODEwalk;      % equal to "SUM" with lazy = 1;
MIN_SUM = agentDMODEsum_lazy;
MIN_QUEUE = agentDMODEqueue;    % equal to "SUM" with lazy = 0;
MIN_WAIT = agentDMODEwait;
RANDOM = agentDMODErandom;

% Force coeffs
FC_STANDARD = ones(5,1);
    FC_obstacleRetraction = 1;
    FC_agentAttraction = 2;
    FC_agentAttractionGroup = 3;
    FC_agentRetraction = 4;
    FC_doorAttraction = 5;

FC_STANDARD(FC_obstacleRetraction) = 10000;
FC_STANDARD(FC_agentAttraction) = 1000;
FC_STANDARD(FC_agentAttractionGroup) = 2000;
FC_STANDARD(FC_agentRetraction) = 2;
FC_STANDARD(FC_doorAttraction) = 20000;
```

Listing 5: init_style.m

```
% setup of special behaviour (non test case specific options, like movie
    output, save paths and plotting mode)

% plotting mode
plotting_mode = plotMAPview;
% plotting_mode = plotGRAPHview;
% plotting_mode = plotDEFAULT;
```

```
if plotting_mode ≠ plotDEFAULT
    my_figure = figure('Position', [20, 100, 1200, 600], 'Name','Simulation Plot Window');
end

% video recording
%video_mode = videoON;
video_mode = videoOFF;
avi_file_dir = 'results/movies/';
avi_file_specs = strcat('simulation-',int2str(Testcase),'-',int2str(isample),'-');

init_video


% Data Export Mode Configuration
data_export_mode = data_export_OFF;
save_dt = 0.5;
save_file_prefix = strcat('results/frames/simulation-',int2str(Testcase),'-',int2str(isample),'-');
save_file_suffix = '.mat';
```

Listing 6: init_video.m

```
% initialising terms for capturing an avi-file
if video_mode == videoON

    avi_file_prefix = 'video_';
    avi_file_date = datestr(now, 'yyyy-mm-dd-HH-MM-SS');
    avi_file_suffix ='.avi';

    avi_filename = strcat(avi_file_dir, avi_file_prefix, avi_file_date, ...
        '_', avi_file_specs, avi_file_suffix)

    aviobj = avifile(avi_filename);
    aviobj.fps = 20;    % Because we simulate with dt = 0.05s
    aviobj.compression = 'Cinepak';
    aviobj.quality = 60; % percent

end
```

Listing 7: init_szenario_one_train.m

```
%simulate one train as on a platform in Sargans

% -------
% GENERAL
```

```
% -------

simulation_mode = simulationMODEonetrain;

% specify scenario (SI units)
border = [0,0,200,45]; %left, bottom, width, height

% time specification
tmax = TIMEMAX;
dt = TIMESTEP;
stepcount = tmax/dt;

% ------
% AGENTS
% ------

class_FIRST = 1;
class_SECOND = 2;
class_count = 2;

agent_type_BOARDING = 1;
agent_type_DEBOARDING = 2;
agent_type_count = 2;

% number of agents as summed up (for later use as index ranges)

agent_part_count = zeros(class_count, agent_type_count);
agent_part_sum = zeros(class_count, agent_type_count);

agent_part_count(class_FIRST, agent_type_BOARDING) = round(AGENTS_OP*PART_FC);
agent_part_count(class_FIRST, agent_type_DEBOARDING) = round(AGENTS_D*PART_FC);

agent_part_count(class_SECOND, agent_type_BOARDING) = round(AGENTS_OP*(1-PART_FC));
agent_part_count(class_SECOND, agent_type_DEBOARDING) = round(AGENTS_D*(1-PART_FC));

agent_part_sum(class_FIRST, agent_type_BOARDING) = agent_part_count(class_FIRST, agent_type_BOARDING);
agent_part_sum(class_FIRST, agent_type_DEBOARDING) = agent_part_count(class_FIRST, agent_type_DEBOARDING)   + agent_part_sum(class_FIRST, agent_type_BOARDING);

agent_part_sum(class_SECOND, agent_type_BOARDING) = agent_part_count(class_SECOND, agent_type_BOARDING)    + agent_part_sum(class_FIRST, agent_type_DEBOARDING);
agent_part_sum(class_SECOND, agent_type_DEBOARDING) = agent_part_count(class_SECOND, agent_type_DEBOARDING) + agent_part_sum(class_SECOND,
```

```
    agent_type_BOARDING);

agentcount = agent_part_sum(class_SECOND, agent_type_DEBOARDING);


% Array for agents
agent = zeros(agentcount, agentCOLCOUNT);
agentspace = FORCES_COEFF(FC_agentRetraction);    % extension of an agent (m)
agentmass = 80;       % mass of an agent (kg)

%specify type of entering door (1 (subway), 2 (2nd class), 3 (1st class)
agent_mode_enter_subway = 1;
agent_mode_enter_second_class = 2;
agent_mode_enter_first_class = 3;

agent(1                                                      : agent_part_sum(class_FIRST, agent_type_BOARDING), agentMODE) = agent_mode_enter_first_class;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING)+1      : agent_part_sum(class_FIRST, agent_type_DEBOARDING), agentMODE) = agent_mode_enter_subway;

agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING)+1    : agent_part_sum(class_SECOND, agent_type_BOARDING), agentMODE) = agent_mode_enter_second_class;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING)+1     : agent_part_sum(class_SECOND, agent_type_DEBOARDING), agentMODE) = agent_mode_enter_subway;

% Specify initial state (moving, deboarding)
agent(1                                                      : agent_part_sum(class_FIRST, agent_type_BOARDING), agentSTATE) = agentSTATEmoving;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING)+1      : agent_part_sum(class_FIRST, agent_type_DEBOARDING), agentSTATE) = agentSTATEdeboarding;

agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING)+1    : agent_part_sum(class_SECOND, agent_type_BOARDING), agentSTATE) = agentSTATEmoving;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING)+1   : agent_part_sum(class_SECOND, agent_type_DEBOARDING), agentSTATE) = agentSTATEdeboarding;

%set leaving doors
agent(:, agentLDOOR) = zeros(agentcount, 1);

agent(agent_part_sum(class_FIRST, agent_type_BOARDING)+1      : agent_part_sum(class_FIRST, agent_type_DEBOARDING), agentLDOOR)  ...
```

```
    = round(linspace(10,13,agent_part_count(class_FIRST, agent_type_DEBOARDING)));
agent(agent_part_sum(class_SECOND, agent_type_BOARDING)+1     : agent_part_sum(class_SECOND, agent_type_DEBOARDING), agentLDOOR) ...
    = round(linspace(3,9,agent_part_count(class_SECOND, agent_type_DEBOARDING)));

%choice for entering door based on agentMODE
agent(:, agentCDOOR) = ones(agentcount,1);

agent(:, agentMAXV) = VELOCITY*ones(agentcount, 1) + VELOCITY_VAR * rand(agentcount, 1);
agent(:, agentPATIENT) = 0.9*ones(agentcount, 1);
agent(:, agentLAZY) = LAZINESS*ones(agentcount, 1);
agent(:, agentDMODE) = DOOR_DECISION_MODE*ones(agentcount, 1);
agentDECstepfrequency = DECISION_STEPFREQ;      % inicates the step-based freq. agent decides for best door
agent(:, agentDECTIMES) = DECISION_LIMIT*ones(agentcount, 1);
agent(:, agentGROUP) = agentGROUPnone*ones(agentcount, 1);


% -----
% DOORS
% -----
doorcount = 13; % 1 train,   3x second class waggons, 1x Bistro (1 door), 2x first class, 2 exits

% Array for doors
door = zeros(doorcount, doorCOLCOUNT);
doorrange = 0.5;
doorstrength = FORCES_COEFF(FC_doorAttraction);
doors_opening_time = DOORS_DELAY;

% Exits
door(1, doorXPOS) = 85-30;
door(2, doorXPOS) = 85+30;

door(1:2, doorYPOS) = 15;
door(1:2, doorMODE) = agent_mode_enter_subway;
door(1:2, doorSTATE) = 0;
door(1:2, doorMEANFREQ) = 5; %more people than on the train
door(1:2, doorVARFREQ) = 0.1;
door(1:2, doorACTIVITY) = doorACTIVE;

% Train
door(3, doorXPOS) = 10+0*25+1.5;
door(4, doorXPOS) = 10+0*25+23.5;   % Second class
door(5, doorXPOS) = 10+1*25+1.5;
door(6, doorXPOS) = 10+1*25+23.5;
door(7, doorXPOS) = 10+2*25+1.5;
```

```
door(8, doorXPOS) = 10+2*25+23.5;
door(9, doorXPOS) = 10+3*25+1.5;    % Bistro
door(10, doorXPOS) = 10+4*25+1.5;   % First class
door(11, doorXPOS) = 10+4*25+23.5;
door(12, doorXPOS) = 10+5*25+1.5;
door(13, doorXPOS) = 10+5*25+23.5;

door(3:13, doorYPOS) = 19.9;
door(3:9, doorMODE) = agent_mode_enter_second_class;
door(10:13, doorMODE) = agent_mode_enter_first_class;
door(3:13, doorSTATE) = doors_opening_time; % wait for some seconds until people can de/board
door(3:13, doorMEANFREQ) = 0.7;
door(3:13, doorVARFREQ) = 0.1;
door(3:13, doorACTIVITY) = doorACTIVE;

doorMODEsum = max(door(:,doorMODE));

% Sum up number of leaving agents
for idoor = 1:doorcount
    door(idoor, doorAGENT) = -sum(agent(:, agentLDOOR)==idoor);
end


% ---------
% OBSTACLES
% ---------

traincount = 1;

obstaclecount = 5; %1 train, 1 waiting area, 1 building, 2 doublesubways

% Array for obstacles
obstacle = zeros(obstaclecount, obstacleCOLCOUNT);

obstacle(:, obstacleRANGE) = FORCES_COEFF(FC_obstacleRetraction)*ones(obstaclecount, 1);
obstacle(:, obstaclePASSABLE) = zeros(obstaclecount, 1);

obstacle(:, obstacleSTART) = 0;
obstacle(:, obstacleEND) = tmax;

% train
obstacle(1, [obstacleXCENTER, obstacleYCENTER]) = [95, 22.5];
obstacle(1, [obstacleWIDTH, obstacleHEIGHT]) = [170,5];

% start area
oSTARTAREA = 2;
obstacle(oSTARTAREA, [obstacleXCENTER, obstacleYCENTER]) = [85, 16];
obstacle(oSTARTAREA, [obstacleWIDTH, obstacleHEIGHT]) = WAITING_AREA(1,:);
```

```
obstacle(oSTARTAREA, obstacleSTART) = 0;
obstacle(oSTARTAREA, obstacleEND) = AREA_DELAY;

% building
obstacle(3, [obstacleXCENTER, obstacleYCENTER]) = [100, 5];
obstacle(3, [obstacleWIDTH, obstacleHEIGHT]) = [200,10];

% subways;
obstacle(4, [obstacleWIDTH, obstacleHEIGHT]) = [20,5];
obstacle(4, [obstacleXCENTER, obstacleYCENTER]) = ...
    door(1, [doorXPOS, doorYPOS]) + [obstacle(4, obstacleWIDTH)/2 + 1, 0];
obstacle(5, [obstacleWIDTH, obstacleHEIGHT]) = [20,5];
obstacle(5, [obstacleXCENTER, obstacleYCENTER]) = ...
    door(2, [doorXPOS, doorYPOS]) - [obstacle(4, obstacleWIDTH)/2 + 1, 0];


% -----------
% TRAIN SEATS
% -----------

trainseats = zeros(traincount,6*10,2);
% Restaurant Coach and first Class already half full
trainseats(:,31:60,1) = 4*ones(traincount,3*10,1);

% set people, that are already seated
trainseats(:,1:30,:) = round(AGENTS_SEATED / 90);
trainseats(:,31:60,2) = round(AGENTS_SEATED / 90);


% ---------------
% AGENT POSITIONS
% ---------------

% Set random position for boarding agents
% (debording agents are going to be reset on their startposition in
% "simulation.m")
dspace = 0.2;   % min space between obstacle and agent (and also STARTAREA
    and agent)

for iagent=1:agentcount
    % call of script that sets random position until outside of any
    % obstacle, that is not the starting area
    set_agent_outside_of_any_obstacle
end


% - - -
% Group
% - - -
```

```
% A group consists of a couple of boarding(!) agents. They are all heading
% to the same door, which only can be chosen by their group-master.

N_groups(class_FIRST) = round(GROUPING*PART_FC*AGENTS_OP/GROUP_SIZE);
N_groups(class_SECOND) = round(GROUPING*(1-PART_FC)*AGENTS_OP/GROUP_SIZE);
groupcount = N_groups(class_FIRST)+N_groups(class_SECOND);

Size_group = zeros(groupcount, 1);
Size_group(1:N_groups(class_FIRST)) = GROUP_SIZE;
Size_group(N_groups(class_FIRST)+1:groupcount) = GROUP_SIZE;

% First-Class Groups
sagent = 1;
sgroup = 1;
for igroup = sgroup:N_groups(class_FIRST)
    init_group
end

% Second-Class Groups
sgroup = sgroup + N_groups(class_FIRST);
sagent = agent_part_sum(class_FIRST, agent_type_DEBOARDING)+1;
for igroup = sgroup:(sgroup-1) + N_groups(class_SECOND)
    init_group
end


% remaining Time between agent and door
remainingdistance = zeros(agentcount, doorcount);
remainingwalktime = zeros(agentcount, doorcount);
remainingqueuetime = zeros(agentcount, doorcount);
placeinqueue = ones(agentcount, doorcount);

% load statistic variables
init_statistics

%start simulation

simulation
```

Listing 8: init_szenario_two_trains.m

```
%simulate two parallel trains as in Zurich HB

% -------
% GENERAL
% -------

simulation_mode = simulationMODEtwotrains;

```

```
% specify scenario (SI units)
border = [0,0,200,45]; %left, bottom, width, height

% time specification
tmax = TIMEMAX;
dt = TIMESTEP;
stepcount = tmax/dt;

% ------
% AGENTS
% ------

class_FIRST = 1;
class_SECOND = 2;
class_count = 2;

agent_type_BOARDING_A = 1;
agent_type_BOARDING_B = 2;
agent_type_DEBOARDING_A = 3;
agent_type_DEBOARDING_B = 4;
agent_type_CHANGING_A_B = 5;
agent_type_CHANGING_B_A = 6;
agent_type_count = 6;

% number of agents as summed up (for later use as index ranges)

agent_part_count = zeros(class_count, agent_type_count);
agent_part_sum = zeros(class_count, agent_type_count);

agent_part_count(class_FIRST, agent_type_BOARDING_A) = round(AGENTS_OP*
    PART_FC);
agent_part_count(class_FIRST, agent_type_BOARDING_B) = round(AGENTS_OP*
    PART_FC);
agent_part_count(class_FIRST, agent_type_DEBOARDING_A) = round(AGENTS_D*
    PART_FC/2);
agent_part_count(class_FIRST, agent_type_DEBOARDING_B) = round(AGENTS_D*
    PART_FC/2);
agent_part_count(class_FIRST, agent_type_CHANGING_A_B) = round(AGENTS_D*
    PART_FC/2);
agent_part_count(class_FIRST, agent_type_CHANGING_B_A) = round(AGENTS_D*
    PART_FC/2);

agent_part_count(class_SECOND, agent_type_BOARDING_A) = round(AGENTS_OP*(1-
    PART_FC));
agent_part_count(class_SECOND, agent_type_BOARDING_B) = round(AGENTS_OP*(1-
    PART_FC));
agent_part_count(class_SECOND, agent_type_DEBOARDING_A) = round(AGENTS_D*(1-
    PART_FC)/2);
agent_part_count(class_SECOND, agent_type_DEBOARDING_B) = round(AGENTS_D*(1-
    PART_FC)/2);
```

```
agent_part_count(class_SECOND, agent_type_CHANGING_A_B) = round(AGENTS_D*(1-PART_FC)/2);
agent_part_count(class_SECOND, agent_type_CHANGING_B_A) = round(AGENTS_D*(1-PART_FC)/2);


agent_part_sum(class_FIRST, agent_type_BOARDING_A) = agent_part_count(class_FIRST, agent_type_BOARDING_A);
agent_part_sum(class_FIRST, agent_type_BOARDING_B) = agent_part_count(class_FIRST, agent_type_BOARDING_B) + agent_part_sum(class_FIRST, agent_type_BOARDING_A);
agent_part_sum(class_FIRST, agent_type_DEBOARDING_A) = agent_part_count(class_FIRST, agent_type_DEBOARDING_A) + agent_part_sum(class_FIRST, agent_type_BOARDING_B);
agent_part_sum(class_FIRST, agent_type_DEBOARDING_B) = agent_part_count(class_FIRST, agent_type_DEBOARDING_B) + agent_part_sum(class_FIRST, agent_type_DEBOARDING_A);
agent_part_sum(class_FIRST, agent_type_CHANGING_A_B) = agent_part_count(class_FIRST, agent_type_CHANGING_A_B) + agent_part_sum(class_FIRST, agent_type_DEBOARDING_B);
agent_part_sum(class_FIRST, agent_type_CHANGING_B_A) = agent_part_count(class_FIRST, agent_type_CHANGING_B_A) + agent_part_sum(class_FIRST, agent_type_CHANGING_A_B);

agent_part_sum(class_SECOND, agent_type_BOARDING_A) = agent_part_count(class_SECOND, agent_type_BOARDING_A) + agent_part_sum(class_FIRST, agent_type_CHANGING_B_A);
agent_part_sum(class_SECOND, agent_type_BOARDING_B) = agent_part_count(class_SECOND, agent_type_BOARDING_B) + agent_part_sum(class_SECOND, agent_type_BOARDING_A);
agent_part_sum(class_SECOND, agent_type_DEBOARDING_A) = agent_part_count(class_SECOND, agent_type_DEBOARDING_A) + agent_part_sum(class_SECOND, agent_type_BOARDING_B);
agent_part_sum(class_SECOND, agent_type_DEBOARDING_B) = agent_part_count(class_SECOND, agent_type_DEBOARDING_B) + agent_part_sum(class_SECOND, agent_type_DEBOARDING_A);
agent_part_sum(class_SECOND, agent_type_CHANGING_A_B) = agent_part_count(class_SECOND, agent_type_CHANGING_A_B) + agent_part_sum(class_SECOND, agent_type_DEBOARDING_B);
agent_part_sum(class_SECOND, agent_type_CHANGING_B_A) = agent_part_count(class_SECOND, agent_type_CHANGING_B_A) + agent_part_sum(class_SECOND, agent_type_CHANGING_A_B);

agentcount = agent_part_sum(class_SECOND, agent_type_CHANGING_B_A);


% Array for agents
agent = zeros(agentcount, agentCOLCOUNT);
agentspace = FORCES_COEFF(FC_agentRetraction);    % extension of an agent (m)
```

```
agentmass = 80;       % mass of an agent (kg)

%specify type of entering door (1 (subway), 2 (A 2nd class), 3 (A 1st
%class), 4 (B 2nd class), 5 (B 1st class))
agent(:, agentMODE) = zeros(agentcount, 1);
agent(1                                                  : agent_part_sum(class_FIRST, agent_type_BOARDING_A), agentMODE) = 3;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_A)+1    : agent_part_sum(class_FIRST, agent_type_BOARDING_B), agentMODE) = 5;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_B)+1    : agent_part_sum(class_FIRST, agent_type_DEBOARDING_A), agentMODE) = 1;
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_A)+1  : agent_part_sum(class_FIRST, agent_type_DEBOARDING_B), agentMODE) = 1;
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_B)+1  : agent_part_sum(class_FIRST, agent_type_CHANGING_A_B), agentMODE) = 5;
agent(agent_part_sum(class_FIRST, agent_type_CHANGING_A_B)+1  : agent_part_sum(class_FIRST, agent_type_CHANGING_B_A), agentMODE) = 3;

agent(agent_part_sum(class_FIRST, agent_type_CHANGING_B_A)+1  : agent_part_sum(class_SECOND, agent_type_BOARDING_A), agentMODE) = 2;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_A)+1   : agent_part_sum(class_SECOND, agent_type_BOARDING_B), agentMODE) = 4;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_B)+1   : agent_part_sum(class_SECOND, agent_type_DEBOARDING_A), agentMODE) = 1;
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_A)+1 : agent_part_sum(class_SECOND, agent_type_DEBOARDING_B), agentMODE) = 1;
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_B)+1 : agent_part_sum(class_SECOND, agent_type_CHANGING_A_B), agentMODE) = 4;
agent(agent_part_sum(class_SECOND, agent_type_CHANGING_A_B)+1 : agent_part_sum(class_SECOND, agent_type_CHANGING_B_A), agentMODE) = 2;


% Specify initial state (moving, deboarding)
agent(1                                                  : agent_part_sum(class_FIRST, agent_type_BOARDING_A), agentSTATE) = agentSTATEmoving;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_A)+1    : agent_part_sum(class_FIRST, agent_type_BOARDING_B), agentSTATE) = agentSTATEmoving;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_B)+1    : agent_part_sum(class_FIRST, agent_type_DEBOARDING_A), agentSTATE) = agentSTATEdeboarding;
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_A)+1  : agent_part_sum(class_FIRST, agent_type_DEBOARDING_B), agentSTATE) = agentSTATEdeboarding;
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_B)+1  : agent_part_sum(class_FIRST, agent_type_CHANGING_A_B), agentSTATE) = agentSTATEdeboarding;
agent(agent_part_sum(class_FIRST, agent_type_CHANGING_A_B)+1  : agent_part_sum(class_FIRST, agent_type_CHANGING_B_A), agentSTATE) =
```

```
    agentSTATEdeboarding;

agent(agent_part_sum(class_FIRST, agent_type_CHANGING_B_A)+1  :
    agent_part_sum(class_SECOND, agent_type_BOARDING_A), agentSTATE) =
    agentSTATEmoving;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_A)+1   :
    agent_part_sum(class_SECOND, agent_type_BOARDING_B), agentSTATE) =
    agentSTATEmoving;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_B)+1   :
    agent_part_sum(class_SECOND, agent_type_DEBOARDING_A), agentSTATE) =
    agentSTATEdeboarding;
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_A)+1 :
    agent_part_sum(class_SECOND, agent_type_DEBOARDING_B), agentSTATE) =
    agentSTATEdeboarding;
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_B)+1 :
    agent_part_sum(class_SECOND, agent_type_CHANGING_A_B), agentSTATE) =
    agentSTATEdeboarding;
agent(agent_part_sum(class_SECOND, agent_type_CHANGING_A_B)+1 :
    agent_part_sum(class_SECOND, agent_type_CHANGING_B_A), agentSTATE) =
    agentSTATEdeboarding;


%set leaving doors
agent(:, agentLDOOR) = zeros(agentcount, 1);

agent(1                                                       :
    agent_part_sum(class_FIRST, agent_type_BOARDING_A), agentLDOOR)  ...
    = 0;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_A)+1    :
    agent_part_sum(class_FIRST, agent_type_BOARDING_B), agentLDOOR)   ...
    = 0;
agent(agent_part_sum(class_FIRST, agent_type_BOARDING_B)+1    :
    agent_part_sum(class_FIRST, agent_type_DEBOARDING_A), agentLDOOR) ...
    = round(linspace(11,14,agent_part_count(class_FIRST,
        agent_type_DEBOARDING_A)));
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_A)+1  :
    agent_part_sum(class_FIRST, agent_type_DEBOARDING_B), agentLDOOR) ...
    = round(linspace(22,25,agent_part_count(class_FIRST,
        agent_type_DEBOARDING_B)));
agent(agent_part_sum(class_FIRST, agent_type_DEBOARDING_B)+1  :
    agent_part_sum(class_FIRST, agent_type_CHANGING_A_B), agentLDOOR) ...
    = round(linspace(11,14,agent_part_count(class_FIRST,
        agent_type_CHANGING_A_B)));
agent(agent_part_sum(class_FIRST, agent_type_CHANGING_A_B)+1  :
    agent_part_sum(class_FIRST, agent_type_CHANGING_B_A), agentLDOOR) ...
    = round(linspace(22,25,agent_part_count(class_FIRST,
        agent_type_CHANGING_B_A)));

agent(agent_part_sum(class_FIRST, agent_type_CHANGING_B_A)+1  :
    agent_part_sum(class_SECOND, agent_type_BOARDING_A), agentLDOOR)  ...
```

```
    = 0;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_A)+1   : agent_part_sum(class_SECOND, agent_type_BOARDING_B), agentLDOOR)  ...
    = 0;
agent(agent_part_sum(class_SECOND, agent_type_BOARDING_B)+1   : agent_part_sum(class_SECOND, agent_type_DEBOARDING_A), agentLDOOR) ...
    = round(linspace(4,10,agent_part_count(class_SECOND, agent_type_DEBOARDING_A)));
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_A)+1 : agent_part_sum(class_SECOND, agent_type_DEBOARDING_B), agentLDOOR) ...
    = round(linspace(15,21,agent_part_count(class_SECOND, agent_type_DEBOARDING_B)));
agent(agent_part_sum(class_SECOND, agent_type_DEBOARDING_B)+1 : agent_part_sum(class_SECOND, agent_type_CHANGING_A_B), agentLDOOR) ...
    = round(linspace(4,10,agent_part_count(class_SECOND, agent_type_CHANGING_A_B)));
agent(agent_part_sum(class_SECOND, agent_type_CHANGING_A_B)+1 : agent_part_sum(class_SECOND, agent_type_CHANGING_B_A), agentLDOOR) ...
    = round(linspace(15,21,agent_part_count(class_SECOND, agent_type_CHANGING_B_A)));

%choice for entering door based on agentMODE
agent(:, agentCDOOR) = ones(agentcount,1);

agent(:, agentMAXV) = VELOCITY*ones(agentcount, 1) + VELOCITY_VAR * rand(agentcount, 1);
agent(:, agentPATIENT) = 0.9*ones(agentcount, 1);
agent(:, agentLAZY) = LAZINESS*ones(agentcount, 1);
agent(:, agentDMODE) = DOOR_DECISION_MODE*ones(agentcount, 1);
agentDECstepfrequency = DECISION_STEPFREQ;  % inicates the step-based freq. agent decides for best door
agent(:, agentDECTIMES) = DECISION_LIMIT*ones(agentcount, 1);
agent(:, agentGROUP) = agentGROUPnone*ones(agentcount, 1);




% -----
% DOORS
% -----
doorcount = 25; % 2 trains, each 3x second class, 1x Bistro (1 door), 2x first class, 3 exits

% Array for doors
door = zeros(doorcount, doorCOLCOUNT);
doorrange = 0.5;
doorstrength = FORCES_COEFF(FC_doorAttraction);
doors_opening_time = DOORS_DELAY;


```

```
% Exits
door(1, doorXPOS) = 49.9;
door(2, doorXPOS) = 155.1;
door(3, doorXPOS) = 195;

door(1:3, doorYPOS) = 20;
agent_mode_enter_subway = 1;
door(1:3, doorMODE) = agent_mode_enter_subway;
door(1:3, doorSTATE) = 0;
door(1:3, doorMEANFREQ) = 10; %more people than on the train
door(1:3, doorVARFREQ) = 0.1;
door(1:3, doorACTIVITY) = doorACTIVE;

% First Train
door(4, doorXPOS) = 15+0*25+1.5;
door(5, doorXPOS) = 15+0*25+23.5;   % Second class
door(6, doorXPOS) = 15+1*25+1.5;
door(7, doorXPOS) = 15+1*25+23.5;
door(8, doorXPOS) = 15+2*25+1.5;
door(9, doorXPOS) = 15+2*25+23.5;
door(10, doorXPOS) = 15+3*25+1.5;    % Bistro
door(11, doorXPOS) = 15+4*25+1.5;   % First class
door(12, doorXPOS) = 15+4*25+23.5;
door(13, doorXPOS) = 15+5*25+1.5;
door(14, doorXPOS) = 15+5*25+23.5;

door(4:14, doorYPOS) = 24.9;
door(4:10, doorMODE) = 2;
door(11:14, doorMODE) = 3;
door(4:14, doorSTATE) = doors_opening_time; % wait for some seconds until
    people can de/board
door(4:14, doorMEANFREQ) = 0.7;
door(4:14, doorVARFREQ) = 0.1;
door(4:14, doorACTIVITY) = doorACTIVE;

% Second Train
door(15, doorXPOS) = 15+0*25+1.5;
door(16, doorXPOS) = 15+0*25+23.5;   % Second class
door(17, doorXPOS) = 15+1*25+1.5;
door(18, doorXPOS) = 15+1*25+23.5;
door(19, doorXPOS) = 15+2*25+1.5;
door(20, doorXPOS) = 15+2*25+23.5;
door(21, doorXPOS) = 15+3*25+1.5;    % Bistro
door(22, doorXPOS) = 15+4*25+1.5;   % First class
door(23, doorXPOS) = 15+4*25+23.5;
door(24, doorXPOS) = 15+5*25+1.5;
door(25, doorXPOS) = 15+5*25+23.5;

door(15:25, doorYPOS) = 15.1;
door(15:21, doorMODE) = 4;
```

```
door(22:25, doorMODE) = 5;
door(15:25, doorSTATE) = doors_opening_time; % wait for some seconds until people can de/board
door(15:25, doorMEANFREQ) = 0.7;
door(15:25, doorVARFREQ) = 0.1;
door(15:25, doorACTIVITY) = doorACTIVE;

doorMODEsum = max(door(:,doorMODE));

% Sum up number of leaving agents
for idoor = 1:doorcount
    door(idoor, doorAGENT) = -sum(agent(:, agentLDOOR)==idoor);
end


% ---------
% OBSTACLES
% ---------

traincount = 2;

obstaclecount = 9; %2 trains, 2 triple subway entrances, 1 start area

% Array for obstacles
obstacle = zeros(obstaclecount, obstacleCOLCOUNT);

obstacle(:, obstacleSTART) = 0;
obstacle(:, obstacleEND) = tmax;

obstacle(:, obstacleRANGE) = FORCES_COEFF(FC_obstacleRetraction)*ones(obstaclecount, 1);
obstacle(:, obstaclePASSABLE) = zeros(obstaclecount, 1);

% trains
obstacle(1, [obstacleXCENTER, obstacleYCENTER]) = [100, 27.5];
obstacle(1, [obstacleWIDTH, obstacleHEIGHT]) = [170,5];
obstacle(2, [obstacleXCENTER, obstacleYCENTER]) = [100, 12.5];
obstacle(2, [obstacleWIDTH, obstacleHEIGHT]) = [170,5];

% subway entrances
obstacle(3, [obstacleXCENTER, obstacleYCENTER]) = [55, 20];
obstacle(3, [obstacleWIDTH, obstacleHEIGHT]) = [9.5, 5];
obstacle(4, [obstacleXCENTER, obstacleYCENTER]) = [150, 20];
obstacle(4, [obstacleWIDTH, obstacleHEIGHT]) = [9.5, 5];

% further small obstacles
obstacle(5, [obstacleXCENTER, obstacleYCENTER]) = [80, 20];
obstacle(5, [obstacleWIDTH, obstacleHEIGHT]) = [3, 1.5];
obstacle(6, [obstacleXCENTER, obstacleYCENTER]) = [105, 20];
obstacle(6, [obstacleWIDTH, obstacleHEIGHT]) = [3, 1.5];
```

```
obstacle(7, [obstacleXCENTER, obstacleYCENTER]) = [130, 20];
obstacle(7, [obstacleWIDTH, obstacleHEIGHT]) = [3, 1.5];
obstacle(8, [obstacleXCENTER, obstacleYCENTER]) = [30, 20];
obstacle(8, [obstacleWIDTH, obstacleHEIGHT]) = [3, 1.5];


%start area
oSTARTAREA = 9;
obstacle(oSTARTAREA, [obstacleXCENTER, obstacleYCENTER]) = [90, 20];
obstacle(oSTARTAREA, [obstacleWIDTH, obstacleHEIGHT]) = WAITING_AREA(2,:);

obstacle(oSTARTAREA, obstacleSTART) = 0;
obstacle(oSTARTAREA, obstacleEND) = AREA_DELAY;


% -----------
% TRAIN SEATS
% -----------

trainseats = zeros(traincount,6*10,2);
% Restaurant Coach and first Class already half full
trainseats(:,31:60,1) = 4*ones(traincount,3*10,1);

% set people, that are already seated
trainseats(:,1:30,:) = round(AGENTS_SEATED / 90);
trainseats(:,31:60,2) = round(AGENTS_SEATED / 90);

% ---------------
% AGENT POSITIONS
% ---------------

% Set random position for boarding agents
% (debording agents are going to be reset on their startposition in
% "simulation.m")
dspace = 0.2;   % min space between obstacle and agent

for iagent=1:agentcount
    % call of script that sets random position until outside of any
    % obstacle, that is not the starting area
    set_agent_outside_of_any_obstacle
end


% - - -
% Group
% - - -
% A group consists of a couple of boarding(!) agents. They are all heading
% to the same door, which only can be chosen by their group-master.

N_groups(class_FIRST) = round(GROUPING*PART_FC*AGENTS_OP*2/GROUP_SIZE);
```

```
N_groups(class_SECOND) = round(GROUPING*(1-PART_FC)*AGENTS_OP*2/GROUP_SIZE);
groupcount = N_groups(class_FIRST)+N_groups(class_SECOND);

Size_group = zeros(groupcount, 1);
Size_group(1:N_groups(class_FIRST)) = GROUP_SIZE;
Size_group(N_groups(class_FIRST)+1:groupcount) = GROUP_SIZE;

% First-Class Groups
sagent = 1;
sgroup = 1;
for igroup = sgroup:N_groups(class_FIRST)
    init_group
end
sgroup = sgroup + N_groups(class_FIRST);

% Second-Class Groups
sagent = agent_part_sum(class_FIRST, agent_type_CHANGING_B_A)+1;
for igroup = sgroup:(sgroup-1) + N_groups(class_SECOND)
    init_group
end


% ----------


% remaining Time between agent and door
remainingdistance = zeros(agentcount, doorcount);
remainingwalktime = zeros(agentcount, doorcount);
remainingqueuetime = zeros(agentcount, doorcount);
placeinqueue = ones(agentcount, doorcount);



% load statistic variables
init_statistics

%start simulation
simulation
```

Listing 9: init_group.m

```
% loop through all members of a group and init agent-values for XPOS,
% YPOS,GROUP and DECTIMES

for iagent = sagent:(sagent-1)+Size_group(igroup)
    % all agents of the group get grouped around a circle, that gets scaled
    % according to the number of people inside the group (10 people -> in
    % circle with diameter of 2 meters
    radius = Size_group(igroup)/10;
```

```
    % only master of group can decide
    if iagent ≠ sagent
        agent(iagent, agentDECTIMES) = agentDECTIMESnone;
    else
        dspace = 1.5 * radius;
        set_agent_outside_of_any_obstacle;
        group_CENTER = agent(sagent, [agentXPOS, agentYPOS]);
    end
    agent(iagent, agentGROUP) = igroup;
    phi = 2*pi()*iagent/Size_group(igroup); % Angle to set group in a cyrcle
    agent(iagent, [agentXPOS, agentYPOS]) = group_CENTER + radius*[cos(phi), sin(phi)];
end
sagent = sagent + Size_group(igroup);
```

Listing 10: init_statistics.m

```
% initialistation of saved statistic data
%--------
% Arrays for saving data
%--------

% # agents heading to spec. door
stat_approaching_to_door = zeros(stepcount,doorcount);
% # agents on platform
stat_moving_agents = zeros(stepcount, 1);
% mean distance of all moving agents to their door
stat_distance_to_go = zeros(stepcount, 1);
% # agents boarded on spec. door
stat_boarded_per_door = zeros(stepcount, doorcount);
% start end end time of moving
stat_moving_time = zeros(agentcount, 2);
    stat_movSTART = 1;
    stat_movEND = 2;
% # agents waiting in queue
stat_waiting_agents = zeros(stepcount, 2);
% sum. waiting time
stat_sum_waiting = zeros(agentcount, 1);
% walking distance per agent
stat_sum_distance = zeros(agentcount, 1);
% min. distance between leavingdoor and chosendoor
stat_min_distance = zeros(agentcount, 1);
% startpostion (is needed to calculate stat_min_distance
stat_start_position = zeros(agentcount, 2); % set in simulation.m
% sum of redecicions
stat_sum_decision = zeros(agentcount, 1);
```

Listing 11: set_agent_outside_of_any_obstacle.m

```
 % move agent randomly inside the StartArea until a minimal distance of ...
 % 'dspace' to every other obstacle and the border of the startarea is garanteed

 % Consider also, that a person will not start from a position too far from
 % its desteny, i.e. that a first-class passenger is rather startig from a
 % point not so far from the nearest first-class entrance.

%agent(iagent, agentXPOS) = obstacle(oSTARTAREA, obstacleXCENTER) - obstacle(oSTARTAREA, obstacleWIDTH)/2 + ...
%     (obstacle(oSTARTAREA, obstacleWIDTH)-2*dspace)*rand(1,1)+dspace;
%agent(iagent, agentYPOS) = obstacle(oSTARTAREA, obstacleYCENTER) - obstacle(oSTARTAREA, obstacleHEIGHT)/2 + ...
%     (obstacle(oSTARTAREA, obstacleHEIGHT)-2*dspace)*rand(1,1)+dspace;

is_agentPOSok = 0;
while is_agentPOSok == 0
    agent(iagent, agentXPOS) = obstacle(oSTARTAREA, obstacleXCENTER) - obstacle(oSTARTAREA, obstacleWIDTH)/2 + ...
        (obstacle(oSTARTAREA, obstacleWIDTH)-2*dspace)*rand(1,1)+dspace;
    agent(iagent, agentYPOS) = obstacle(oSTARTAREA, obstacleYCENTER) - obstacle(oSTARTAREA, obstacleHEIGHT)/2 + ...
        (obstacle(oSTARTAREA, obstacleHEIGHT)-2*dspace)*rand(1,1)+dspace;

    is_agentPOSok = 1;  % assuming pos is ok

    % Check if agent ist not too far away from its nearest possible door
    for idoor = 1:doorcount
    remainingdistance(iagent, idoor) = norm(agent(iagent, [agentXPOS, agentYPOS]) - door(idoor, [doorXPOS, doorYPOS]));
  end
    min_remainingdistance_MODE = min(remainingdistance(iagent, door(:, doorMODE) == agent(iagent, agentMODE)));
    if min_remainingdistance_MODE > border(3)/4      % more than quarter scene?
        is_agentPOSok = 0;

    end

    % now check for obstacles
    for iobstacle = 1:obstaclecount
        if iobstacle ≠ oSTARTAREA   % dont check start area
            if agent(iagent, agentXPOS) > (obstacle(iobstacle, obstacleXCENTER) - obstacle(iobstacle, obstacleWIDTH)/2 - dspace)
                if agent(iagent, agentXPOS) < (obstacle(iobstacle, obstacleXCENTER) + obstacle(iobstacle, obstacleWIDTH)/2 + dspace)
```

```
                        if agent(iagent, agentYPOS) > (obstacle(iobstacle,
                            obstacleYCENTER) - obstacle(iobstacle,
                            obstacleHEIGHT)/2 - dspace)
                            if agent(iagent, agentYPOS) < (obstacle(iobstacle,
                                obstacleYCENTER) + obstacle(iobstacle,
                                obstacleHEIGHT)/2 + dspace)
                                is_agentPOSok = 0;    % Position was NOT ok
                                %agent(iagent, agentXPOS) = obstacle(oSTARTAREA,
                                     obstacleXCENTER) - obstacle(oSTARTAREA,
                                    obstacleWIDTH)/2 + ...
                                %    (obstacle(oSTARTAREA, obstacleWIDTH)-2*
                                    dspace)*rand(1,1)+dspace;
                                %agent(iagent, agentYPOS) = obstacle(oSTARTAREA,
                                     obstacleYCENTER) - obstacle(oSTARTAREA,
                                    obstacleHEIGHT)/2 + ...
                                %    (obstacle(oSTARTAREA, obstacleHEIGHT)-2*
                                    dspace)*rand(1,1)+dspace;
                            end
                        end
                    end
                end
            end
        end
    end
```

8.2.3 Simulation

Listing 12: simulation.m

```
% simulation

% set deboarding agents on the doorpoint
for iagent = 1:agentcount
    if agent(iagent, agentSTATE) == agentSTATEdeboarding
        agent(iagent, [agentXPOS, agentYPOS]) = door(agent(iagent,
            agentLDOOR), [doorXPOS, doorYPOS]);
    end
    stat_start_position(iagent, :) = agent(iagent, [agentXPOS, agentYPOS]);
end

%timestep iteration
for step = 1:stepcount
    t = step*dt;

    if t <= DOORS_DELAY
        train_entrance;
    end
```

```
    % random order for agents
    order = randperm(agentcount);
    doordecision_frequency;
    % decrement door state ("blocking" time)
    door(:,doorSTATE) = door(:,doorSTATE) - dt*ones(doorcount,1);
    calculate_distances;

    for i = 1:agentcount
        iagent = order(i);
        % people update in random order (board, deboard, status change)
        agent_update
        if agent(iagent, agentSTATE) == agentSTATEmoving
            door_decision;
            calculate_forces;
        end
    end
    % move agents simultaneously
    move_agents

    % draw
    paint
    video_capture
    pause(0.02)
    save_data
    data_export

end
```

Listing 13: train_entrance.m

```
% let train drive into its final position before the doors open

if step == 1
    % set initial position of train and doors
    obstacle(1:traincount, obstacleXCENTER) = obstacle(1:traincount,
        obstacleXCENTER) - DOORS_DELAY * trainVELOCITY;
    door(door(:,doorMODE) ≠ agent_mode_enter_subway, doorXPOS) = door(door
        (:,doorMODE) ≠ agent_mode_enter_subway, doorXPOS) - DOORS_DELAY *
        trainVELOCITY;
end
obstacle(1:traincount, obstacleXCENTER) = obstacle(1:traincount,
    obstacleXCENTER) + dt * trainVELOCITY;
door(door(:,doorMODE) ≠ agent_mode_enter_subway, doorXPOS) = door(door(:,
    doorMODE) ≠ agent_mode_enter_subway, doorXPOS) + dt * trainVELOCITY;
```

Listing 14: doordecision_frequency.m

```
% The frequency of redecision is checked
if agentDECstepfrequency ≤ 1
    if mod(step, 1/agentDECstepfrequency) < mod((step-1), 1/
        agentDECstepfrequency);
        Ndec = 0;
    else
        Ndec = 1;
    end
else
    if mod(step, agentDECstepfrequency) < mod((step-1),
        agentDECstepfrequency);
        Ndec = floor(agentDECstepfrequency) + 0;
    else
        Ndec = floor(agentDECstepfrequency) + 1;
    end
end
```

Listing 15: calculate_distances.m

```
% calculate the all pairs of distances between any person and any door

for iagent = 1:agentcount
  for idoor = 1:doorcount
    remainingdistance(iagent, idoor) = norm(agent(iagent, [agentXPOS,
        agentYPOS]) - door(idoor, [doorXPOS, doorYPOS]));
  end
end
```

Listing 16: agent_update.m

```
% agent's state update (board, deboard: status change)

% check if agent can debord
if agent(iagent, agentSTATE) == agentSTATEdeboarding;
    ldoor = agent(iagent, agentLDOOR);
    if door(ldoor, doorSTATE) ≤ 0;
        % let agent debord
        agent(iagent, agentSTATE) = agentSTATEmoving;
        stat_moving_time(iagent, stat_movSTART) = t;
        % block door for a moment
        door(ldoor, doorSTATE) = 1/(door(ldoor, doorMEANFREQ)+...
            randn(1)*door(ldoor, doorVARFREQ));
        % decrease counter of people left in door
        door(ldoor, doorAGENT) = door(ldoor, doorAGENT) - doorAGENTdebord;
    end
end

```

```
% check if agent can bord
if agent(iagent, agentSTATE) == agentSTATEmoving;
    cdoor = agent(iagent, agentCDOOR);
    if remainingdistance(iagent, cdoor) < doorrange & ...
            door(cdoor, doorSTATE) < 0
        % check whether there is a free seat in this coach
        agent_seat_search;
        if free_seat_found == 1
            % let agent board
            agent(iagent, agentSTATE) = agentSTATEboarded;
            stat_moving_time(iagent, stat_movEND) = t;
            stat_sum_distance(iagent) = stat_sum_distance(iagent) + norm(
                agent(iagent, [agentXPOS, agentYPOS]) - door(agent(iagent,
                agentCDOOR), [doorXPOS, doorYPOS]));
            % block door for a moment
            door(cdoor, doorSTATE) = 1/(door(cdoor, doorMEANFREQ)+...
                randn(1)*door(cdoor, doorVARFREQ));
            % increase counter of people borded
            door(cdoor, doorAGENT) = door(cdoor, doorAGENT) + doorAGENTbord;
        else
            % lock the door
            door(cdoor, doorACTIVITY) = doorINACTIVE;
            % give the agent a chance to possibly redecide for a new door
            if agent(iagent, agentDECTIMES) == agentDECTIMESnone
                agent(iagent, agentDECTIMES) = 1;
            end
        end
    end
end
```

Listing 17: agent_seat_search.m

```
% tries to find a free seat for iagent starting from its current chosen
% door

% important note: this file is specifically designed for the two train
% station layouts and needs to be updated if any door-configuration gets
% changed
free_seat_found = 0;
% making szenario-specific-separations
if simulation_mode == simulationMODEonetrain
    if cdoor ≤ 2
        free_seat_found = 1;
        return;
    end
    % each door of second class coach and bistro wagon
    ctrain = 1;
    if (cdoor ≤ 9)
        ccoach = (cdoor - mod(cdoor+1,2) - 1) / 2;
```

```
    % first class coaches
    else
        ccoach = (cdoor - mod(cdoor,2)) / 2;
    end
end

if simulation_mode == simulationMODEtwotrains
    if cdoor ≤ 3
        free_seat_found = 1;
        return;
    end
    % each door of second class coach and bistro wagon
    if (cdoor ≤ 10)
        ctrain = 1;
        ccoach = (cdoor - mod(cdoor,2) - 2) / 2;
    % first class coaches
    elseif (cdoor ≤ 14)
        ctrain = 1;
        ccoach = (cdoor - mod(cdoor+1,2) -1) / 2;

    elseif (cdoor ≤ 21)
        ctrain = 2;
        ccoach = (cdoor - mod(cdoor+1,2) -1) / 2 - 6;
    else
        ctrain = 2;
        ccoach = (cdoor - mod(cdoor,2)) / 2 - 6;
    end

end
coach_seat_search;
```

Listing 18: coach_seat_search.m

```
start_compartment = 10*(ccoach-1)+1;
end_compartment = 10*ccoach;

for icompartment = start_compartment : end_compartment
    for iside = 1 : 2
        if trainseats(ctrain, icompartment, iside) < 4
          trainseats(ctrain, icompartment, iside) = trainseats(ctrain,
              icompartment, iside) + 1;
          free_seat_found = 1;
          return;
        end
    end

end
```

Listing 19: door_decision.m

```
% choose best door for the current moving agents

for idec = 1:Ndec
    if agent(iagent,agentSTATE) == agentSTATEmoving
        if agent(iagent, agentDECTIMES) ≠ agentDECTIMESnone
             placeinqueue(iagent, :) = ones(1, doorcount);
             for kagent = 1:agentcount
                 if agent(kagent,agentSTATE) == agentSTATEmoving
                     kdoor = agent(kagent,agentCDOOR);
                     if(remainingdistance(kagent,kdoor) < remainingdistance(iagent,kdoor))
                         % agents in the same group are not considered
                         if ((agent(kagent, agentGROUP) ≠ agent(iagent, agentGROUP)) || (agent(iagent, agentGROUP) == agentGROUPnone))
                             placeinqueue(iagent, kdoor) = placeinqueue(iagent, kdoor) + 1;
                         end
                     end
                 end
             end

             % calculate expected remaining time
             for idoor = 1:doorcount
                 if ((agent(iagent, agentMODE) == door(idoor, doorMODE)) && (agent(iagent, agentLDOOR) ≠ idoor) && (door(idoor, doorACTIVITY) == doorACTIVE))
                     remainingwalktime(iagent, idoor) = remainingdistance(iagent, idoor) / agent(iagent, agentMAXV);
                     remainingqueuetime(iagent, idoor) = placeinqueue(iagent, idoor)/door(idoor, doorMEANFREQ);
                 else
                     remainingwalktime(iagent, idoor) = 9999; %%%%%%% NOT PROPER
                     remainingqueuetime(iagent, idoor) = 9999999; %%%%%% NEITHER
                 end
             end

             % prefer current decision with the patient factor
             remainingwalktime(iagent, agent(iagent, agentCDOOR)) = remainingwalktime(iagent, agent(iagent, agentCDOOR)) * (agent(iagent, agentPATIENT));
             remainingqueuetime(iagent, agent(iagent, agentCDOOR)) = (remainingqueuetime(iagent, agent(iagent, agentCDOOR))) * (agent(iagent, agentPATIENT));

             % Choose appropriate door (depending choosing-mode)
             switch agent(iagent, agentDMODE)
```

```
                case agentDMODEsum_lazy
                    [remainingtime(iagent), newdoor] = min(remainingwalktime(iagent, :)*agent(iagent, agentLAZY) + remainingqueuetime(iagent, :)*(1-agent(iagent, agentLAZY)));
                case agentDMODEsum
                    [remainingtime(iagent), newdoor] = min(remainingwalktime(iagent, :) + remainingqueuetime(iagent, :));
                case agentDMODEwalk
                    [remainingtime(iagent), newdoor] = min(remainingwalktime(iagent, :));
                case agentDMODEqueue
                    [remainingtime(iagent), newdoor] = min(remainingqueuetime(iagent, :));
                case agentDMODEwait
                    [remainingtime(iagent), newdoor] = min(abs(remainingqueuetime(iagent, :) - remainingwalktime(iagent, :)));
                case agentDMODErandom
                    [remainingtime(iagent), newdoor] = max(rand(doorcount,1) .* (remainingwalktime(iagent, :)' < 9000*ones(doorcount,1)));
            end


            if agent(iagent, agentCDOOR) ≠ newdoor
                % decrease number of possible redicisiontimes left
                if agent(iagent, agentDECTIMES) ≠ agentDECTIMESinfinite;
                    agent(iagent, agentDECTIMES) = agent(iagent, agentDECTIMES) - 1;
                end
                % All members of a group to the same door
                igroup = agent(iagent, agentGROUP);
                if (igroup ≠ agentGROUPnone)
                    agent(agent(:, agentGROUP)==igroup, agentCDOOR) = newdoor;
                    stat_sum_decision(agent(:, agentGROUP)==igroup) = stat_sum_decision(agent(:, agentGROUP)==igroup) + 1;
                else
                    agent(iagent, agentCDOOR) = newdoor;
                    stat_sum_decision(iagent) = stat_sum_decision(iagent) + 1;
                end
            end
        end
    end
end
```

Listing 20: calculate_forces.m

```
% calculate the resulting force acting on "iagent"

% temporary sum of forces
agentforce = [0,0];
doorforce = [0,0];
obstacleforce = [0,0];
% people attraction and retraction
for kagent = 1:agentcount
    if (kagent ≠ iagent & agent(kagent, agentSTATE) == agentSTATEmoving)
        vec_agentdist = agent(kagent, [agentXPOS, agentYPOS]) - ...
            agent(iagent, [agentXPOS, agentYPOS]);
        norm_agentdist = norm(vec_agentdist);

        % group people keep more together
        if (agent(iagent, agentGROUP) == agent(kagent, agentGROUP) & ...
                agent(iagent, agentGROUP) > 0)
            strength = FORCES_COEFF(FC_agentAttractionGroup);
        else
            strength = FORCES_COEFF(FC_agentAttraction);
        end

        agentforce = agentforce - strength/(norm_agentdist)^3 ...
            * vec_agentdist/norm_agentdist;
        % agent attraction
        agentforce = agentforce + (strength/agentspace)/(norm_agentdist^2) ...
            * vec_agentdist/norm_agentdist;
    end
end

% Door attraction and retraction
vec_doordist = door(agent(iagent, agentCDOOR), [doorXPOS, doorYPOS]) - ...
    agent(iagent, [agentXPOS, agentYPOS]);
norm_doordist = norm(vec_doordist);

% door attraction
doorforce = vec_doordist/norm_doordist;
% door retraction while occupied
if door(agent(iagent, agentCDOOR), doorSTATE) > 0
    doorforce = doorforce - doorrange/norm_doordist * ...
        vec_doordist/norm_doordist;
end
doorforce = doorstrength * doorforce;


% Obstacle retraction
for kobstacle = 1:obstaclecount
iforce = [0,0];

```

```
    if obstacle(kobstacle,obstacleSTART) ≤ t && obstacle(kobstacle,obstacleEND) ≥ t
        agentx = agent(iagent, agentXPOS);
        agenty = agent(iagent, agentYPOS);
        obstaclex = obstacle(kobstacle, obstacleXCENTER);
        obstacley = obstacle(kobstacle, obstacleYCENTER);
        obstaclew = obstacle(kobstacle, obstacleWIDTH);
        obstacleh = obstacle(kobstacle, obstacleHEIGHT);
        %if inside obstacle
        if (abs(agentx - obstaclex) < obstaclew/2) && (abs(agenty - obstacley) < obstacleh/2)

            mindistance = max([obstaclew, obstacleh]);
            closestwall = 0;
            for idir = 1:4
                % orthogonal distance to the closest wall
                distance = abs(xdir(idir)) * abs(obstaclex+xdir(idir)*obstaclew/2 - agentx) + abs(ydir(idir)) * abs(obstacley+ydir(idir)*obstacleh/2 - agenty);

                if mindistance > distance
                    mindistance = distance;
                    closestwall = idir;
                end
            end

            iforce(1) = (-xdir(closestwall)*obstacle(kobstacle,obstacleRANGE))/mindistance;
            iforce(2) = (-ydir(closestwall)*obstacle(kobstacle,obstacleRANGE))/mindistance;

            obstacleforce = obstacleforce + iforce;

        %outside of the obstacle
        else
            x∆ = 0; y∆ = 0;
            if agentx > obstaclex + obstaclew/2
                x∆ = 1;
            end
            if agentx < obstaclex - obstaclew/2
                x∆ = -1;
            end
            if agenty > obstacley + obstacleh/2
                y∆ = 1;
            end
            if agenty < obstacley - obstacleh/2
                y∆ = -1;
            end

            edge = [0,0];
```

```

            %nearest point is an edge
            if (xΔ ≠ 0) && (yΔ ≠ 0)
                edge(1) = obstaclex + xΔ*obstaclew/2;
                edge(2) = obstacley + yΔ*obstacleh/2;

            %neares point is a side
            else
                if xΔ ≠ 0
                    edge(1) = obstaclex + xΔ*obstaclew/2;
                    edge(2) = agenty;
                else
                    edge(1) = agentx;
                    edge(2) = obstacley + yΔ*obstacleh/2;
                end
            end

            %calculate distance and resulting force
            vec_diff = agent(iagent, [agentXPOS, agentYPOS]) - edge;
            iforce = vec_diff/norm(vec_diff) * obstacle(kobstacle, obstacleRANGE)/norm(vec_diff);
            obstacleforce = obstacleforce + iforce;

        end
    end
end

% Correction Force (helps to avoid obstacles)
if (abs(doorforce(1) + obstacleforce(1)) < 1000  && remainingdistance(iagent, agent(iagent, agentCDOOR)) > 5*doorrange) ...
    % "cross product" between obstacleforce and doorforce
    cross_OxD = obstacleforce(1)*doorforce(2)-obstacleforce(2)*doorforce(1);
    orth_O = obstacleforce*[0, -1; 1, 0];
    %orth_O = [obstacleforce(2), -obstacleforce(1)];
    corrforce = sign(cross_OxD) * 999999999*orth_O;  %very high
    % For Agents entering Train inverse way round
    if agent(iagent, agentMODE) ≠ agent_mode_enter_subway;
        corrforce = -corrforce;
    end
else
    corrforce = [0, 0];
end

agent(iagent, [agentXFORCE, agentYFORCE]) = doorforce + agentforce + obstacleforce + corrforce;
```

Listing 21: move_agents.m

```
% move agents: a -> dv -> v -> ds -> s
```

```

for iagent = 1:agentcount
    dv = agent(iagent, [agentXFORCE, agentYFORCE])/agentmass;

    agent(iagent, [agentXVEL, agentYVEL]) = ...
        agent(iagent, [agentXVEL, agentYVEL]) + dv*dt;

    if norm(agent(iagent, [agentXVEL, agentYVEL])) > agent(iagent, agentMAXV)
    agent(iagent, [agentXVEL, agentYVEL]) = agent(iagent, [agentXVEL, agentYVEL]) / norm(agent(iagent, [agentXVEL, agentYVEL])) * agent(iagent, agentMAXV);
    end

    if agent(iagent, agentSTATE) == agentSTATEmoving
        ds = (agent(iagent, [agentXVEL, agentYVEL])) * dt;
    else
        ds = [0, 0];
    end

    % sum distance per agent
    stat_sum_distance(iagent) = stat_sum_distance(iagent) + norm(ds);

    agent(iagent, [agentXPOS, agentYPOS]) = agent(iagent, [agentXPOS, agentYPOS]) + ds;
end
```

8.2.4 Statistical evaluation

Listing 22: save_data.m

```
% save/update statistical data

stat_moving_agents(step) = sum(agent(:,agentSTATE)==agentSTATEmoving);

for iagent = 1 : agentcount
    if agent(iagent, agentSTATE) == agentSTATEmoving
        % calculate average distance
        stat_distance_to_go(step) = stat_distance_to_go(step) + (remainingdistance(iagent, agent(iagent,agentCDOOR))/stat_moving_agents(step));


        % count waiting agents
        if (remainingdistance(iagent, agent(iagent, agentCDOOR)) < 3*doorrange)
            % Differentiate between boarder and deboarder
```

```
            if agent(iagent, agentMODE) == agent_mode_enter_subway
                stat_waiting_agents(step,1) = stat_waiting_agents(step,1) + 1;
            else
                stat_waiting_agents(step,2) = stat_waiting_agents(step,2) + 1;
            end
                stat_sum_waiting(iagent) = stat_sum_waiting(iagent) + dt;
            else
        end

        % calculate min distance between startposition and heading door
        stat_min_distance(iagent) = norm(stat_start_position(iagent, :) - door(agent(iagent, agentCDOOR), [doorXPOS, doorYPOS]));
    end
end

for kdoor = 1:doorcount
    for iagent = 1:agentcount
        if (agent(iagent,agentSTATE) == agentSTATEmoving) && (agent(iagent, agentCDOOR) == kdoor)
            stat_approaching_to_door(step,kdoor) = stat_approaching_to_door(step,kdoor) + 1;
        end
    end
end

% if all agents boarded, save the time:
if (sum(agent(agent(:, agentMODE) ≠ agent_mode_enter_subway, agentSTATE) ≠ agentSTATEboarded) == 0) ...
        && (final_boarding_time == 0)
    final_boarding_time = t;
end

stat_boarded_per_door(step,:) = door(:, doorAGENT);
```

8.2.5 Plotting

Listing 23: paint.m

```
% plot current situation

if plotting_mode == plotDEFAULT
    if(mod(t,10) == 0)
        t
    end
else
```

```
    figure(my_figure);
    if plotting_mode == plotMAPview
        clf

        img_scale_factor = 10;

        axis(img_scale_factor*[border(1), border(1)+border(3), border(2), border(2)+border(4)]);
        axis equal;
        hold on;

        rgb = imread('Train_EC.jpg');
        for itrain = 1:traincount
            xtrain = obstacle(itrain, obstacleXCENTER) - obstacle(itrain, obstacleWIDTH)/2;
            ytrain = obstacle(itrain, obstacleYCENTER) - obstacle(itrain, obstacleHEIGHT)/2;

            % shift dots outside of the train image
            if traincount == 2
                if itrain == 1
                    y_offshift = 5;
                else
                    y_offshift = -5;
                end
            else
                y_offshift = 5;
            end

            image(img_scale_factor*(xtrain), img_scale_factor*(ytrain),rgb);

            for a = 1:60
                for b = 1:2
                    if(trainseats(itrain,a,b) ≥ 4)
                        plot(img_scale_factor*( xtrain + 150/60 * (a-0.5)), img_scale_factor*( y_offshift + ytrain - 2 + 3*b ), 'sr');
                    elseif(trainseats(itrain,a,b) == 3)
                        plot(img_scale_factor*( xtrain + 150/60 * (a-0.5)), img_scale_factor*( y_offshift + ytrain - 2 + 3*b ), 'sm');
                    elseif(trainseats(itrain,a,b) == 2)
                        plot(img_scale_factor*( xtrain + 150/60 * (a-0.5)), img_scale_factor*( y_offshift + ytrain - 2 + 3*b ), 'sy');
                    elseif(trainseats(itrain,a,b) == 1)
                        plot(img_scale_factor*( xtrain + 150/60 * (a-0.5)), img_scale_factor*( y_offshift + ytrain - 2 + 3*b ), 'sc');
                    else
```

```
                        plot(img_scale_factor*( xtrain + 150/60 * (a-0.5)), img_scale_factor*( y_offshift + ytrain - 2 + 3*b ), 'sg');
                    end
                end
            end
        end

        % plot the moving agents (red color = no group, blue color = some
        % group)
        plot(img_scale_factor*agent(((agent(:,agentSTATE)==agentSTATEmoving) & (agent(:,agentGROUP)==agentGROUPnone)),agentXPOS), ...
            img_scale_factor*agent(((agent(:,agentSTATE)==agentSTATEmoving) & (agent(:,agentGROUP)==agentGROUPnone)),agentYPOS), ...
            'r.')
        plot(img_scale_factor*agent(((agent(:,agentSTATE)==agentSTATEmoving) & (agent(:,agentGROUP)≠agentGROUPnone)),agentXPOS), ...
            img_scale_factor*agent(((agent(:,agentSTATE)==agentSTATEmoving) & (agent(:,agentGROUP)≠agentGROUPnone)),agentYPOS), ...
            'm.')

        plot(img_scale_factor*door(door(:,doorSTATE)>10*dt,doorXPOS), 10*door(door(:,doorSTATE)>10*dt,doorYPOS), 'xr')
        plot(img_scale_factor*door(door(:,doorSTATE)<10*dt,doorXPOS), 10*door(door(:,doorSTATE)<10*dt,doorYPOS), 'og')

        for iobstacle = (traincount+1):obstaclecount
            rect(1,:) = [obstacle(iobstacle,obstacleXCENTER) - obstacle(iobstacle,obstacleWIDTH)/2 , obstacle(iobstacle,obstacleYCENTER) - obstacle(iobstacle,obstacleHEIGHT)/2];
            rect(2,:) = [obstacle(iobstacle,obstacleXCENTER) - obstacle(iobstacle,obstacleWIDTH)/2 , obstacle(iobstacle,obstacleYCENTER) + obstacle(iobstacle,obstacleHEIGHT)/2];
            rect(3,:) = [obstacle(iobstacle,obstacleXCENTER) + obstacle(iobstacle,obstacleWIDTH)/2 , obstacle(iobstacle,obstacleYCENTER) + obstacle(iobstacle,obstacleHEIGHT)/2];
            rect(4,:) = [obstacle(iobstacle,obstacleXCENTER) + obstacle(iobstacle,obstacleWIDTH)/2 , obstacle(iobstacle,obstacleYCENTER) - obstacle(iobstacle,obstacleHEIGHT)/2];
            rect(5,:) = [obstacle(iobstacle,obstacleXCENTER) - obstacle(iobstacle,obstacleWIDTH)/2 , obstacle(iobstacle,obstacleYCENTER) - obstacle(iobstacle,obstacleHEIGHT)/2];
            if (obstacle(iobstacle,obstacleSTART) ≤ t) && (obstacle(iobstacle,obstacleEND) ≥ t)
                plot(img_scale_factor*rect(:,1),img_scale_factor*rect(:,2),'k-');
            else
                plot(img_scale_factor*rect(:,1),img_scale_factor*rect(:,2),'k.:');
            end
```

```
        end

        text(img_scale_factor*(border(1)+1),img_scale_factor*(border(2)+1),
            num2str(t));

    end

    if plotting_mode == plotGRAPHview
        clf
        plot_saved_data
    end
end
```

Listing 24: plot_saved_data.m

```
timevector = dt:dt:t;

subplot(3,3,1)
plot_saved_approaching;

subplot(3,3,2)
plot_saved_moving;

subplot(3,3,3)
plot_saved_distance;

subplot(3,3,4)
plot_saved_deboarded;

subplot(3,3,5)
plot_saved_boarded;

subplot(3,3,6)
plot_saved_waiting;

subplot(3,3,7)
plot_saved_time_waited;

subplot(3,3,8)
plot_saved_redecisions;

subplot(3,3,9)
plot_saved_distance_walked;
```

Listing 25: plot_saved_approaching.m

```
hold on
```

```
if(1 ≤ doorMODEsum)
    plot(timevector, stat_approaching_to_door(1:step, door(:, doorMODE)==1), '-')
end
if(2 ≤ doorMODEsum)
    plot(timevector, stat_approaching_to_door(1:step, door(:, doorMODE)==2), '-.')
end
if(3 ≤ doorMODEsum)
    plot(timevector, stat_approaching_to_door(1:step, door(:, doorMODE)==3), '--')
end
if(4 ≤ doorMODEsum)
    plot(timevector, stat_approaching_to_door(1:step, door(:, doorMODE)==4), ':')
end
if(5 ≤ doorMODEsum)
    plot(timevector, stat_approaching_to_door(1:step, door(:, doorMODE)==5), '-')
end
xlabel('time')
ylabel('# approaching')
hold off
```

Listing 26: plot_saved_moving.m

```
plot(timevector, stat_moving_agents(1:step, :))
xlabel('time')
ylabel('# moving')
```

Listing 27: plot_saved_distance.m

```
plot(timevector, stat_distance_to_go(1:step, :))
xlabel('time')
ylabel('distance')
```

Listing 28: plot_saved_deboarded.m

```
plot(timevector, (stat_boarded_per_door(1:step, :) < 0) .* abs(stat_boarded_per_door(1:step, :)))
xlabel('time')
ylabel('# deboarded')
```

Listing 29: plot_saved_boarded.m

```
plot(timevector, (stat_boarded_per_door(1:step, :) > 0) .*
    stat_boarded_per_door(1:step, :))
xlabel('time')
ylabel('# boarded')
```

Listing 30: plot_saved_waiting.m

```
plot(timevector, stat_waiting_agents(1:step, :))
hold on
plot(timevector, sum(stat_waiting_agents(1:step, :)')', 'r')
hold off
legend('to subway', 'to train', 'sum')
xlabel('time')
ylabel('# waiting')
```

Listing 31: plot_saved_time_waited.m

```
hold on
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEmoving) .*
    stat_sum_waiting, 'b.')
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEboarded) .*
    stat_sum_waiting, 'g.')
plot_separation_lines;
legend('moving', 'boarded')
hold off
xlabel('agent')
ylabel('time waited')
```

Listing 32: plot_saved_redecisions.m

```
hold on
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEmoving) .*
    stat_sum_decision, 'b.')
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEboarded) .*
    stat_sum_decision, 'g.')
plot_separation_lines;
legend('moving', 'boarded')
hold off
xlabel('agent')
ylabel('redecisions')
```

Listing 33: plot_saved_distance_walked.m

```
hold on
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEmoving) .*
    stat_sum_distance, 'b.')
plot(1:agentcount, (agent(:, agentSTATE) == agentSTATEboarded) .*
    stat_sum_distance, 'g.')
plot(1:agentcount, stat_min_distance, 'r.')
plot_separation_lines;
legend('moving', 'boarded', 'minimal-dist')
hold off
xlabel('agent')
ylabel('distance walked')
```

Listing 34: plot_separation_lines.m

```

dimension = axis;
for iclass = 1 : class_count
    for itype = 1 : agent_type_count
        plot([agent_part_sum(iclass,itype),agent_part_sum(iclass,itype)],[
            dimension(3), dimension(4)],'k');
    end
end
```

8.2.6 Saving and loading simulation data

Listing 35: video_capture.m

```
% add picture to video
if video_mode == videoON

    new_Frame = getframe(my_figure);
    aviobj = addframe(aviobj, new_Frame);

    if t == tmax
        aviobj = close(aviobj);
    end

end
```

Listing 36: data_export.m

```
 % save current workspace to file if activated
 if data_export_mode == data_export_ON
     if (mod(step,round(save_dt/dt)) == 0)
```

```
        save(strcat(save_file_prefix,int2str(step),save_file_suffix))
    end
end
```

Listing 37: load_and_playback.m

```
% playback saved simulation keyframes in real time
    init_style
for i = 1:round(tmax/save_dt)
    filename = strcat(save_file_prefix,num2str(i*(round(save_dt/dt))),
        save_file_suffix)
    load(filename)
    plotting_mode = plotMAPview;
    paint
    pause(save_dt)
end
```

8.3 Simulation Results

This is the complete list of all simulated test cases, that have been used for the analysis in *section 5.3*. There are 5 samples per test case. The number of the test case corresponds to the variable that needs to be set in order to initialize the Matlab-program. For each measurement in each test case, there is the calculated average and standard deviation in the two grey bottom lines. The last column indicates the number of agents that have not reached their destination until the end of the simulation. Especially for the very crowded setup and the random door decision mode, this number can get quite relevant. The order of the other columns matches the list of statistical measurements in *section 5.2*.

1 Standard OT 150 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
61.900	29.647	11.497	46.921	17.070	12.769	7.812	5.750	2.679	8.272	4.000
62.450	30.328	12.605	47.197	18.597	12.847	7.837	5.090	2.462	8.933	11.000
61.100	29.186	11.570	45.697	17.655	11.729	6.783	5.542	2.934	8.380	4.000
63.500	29.541	11.494	46.675	17.829	12.000	8.342	4.876	2.247	8.530	3.000
65.600	30.396	12.861	48.090	19.040	12.624	8.000	4.906	2.437	8.912	4.000
62.910	**29.820**	**12.005**	**46.916**	**18.038**	**12.394**	**7.755**	**5.233**	**2.552**	**8.605**	**5.200**
3.023	**0.275**	**0.450**	**0.750**	**0.611**	**0.249**	**0.340**	**0.154**	**0.069**	**0.092**	**10.700**

2 Standard TT 2x150 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
75.650	26.887	12.879	41.778	19.495	11.637	8.425	3.932	2.568	6.623	0.000
75.800	27.306	13.328	42.510	19.835	11.287	8.249	3.784	2.402	6.870	0.000
75.250	26.831	12.794	42.162	19.155	11.346	7.498	3.536	2.413	6.959	0.000
77.200	26.526	11.859	41.453	17.680	11.591	8.313	3.868	2.537	6.543	0.000
75.150	26.587	12.264	41.596	18.087	11.217	7.655	3.728	2.470	6.455	0.000
75.810	**26.827**	**12.625**	**41.900**	**18.850**	**11.416**	**8.028**	**3.770**	**2.478**	**6.690**	**0.000**
0.677	**0.095**	**0.326**	**0.187**	**0.858**	**0.035**	**0.177**	**0.023**	**0.005**	**0.047**	**0.000**

3 Few OT 75 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
40.100	21.594	7.473	35.350	12.271	5.626	4.569	5.511	2.765	5.140	3.000
37.450	21.192	6.855	35.310	11.655	4.570	3.750	5.265	2.439	4.967	1.000
37.250	21.521	6.886	35.998	11.436	5.366	3.906	4.646	2.514	5.085	2.000
36.650	21.806	6.399	35.733	10.392	5.917	4.649	5.755	3.257	5.317	1.000
34.300	20.337	6.048	33.662	9.347	5.787	4.507	5.740	3.155	4.741	0.000
37.150	**21.290**	**6.732**	**35.211**	**11.020**	**5.453**	**4.276**	**5.383**	**2.826**	**5.050**	**1.400**
4.294	**0.332**	**0.291**	**0.830**	**1.334**	**0.286**	**0.173**	**0.210**	**0.136**	**0.046**	**1.300**

4 Few TT 2x75 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
34.650	17.764	6.505	28.754	9.706	5.767	4.620	3.405	1.877	3.425	0.000
36.850	18.617	6.968	30.189	11.251	5.891	4.629	3.175	1.948	4.388	0.000
36.850	18.133	7.316	29.766	11.620	5.181	4.713	3.159	2.200	3.847	0.000
34.800	17.969	6.760	29.368	10.704	5.417	4.092	3.413	2.371	3.757	0.000
35.350	17.986	6.932	29.173	10.637	5.657	4.525	4.365	2.982	3.869	0.000
35.700	**18.094**	**6.896**	**29.450**	**10.784**	**5.582**	**4.516**	**3.503**	**2.275**	**3.857**	**0.000**
1.170	**0.103**	**0.088**	**0.303**	**0.527**	**0.081**	**0.061**	**0.247**	**0.195**	**0.120**	**0.000**

5 Too Many OT 300 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
82.950	42.529	15.921	69.423	25.544	20.227	11.884	5.565	2.696	11.193	23.000
84.700	42.288	15.907	69.375	26.238	20.016	12.362	5.840	3.079	11.021	19.000
81.200	41.834	15.704	67.916	25.790	19.465	11.573	5.346	2.547	10.750	21.000
83.350	41.938	15.848	68.752	26.432	19.424	11.811	5.546	2.810	10.890	24.000
86.900	42.410	15.799	67.790	24.027	19.527	12.041	5.317	2.588	11.147	20.000
83.820	**42.200**	**15.836**	**68.651**	**25.606**	**19.732**	**11.934**	**5.523**	**2.744**	**11.000**	**21.400**
4.526	**0.091**	**0.008**	**0.603**	**0.903**	**0.133**	**0.086**	**0.044**	**0.045**	**0.033**	**4.300**

6 Too Many TT 2x300 Agents

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
89.400	39.991	16.329	64.216	27.401	18.276	10.877	3.964	2.606	9.531	27.000
89.050	40.100	15.957	64.873	27.823	18.082	10.443	3.802	2.559	9.037	30.000
89.400	40.423	17.175	64.978	28.240	18.402	10.943	4.108	2.613	9.865	27.000
89.100	39.911	16.411	64.349	27.212	18.431	11.144	3.943	2.600	9.323	28.000
89.000	39.929	16.394	64.231	27.435	18.239	10.419	3.945	2.746	9.440	25.000
89.190	**40.071**	**16.453**	**64.529**	**27.622**	**18.286**	**10.765**	**3.952**	**2.625**	**9.439**	**27.400**
0.038	**0.044**	**0.197**	**0.135**	**0.169**	**0.020**	**0.103**	**0.012**	**0.005**	**0.091**	**3.300**

7 Doordecision Walk

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
88.800	29.446	15.995	46.219	23.876	12.491	10.131	1.730	0.744	9.171	2.000
87.400	29.366	16.603	46.150	24.820	12.270	9.470	1.784	0.807	8.861	0.000
73.600	29.396	15.247	46.643	22.485	12.339	9.029	1.722	0.696	9.818	2.000
88.700	29.315	15.375	46.715	23.956	12.638	9.630	1.713	0.708	9.870	6.000
88.800	29.526	15.509	46.766	23.194	12.668	9.654	1.722	0.710	9.667	5.000
85.460	**29.410**	**15.746**	**46.499**	**23.666**	**12.481**	**9.583**	**1.734**	**0.733**	**9.478**	**3.000**
44.308	**0.006**	**0.310**	**0.085**	**0.769**	**0.031**	**0.157**	**0.001**	**0.002**	**0.195**	**6.000**

8 Doordecision Queue

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
61.200	25.481	8.403	40.439	12.990	9.368	6.685	27.016	29.284	4.260	0.000
65.050	25.402	9.253	40.627	14.575	8.277	6.432	33.317	38.217	4.466	2.000
62.450	25.665	9.467	40.798	15.944	8.466	6.776	35.500	40.514	4.345	3.000
56.700	25.538	9.043	41.006	15.087	8.458	6.428	38.316	42.204	4.403	1.000
65.100	25.601	9.426	41.160	15.399	7.808	6.494	44.692	50.620	4.520	1.000
62.100	**25.537**	**9.118**	**40.806**	**14.799**	**8.475**	**6.563**	**35.768**	**40.168**	**4.399**	**1.400**
11.949	**0.010**	**0.188**	**0.083**	**1.269**	**0.321**	**0.025**	**42.202**	**58.944**	**0.010**	**1.300**

9 Doordecision Wait

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
73.550	36.762	17.086	64.487	30.654	2.593	1.714	453.460	283.250	7.017	8.000
69.650	34.515	15.604	59.869	26.887	2.631	1.825	406.080	241.010	6.887	8.000
71.800	37.282	17.124	64.554	29.698	2.999	2.527	458.170	291.140	7.553	8.000
65.700	34.534	15.362	60.559	27.247	2.613	1.969	410.590	260.270	6.890	8.000
78.400	37.162	18.335	64.709	32.512	2.748	1.968	448.670	298.580	6.883	7.000
71.820	**36.051**	**16.702**	**62.836**	**29.400**	**2.717**	**2.000**	**435.394**	**274.850**	**7.046**	**7.800**
22.113	**1.979**	**1.498**	**5.793**	**5.574**	**0.028**	**0.098**	**623.982**	**564.190**	**0.084**	**0.200**

10 Doordecision Random

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
89.050	55.495	19.459	97.933	35.225	2.903	2.199	916.030	332.740	9.297	185.000
89.900	54.120	20.045	96.691	35.207	2.679	1.838	886.390	340.120	9.764	176.000
89.050	54.520	18.165	96.837	33.334	2.605	1.795	899.890	323.040	8.797	186.000
89.050	55.136	18.319	97.886	32.792	2.753	2.515	905.450	310.660	9.480	181.000
89.150	54.549	18.868	96.719	34.699	2.539	2.066	896.380	328.380	9.385	182.000
89.240	**54.764**	**18.971**	**97.213**	**34.251**	**2.696**	**2.082**	**900.828**	**326.988**	**9.344**	**182.000**
0.138	**0.298**	**0.619**	**0.407**	**1.258**	**0.020**	**0.086**	**120.396**	**122.416**	**0.124**	**15.500**

11 Lazy 0.0

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
63.200	25.396	9.204	40.731	15.318	8.721	6.196	27.708	30.521	4.226	0.000
56.300	25.315	9.010	40.345	14.601	8.262	6.547	35.344	39.317	4.489	2.000
69.100	25.871	9.998	41.506	16.172	8.004	6.696	41.220	44.561	4.520	1.000
71.900	25.564	9.712	40.751	16.357	8.747	6.641	30.484	34.906	4.403	1.000
72.400	25.600	9.791	40.948	16.376	8.332	6.283	36.400	40.234	4.403	1.000
66.580	**25.549**	**9.543**	**40.856**	**15.765**	**8.413**	**6.472**	**34.231**	**37.908**	**4.408**	**1.000**
46.407	**0.046**	**0.174**	**0.180**	**0.611**	**0.101**	**0.049**	**27.845**	**28.809**	**0.013**	**0.500**

12 Lazy 0.1

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
48.100	25.198	8.518	39.542	13.469	9.246	6.727	15.232	14.408	4.726	0.000
48.200	25.010	8.632	39.384	13.727	9.472	7.008	16.316	16.913	4.706	0.000
45.250	24.762	8.566	39.161	13.444	9.200	6.850	16.656	16.385	4.531	0.000
46.350	24.927	8.960	39.625	13.956	9.229	6.683	18.080	18.296	4.645	0.000
50.700	25.263	8.914	39.732	13.960	9.433	6.860	14.832	14.786	4.614	0.000
47.720	**25.032**	**8.718**	**39.489**	**13.711**	**9.316**	**6.825**	**16.223**	**16.158**	**4.644**	**0.000**
4.308	**0.041**	**0.042**	**0.050**	**0.063**	**0.016**	**0.016**	**1.640**	**2.534**	**0.006**	**0.000**

13 Lazy 0.2

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
51.900	25.575	10.178	40.594	15.922	9.440	7.114	11.176	8.947	5.491	0.000
48.200	25.516	9.982	40.420	15.506	9.437	7.236	9.700	8.527	5.350	0.000
47.400	25.382	9.345	39.851	14.772	9.259	6.274	9.156	7.408	5.197	0.000
46.700	25.551	9.287	40.420	14.318	9.614	7.057	10.492	8.710	5.403	0.000
51.250	25.416	9.125	40.571	14.579	9.542	6.676	10.240	8.618	5.178	0.000
49.090	**25.488**	**9.583**	**40.371**	**15.019**	**9.458**	**6.871**	**10.153**	**8.442**	**5.324**	**0.000**
5.480	**0.007**	**0.217**	**0.091**	**0.450**	**0.018**	**0.155**	**0.592**	**0.359**	**0.018**	**0.000**

14 Lazy 0.3

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
50.450	25.217	9.417	39.801	14.225	10.447	6.742	5.824	3.868	5.560	0.000
50.950	25.569	9.946	39.764	14.808	10.749	6.783	6.172	4.165	5.670	0.000
52.300	25.633	9.921	40.182	15.317	10.340	6.875	5.932	4.033	6.004	0.000
50.900	25.315	9.474	39.277	14.042	10.601	6.537	5.600	3.794	5.645	0.000
48.850	24.800	9.078	38.785	13.651	10.522	6.466	6.028	4.161	5.067	0.000
50.690	**25.307**	**9.567**	**39.562**	**14.409**	**10.532**	**6.681**	**5.911**	**4.004**	**5.589**	**0.000**
1.537	**0.110**	**0.135**	**0.292**	**0.432**	**0.024**	**0.030**	**0.047**	**0.028**	**0.114**	**0.000**

15 Lazy 0.4

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
74.300	26.329	11.769	41.028	17.652	10.963	6.654	4.744	2.978	6.176	0.000
71.700	26.316	11.383	41.072	16.669	10.835	7.126	4.632	2.827	6.344	0.000
57.950	25.978	10.746	40.475	15.526	11.283	6.839	4.532	2.930	6.404	0.000
56.100	25.607	10.153	40.070	14.644	11.385	7.253	5.216	3.415	5.949	0.000
74.300	26.143	10.776	40.842	16.222	10.910	7.194	5.032	3.260	6.404	0.000
66.870	**26.075**	**10.965**	**40.697**	**16.143**	**11.075**	**7.013**	**4.831**	**3.082**	**6.255**	**0.000**
82.325	**0.089**	**0.391**	**0.178**	**1.297**	**0.059**	**0.066**	**0.081**	**0.060**	**0.038**	**0.000**

16 Lazy 0.5

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
76.400	26.403	11.550	40.803	17.071	11.460	7.888	4.144	2.567	6.366	0.000
78.300	26.675	11.721	41.689	17.109	11.545	7.996	4.000	2.500	6.601	0.000
75.400	26.514	12.451	41.400	18.006	11.281	7.764	3.588	2.540	6.485	0.000
73.450	25.916	10.417	40.696	16.092	11.131	7.216	3.940	2.648	6.291	0.000
75.750	26.272	11.086	41.527	16.940	11.057	7.481	3.964	2.667	6.659	0.000
75.860	**26.356**	**11.445**	**41.223**	**17.044**	**11.295**	**7.669**	**3.927**	**2.584**	**6.480**	**0.000**
3.069	**0.082**	**0.571**	**0.199**	**0.462**	**0.043**	**0.101**	**0.042**	**0.005**	**0.024**	**0.000**

17 Lazy 0.6

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
80.950	27.492	13.568	42.712	19.859	11.856	8.485	3.848	2.376	7.082	0.000
75.650	26.377	11.105	41.130	16.096	11.391	8.034	3.328	2.148	6.849	0.000
78.650	26.584	11.634	41.571	17.330	11.515	7.155	2.944	1.909	6.744	0.000
75.450	26.383	11.651	41.231	17.617	11.432	8.114	3.112	2.091	6.716	0.000
78.600	27.221	12.742	42.578	19.020	11.467	7.924	3.232	2.116	7.235	0.000
77.860	**26.811**	**12.140**	**41.844**	**17.984**	**11.532**	**7.942**	**3.293**	**2.128**	**6.925**	**0.000**
5.353	**0.264**	**0.992**	**0.563**	**2.179**	**0.035**	**0.238**	**0.117**	**0.028**	**0.051**	**0.000**

18 Lazy 0.7

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
87.500	29.549	17.019	46.921	26.491	12.275	9.477	2.468	1.434	7.895	0.000
81.850	27.953	14.702	43.668	21.437	11.947	8.930	2.408	1.435	7.122	0.000
87.950	29.511	17.278	46.693	26.205	12.286	9.014	2.488	1.435	7.506	0.000
61.350	26.496	10.929	41.420	16.243	11.280	7.344	2.520	1.506	6.898	0.000
83.500	27.479	13.323	42.785	19.958	11.910	8.487	2.692	1.582	7.208	0.000
80.430	**28.198**	**14.650**	**44.297**	**22.067**	**11.940**	**8.650**	**2.515**	**1.478**	**7.326**	**0.000**
120.506	**1.756**	**7.032**	**5.896**	**18.865**	**0.167**	**0.657**	**0.011**	**0.004**	**0.149**	**0.000**

19 Lazy 0.8

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
81.100	27.616	13.139	43.496	19.736	12.037	8.022	2.288	1.118	7.088	0.000
89.850	29.661	17.734	47.100	27.322	12.357	9.638	2.169	1.173	7.925	1.000
77.900	26.980	12.399	42.298	18.178	11.359	7.763	2.356	1.340	7.261	0.000
86.650	28.442	15.930	44.415	24.081	12.056	9.091	2.344	1.213	7.359	0.000
88.800	29.248	17.148	45.995	25.940	12.412	10.054	2.339	1.343	7.667	2.000
84.860	**28.389**	**15.270**	**44.661**	**23.051**	**12.044**	**8.913**	**2.299**	**1.237**	**7.460**	**0.600**
26.552	**1.235**	**5.704**	**3.682**	**15.596**	**0.176**	**0.994**	**0.006**	**0.010**	**0.112**	**0.800**

20 Lazy 0.9

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
88.900	29.239	16.391	45.880	24.778	12.265	9.251	2.040	0.928	7.834	1.000
88.600	27.965	14.899	43.445	21.971	12.222	9.106	1.924	0.854	7.506	0.000
87.450	28.720	15.693	45.306	25.045	11.946	9.467	2.128	1.033	7.525	0.000
89.350	29.334	17.309	46.964	26.601	12.444	9.426	2.061	0.953	7.982	2.000
88.800	29.699	17.064	46.673	26.334	12.883	9.942	2.089	0.998	8.328	2.000
88.620	**28.991**	**16.271**	**45.654**	**24.946**	**12.352**	**9.438**	**2.048**	**0.953**	**7.835**	**1.000**
0.503	**0.452**	**0.984**	**1.952**	**3.389**	**0.120**	**0.100**	**0.006**	**0.005**	**0.117**	**1.000**

21 Lazy 1.0

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
89.450	29.472	14.944	46.487	22.713	12.566	9.271	1.744	0.710	9.090	0.000
88.800	29.368	15.016	46.444	22.618	12.376	9.438	1.719	0.736	9.378	1.000
89.100	29.761	17.506	47.078	26.687	12.587	9.971	1.746	0.807	9.160	6.000
88.650	29.210	15.438	45.801	23.053	12.489	10.115	1.699	0.703	9.078	1.000
89.500	29.674	16.461	47.284	25.336	12.402	9.844	1.756	0.736	9.542	8.000
89.100	**29.497**	**15.873**	**46.619**	**24.081**	**12.484**	**9.728**	**1.733**	**0.738**	**9.249**	**3.200**
0.144	**0.050**	**1.200**	**0.343**	**3.359**	**0.009**	**0.129**	**0.001**	**0.002**	**0.041**	**12.700**

22 Decision Step Frequency 1 per second

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
77.250	26.993	13.002	42.156	19.024	11.557	8.072	3.804	2.428	6.723	0.000
76.650	27.122	12.669	42.358	19.364	11.799	7.871	4.000	2.571	6.821	0.000
76.700	26.557	12.051	41.857	18.455	11.432	8.039	3.932	2.616	6.404	0.000
76.250	26.396	11.340	41.182	16.534	11.160	7.468	3.976	2.693	6.499	0.000
74.200	26.349	11.230	40.944	16.671	11.818	7.929	3.984	2.551	6.558	0.000
76.210	**26.683**	**12.058**	**41.699**	**18.010**	**11.553**	**7.876**	**3.939**	**2.572**	**6.601**	**0.000**
1.389	**0.125**	**0.616**	**0.376**	**1.758**	**0.075**	**0.059**	**0.006**	**0.009**	**0.029**	**0.000**

23 Decision Step Frequency 100 per second

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
78.350	26.704	12.246	41.584	17.869	11.447	7.742	4.080	2.750	6.433	0.000
60.150	25.992	10.378	40.397	14.872	11.121	7.416	4.508	3.040	6.558	0.000
76.450	26.556	11.930	41.137	17.345	11.816	7.897	4.100	2.790	6.396	0.000
75.950	26.074	11.151	40.819	16.471	10.824	6.843	3.976	2.683	6.793	0.000
78.150	26.824	13.421	41.784	19.215	11.238	8.182	3.992	2.466	6.898	0.000
73.810	**26.430**	**11.825**	**41.144**	**17.154**	**11.289**	**7.616**	**4.131**	**2.746**	**6.616**	**0.000**
59.398	**0.141**	**1.321**	**0.317**	**2.617**	**0.137**	**0.263**	**0.047**	**0.043**	**0.049**	**0.000**

24 one decision per agent

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
82.000	30.509	17.404	49.319	26.529	9.931	8.841	1.172	0.565	8.116	0.000
88.600	30.708	18.028	49.539	27.956	10.136	9.678	1.218	0.662	8.328	2.000
82.250	30.145	16.889	49.022	26.438	10.365	9.017	1.157	0.535	8.317	1.000
83.450	30.150	17.390	48.634	26.009	9.787	8.923	1.161	0.545	8.079	1.000
81.000	29.921	16.440	47.882	24.340	10.328	9.200	1.173	0.602	8.166	2.000
83.460	**30.287**	**17.230**	**48.879**	**26.254**	**10.109**	**9.132**	**1.176**	**0.582**	**8.201**	**1.200**
9.017	**0.100**	**0.358**	**0.426**	**1.682**	**0.062**	**0.111**	**0.001**	**0.003**	**0.013**	**0.700**

25 ten decisions per agent

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
76.000	26.302	11.981	40.988	17.318	11.353	7.481	3.712	2.284	6.630	0.000
77.850	26.145	11.162	40.874	16.564	11.147	7.322	4.140	2.513	6.507	0.000
76.700	26.725	12.544	41.817	19.144	11.714	8.408	4.116	2.582	6.644	0.000
76.350	26.282	12.025	41.074	18.014	11.499	7.422	3.680	2.248	6.521	0.000
75.400	26.037	11.076	40.480	16.226	11.370	7.588	3.696	2.412	6.536	0.000
76.460	**26.298**	**11.758**	**41.047**	**17.453**	**11.417**	**7.644**	**3.869**	**2.408**	**6.568**	**0.000**
0.834	**0.069**	**0.390**	**0.237**	**1.372**	**0.044**	**0.192**	**0.056**	**0.021**	**0.004**	**0.000**

26 no patience

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
76.950	27.250	13.536	42.896	20.635	11.532	8.355	3.796	2.457	6.680	0.000
74.900	25.769	11.279	40.642	16.763	10.942	7.985	3.588	2.471	6.411	0.000
76.400	26.188	11.927	40.953	17.796	10.962	7.604	3.580	2.296	6.665	0.000
72.900	25.918	10.850	40.418	16.980	11.138	7.490	3.524	2.415	6.433	0.000
71.150	25.800	10.296	39.951	15.279	11.339	6.873	3.976	2.636	6.299	0.000
74.460	**26.185**	**11.578**	**40.972**	**17.491**	**11.183**	**7.661**	**3.693**	**2.455**	**6.497**	**0.000**
5.887	**0.382**	**1.555**	**1.290**	**3.915**	**0.064**	**0.310**	**0.036**	**0.015**	**0.028**	**0.000**

27 patience high

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
73.200	26.174	11.502	40.974	17.778	10.986	7.700	3.712	2.718	6.359	0.000
78.000	27.267	12.934	42.362	19.080	11.236	7.863	3.956	2.636	6.814	0.000
77.300	26.652	11.863	41.722	17.390	11.449	7.327	4.256	2.772	6.694	0.000
75.450	26.981	13.059	42.573	19.937	10.963	7.875	3.940	2.716	6.499	0.000
56.900	25.947	10.490	40.072	15.333	11.413	7.276	3.848	2.430	6.389	0.000
72.170	**26.604**	**11.970**	**41.541**	**17.904**	**11.209**	**7.608**	**3.942**	**2.654**	**6.551**	**0.000**
76.325	**0.300**	**1.134**	**1.063**	**3.102**	**0.053**	**0.083**	**0.040**	**0.018**	**0.039**	**0.000**

28 velocity 1m/s

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
88.850	28.580	12.465	26.788	11.782	9.188	6.985	3.073	1.999	6.977	2.000
89.550	28.592	13.394	26.822	12.582	9.186	6.954	2.886	1.672	7.187	4.000
58.100	26.587	10.152	24.804	9.741	9.025	6.243	3.036	1.848	5.541	1.000
84.850	28.027	13.030	26.273	12.352	9.055	6.596	2.884	1.623	6.715	1.000
90.000	28.606	13.261	27.012	12.715	8.508	6.942	3.136	1.975	6.863	0.000
82.270	**28.078**	**12.460**	**26.340**	**11.834**	**8.992**	**6.744**	**3.003**	**1.823**	**6.657**	**1.600**
186.723	**0.755**	**1.791**	**0.812**	**1.497**	**0.079**	**0.104**	**0.013**	**0.029**	**0.419**	**2.300**

29 velocity 1.5 m/s

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
61.450	26.389	10.255	35.671	12.975	10.856	7.228	3.472	2.162	6.426	0.000
82.750	27.069	12.947	36.699	17.194	10.621	7.229	3.632	2.317	6.572	0.000
82.800	27.055	12.362	36.530	16.125	11.036	7.983	3.496	2.257	6.644	0.000
79.250	26.244	10.331	35.256	13.225	11.087	7.440	3.740	2.453	6.253	0.000
83.000	27.867	14.152	37.710	18.452	11.104	7.272	3.564	2.267	6.856	0.000
77.850	**26.925**	**12.009**	**36.373**	**15.594**	**10.941**	**7.430**	**3.581**	**2.291**	**6.550**	**0.000**
86.489	**0.419**	**2.872**	**0.915**	**5.870**	**0.042**	**0.103**	**0.012**	**0.011**	**0.052**	**0.000**

30 velocity 2.5+/-1.5 m/s

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
49.600	25.393	10.168	71.326	29.401	13.245	8.352	3.756	2.351	5.437	0.000
52.500	25.089	10.306	71.593	29.109	13.069	8.076	3.316	2.119	5.795	0.000
50.700	25.009	10.263	70.649	28.215	12.933	8.459	3.284	1.964	5.542	0.000
48.850	24.705	9.855	70.412	28.313	13.236	8.001	3.440	2.139	5.438	0.000
51.550	25.430	10.858	72.694	30.448	13.075	8.953	3.568	2.490	5.941	0.000
50.640	**25.125**	**10.290**	**71.335**	**29.097**	**13.112**	**8.368**	**3.473**	**2.213**	**5.631**	**0.000**
2.144	**0.089**	**0.132**	**0.809**	**0.828**	**0.017**	**0.143**	**0.038**	**0.043**	**0.051**	**0.000**

31 20% Groups

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
82.350	28.302	14.817	44.192	21.282	11.449	8.050	4.024	2.880	7.280	0.000
80.050	27.545	13.070	42.915	18.627	11.356	8.236	4.032	2.627	7.248	0.000
52.600	25.690	9.225	39.732	13.257	11.078	7.236	3.480	2.232	5.745	0.000
76.150	27.562	13.668	43.370	20.276	11.625	8.075	3.996	2.540	6.925	0.000
73.650	26.751	12.657	42.227	19.491	10.403	7.084	3.668	2.396	6.779	0.000
72.960	**27.170**	**12.687**	**42.487**	**18.587**	**11.182**	**7.736**	**3.840**	**2.535**	**6.795**	**0.000**
140.906	**0.985**	**4.408**	**2.882**	**9.836**	**0.229**	**0.285**	**0.064**	**0.060**	**0.390**	**0.000**

32 50% Groups

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
74.600	27.120	12.151	42.919	18.561	10.287	7.563	3.976	2.653	6.723	0.000
84.100	28.395	14.595	44.280	21.187	11.451	8.154	4.116	2.374	7.352	0.000
78.550	26.816	12.138	41.780	17.761	10.936	7.583	4.408	2.878	6.737	0.000
78.150	28.950	15.381	45.944	23.215	10.811	8.213	3.723	2.416	7.332	1.000
79.350	28.286	14.818	44.989	22.136	10.983	7.983	3.880	2.524	7.326	0.000
78.950	**27.913**	**13.817**	**43.982**	**20.572**	**10.894**	**7.899**	**4.021**	**2.569**	**7.094**	**0.200**
11.601	**0.820**	**2.412**	**2.733**	**5.439**	**0.174**	**0.096**	**0.067**	**0.041**	**0.111**	**0.200**

33 instant start

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
67.550	20.075	10.882	30.507	16.320	9.646	6.824	1.984	1.451	7.378	0.000
49.750	19.462	10.630	29.610	16.556	9.176	6.452	1.948	1.586	6.593	0.000
67.700	19.465	10.943	29.306	16.499	9.778	6.981	1.688	1.082	6.849	0.000
68.450	19.682	10.774	29.693	16.350	9.740	6.572	1.732	1.194	6.898	0.000
69.050	19.610	10.847	29.537	16.106	10.152	6.999	1.520	0.893	6.779	0.000
64.500	**19.659**	**10.815**	**29.731**	**16.366**	**9.698**	**6.766**	**1.774**	**1.241**	**6.899**	**0.000**
68.353	**0.063**	**0.014**	**0.209**	**0.031**	**0.122**	**0.060**	**0.037**	**0.078**	**0.085**	**0.000**

34 late start

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
81.500	34.382	12.814	55.023	19.767	12.935	7.769	5.492	3.315	6.514	0.000
65.000	33.605	11.841	53.611	18.862	12.629	7.311	6.440	4.729	6.215	0.000
61.100	33.247	11.204	53.097	18.075	12.462	7.069	6.372	4.621	5.793	0.000
65.300	33.673	11.536	53.379	18.028	12.771	7.928	6.124	3.769	6.176	0.000
62.000	33.453	11.255	53.078	17.820	12.461	7.192	6.296	4.135	6.107	0.000
66.980	**33.672**	**11.730**	**53.638**	**18.510**	**12.652**	**7.454**	**6.145**	**4.114**	**6.161**	**0.000**
69.237	**0.184**	**0.432**	**0.648**	**0.650**	**0.042**	**0.140**	**0.147**	**0.348**	**0.067**	**0.000**

35 small waiting area OT

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
57.800	30.674	10.000	49.166	13.398	12.546	8.633	6.582	3.004	8.822	9.000
52.850	30.795	8.526	49.632	12.557	12.886	8.167	6.828	3.367	8.017	7.000
57.850	30.968	9.849	49.157	13.864	12.595	8.956	6.000	3.081	8.663	8.000
54.350	31.047	9.066	49.275	12.465	12.579	8.370	6.689	3.193	8.341	10.000
56.250	30.571	9.543	48.175	12.652	13.006	8.199	6.370	3.102	8.594	8.000
55.820	**30.811**	**9.397**	**49.081**	**12.987**	**12.722**	**8.465**	**6.494**	**3.149**	**8.487**	**8.400**
4.802	**0.039**	**0.364**	**0.294**	**0.377**	**0.044**	**0.110**	**0.104**	**0.019**	**0.099**	**1.300**

36 small waiting area TT

f_b_time	m_b_time	d_b_time	m_distance	d_distance	m_waiting_t	d_waiting_t	m_decisions	d_decisions	d_door_distrib.	unboarded
78.200	30.670	14.556	48.422	20.967	12.447	9.275	4.804	2.772	8.589	0.000
75.800	31.033	14.920	49.188	21.539	12.267	8.725	5.512	3.328	8.824	0.000
75.950	30.625	14.776	48.042	21.843	12.499	9.647	4.980	2.947	8.483	0.000
78.050	30.591	15.266	48.046	21.808	12.423	9.501	4.564	2.592	8.775	0.000
74.900	30.067	13.890	47.403	20.096	12.242	9.685	4.920	2.777	8.567	0.000
76.580	**30.597**	**14.682**	**48.220**	**21.251**	**12.376**	**9.367**	**4.956**	**2.883**	**8.647**	**0.000**
2.153	**0.119**	**0.262**	**0.427**	**0.540**	**0.013**	**0.155**	**0.122**	**0.078**	**0.021**	**0.000**