

Technische Universität Dresden

Fachrichtung Mathematik

Institut für Wissenschaftliches Rechnen

**A knowledge-based routing
framework for pedestrian
dynamics simulation.**

Diplomarbeit

zur Erlangung des ersten akademischen Grades

Diplommathematiker

vorgelegt von

Name: Haensel

Vorname: David

geboren am: 31.03.1988

in: Eberswalde-Finow

Tag der Einreichung: 04.11.2014

Betreuer: Jun.-Prof. Dr. Kathrin Padberg-Gehle

Dr. Mohcine Chraïbi

CONTENTS

Contents	I
1. Introduction	1
1.1. Introduction	1
2. Preliminaries	4
2.1. Graph theory	4
2.2. Statistics	6
2.3. Related work	7
3. Methodology	9
3.1. Representing knowledge with a cognitive map	9
3.1.1. Requirements and preconditions	10
3.1.2. The metric map (NavigationGraph)	12
3.1.3. Used routes memory	17
3.1.4. Putting it all together: the cognitive map	18
3.2. Gathering information	18
3.2.1. Sensors and sensor manager	20
3.2.2. Implemented sensors	21
3.3. Making decisions	22
3.3.1. Global way finding	23
3.3.2. Local way finding	28
3.4. Routing the pedestrians	28

4. Simulations and results	32
4.1. Setup and preparation	33
4.1.1. Geometries	34
4.2. Analyzing different sensors with complete cognitive maps	34
4.2.1. LastDestinationSensor and no sensors	35
4.2.2. RoomToCorridorSensor	38
4.2.3. DensitySensor	40
4.2.4. SmokeSensor	45
4.3. Analyzing different sensors with empty cognitive maps	47
4.3.1. Mandatory sensors only	47
4.3.2. RoomToCorridorSensor	50
4.3.3. SmokeSensor	52
5. Conclusion and future work	55
5.1. Conclusion	55
5.2. Future work	56
A. Listings	59
Bibliography	66
Erklärung	68

INTRODUCTION

1.1. Introduction

The simulation of pedestrian dynamics provides important results for different applications. For architects the analysis of people flow is interesting during the planning of new facilities and exit routes. For organizers of large scale events a simulation of pedestrians could help to appraise the location. And for public transport companies the analysis of crowd behavior could help to plan the route network.

Hoogendoorn et al [7] divided pedestrian dynamics decision making into three levels, the strategic, tactical and operational level. The pre trip route planning and the choice of the final destination is done in the strategical level. It should be mentioned that at the strategical level no information about actual circumstances is available. Short term decisions like obstacle avoidance or route changes depending on actual situation are done at the tactical level. At this level additional information is available, like people flow or smoked rooms. At the operational level the pedestrian motion is modeled including interaction with other pedestrians.

Current routing mechanisms in pedestrian dynamics simulations are mostly based on shortest path calculation or quickest path approximation. Some of them already feature perception of congestion and jams in front of doors [8]. This perception leads to another route choice for some individual agents. But most of the routing implementations do not take individual knowledge or behavior into consideration. It is for example unrealistic to assume that pedestrians in a shopping mall take the shortest path only. In contrast one should assume that most pedestrians do not even know more than one emergency exit. To resign individual knowledge assumes

1. Introduction

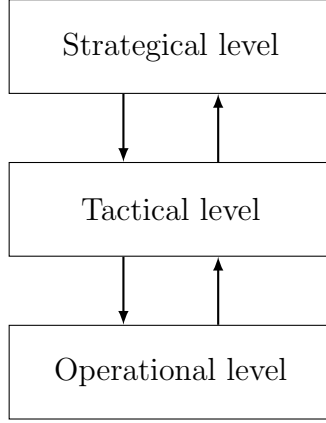


Figure 1.1.: **Three levels of operation**

Three levels of pedestrian dynamics and their relations according to [7, p. 128]

that every agent has perfect knowledge of the actual building.

To reach more realistic simulations it seems necessary to take some individual factors into consideration. Individual knowledge is the basis for those individuality. It is needed for individual decision making and social behavior.

Another important feature, the human perception, is often missing in actual implementation. If we want to consider dynamic circumstances which influence the route choice we need to have a perception layer. The information gathered from this perception layer should then be written in the aforementioned knowledge representation. This shows, that a versatile knowledge representation combined with perception possibilities and decision making is needed.

All implementations developed for this thesis are done with the Jülich Pedestrian Simulator, a collection of tools for pedestrian dynamics simulation developed at the research centre Jülich. For all simulations we used the generalized centrifugal force model [4]. This model is based on the centrifugal force model [19] which features social forces influencing the agents among each other. The generalized centrifugal force model uses ellipses as representation for pedestrians instead of circles. The force guiding every pedestrian P_i is calculated as follows:

$$m_i \frac{d^2 \vec{R}_i}{dt^2} = \vec{F}_i = \vec{F}_i^{drv} + \sum_{j \in \mathcal{N}_i} \vec{F}_{ij}^{rep} + \sum_{w \in \mathcal{W}_i} \vec{F}_{iw}^{rep}.$$

With the following forces:

\vec{F}_i^{drv} the driving force directing to the desired destination,

1. Introduction

\vec{F}_{ij}^{rep} the repulsive force between pedestrian j and pedestrian i and

\vec{F}_{iw}^{rep} the repulsive force emerging from obstacles and walls.

For more information we refer to [\[4\]](#).

PRELIMINARIES

In this chapter we want to give a short introduction to the used methods and theories. We present some fundamental graph theory which is used for the conceptual design of the routing framework in general and the cognitive map in particular. We recall a little statistics which are used for the analysis and evaluation in chapter 4 and at the end we present existing works related to the topic of this thesis.

2.1. Graph theory

The main topic of this work is the routing of pedestrians in evacuation simulations. It is nearby to use graph based structures for the representation of spatial knowledge for routing purposes. That is why we want to give a slightly introduction to the topic of graph theory especially the properties we are going to use later. The main structure we want to define is the directed graph. In contrast to the usually used definitions of graphs we need to have the possibility of multiple edges between vertices. That is why the following definition is more general and introduces a “counter” n for edges connecting the same vertices in the same direction.

Definition 2.1 (Directed graph):

A **directed graph** $G = G(V, E)$ is a pair of sets V and E with

- V called vertices and
- $E = \{(x, y, n) \in E \mid (x, y) \in V^2 \wedge n \in \mathbb{N}\}$ called edges.

2. Preliminaries

The graph $G' = (V', E')$ is called subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Additionally we define two functions mapping from edges E to vertices V and assign the corresponding source and destination to the edge.

Definition 2.2:

For all $(x, y, n) \in E$ the functions $source, dest : E \mapsto V$ are defined as

$$source((x, y, n)) := x \quad \text{and}$$

$$dest((x, y, n)) := y.$$

We call an edge e with $source(e) = dest(e)$ a loop. For the rest of the work we write (x, y) for the edge $(x, y, 1) \in E$.

Definition 2.3 (Cycle & acyclic graph):

A **cycle** of length l in a graph $G = G(V, E)$ is a sequence of distinct vertices $\{v_1, \dots, v_l\}$ with $l \geq 3$ which meets the following two condition:

$$\exists n \in \mathbb{N} \text{ with } (v_l, v_1, n) \in E$$

$$\forall i \in \{1, \dots, l-1\} \exists n \in \mathbb{N} \text{ with } (v_i, v_{i+1}, n) \in E.$$

A graph without any cycle is called **acyclic graph**.

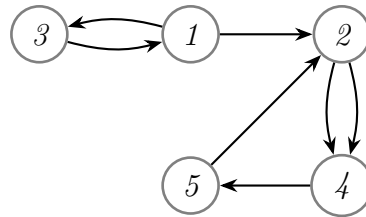
For the illustration of graphs we use graph diagrams. In those diagrams vertices are represented as shapes and connected by arrows pointing from the source to the destination of the corresponding edge.

Example 2.1:

The graph $G = G(V, E)$ with:

- $V = \{1, 2, 3, 4, 5\}$ and
- $E = \{(1, 2, 1), (1, 3, 1), (3, 1, 1), (2, 4, 1), (2, 4, 2), (4, 5, 1), (5, 2, 1)\}$

is represented by the following graph-diagram. The graph has a cycle of length 3 with the vertex sequence $\{2, 4, 5\}$.



2.2. Statistics

In this section we want to introduce the statistics needed for the later analysis. We will compare the evacuation time with descriptive statistics like histograms and box plots mainly and show the significance with Welch's t-test [18].

With the histogram we want to illustrate the distribution of the evacuation times. A histogram is a bar plot with the absolute frequencies of a sample. An example could be seen in table 4.4.

A box plot is used for the same purpose as the histogram and gives a quick overview of the distribution and especially the empirical variance of the sample. The box plot uses the 25% quartile and the 75% quartile to mark an inner box with a third inner line marking the median. In addition two whiskers show the minimum and maximum value. An example could be seen in table 4.5.

Welch's t-test is a statistical hypothesis test. It is a modification of the student's two sample t-test for testing two samples with unknown and possibly different variances. The null hypothesis (in the two sided test) is, that the expected values of two samples are equal and the alternative hypothesis that they are unequal:

$$H_0 : \mu_1 = \mu_2 \quad \text{and} \quad H_1 : \mu_1 \neq \mu_2.$$

The sample size needs to be large enough that the central limit theorem is valid. The following statistic t for two samples x and y is then compared with the student's t-distribution.

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{s_x^2}{n} + \frac{s_y^2}{m}}} = \frac{\frac{1}{n} \sum_{i=1}^n x_i - \frac{1}{m} \sum_{i=1}^m y_i}{\sqrt{\frac{s_x^2}{n} + \frac{s_y^2}{m}}}$$

$$\text{with: } s_x^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \frac{1}{n} \sum_{j=1}^n x_j)^2 \quad \text{and} \quad s_y^2 = \frac{1}{m-1} \sum_{i=1}^m (y_i - \frac{1}{m} \sum_{j=1}^m y_j)^2.$$

The degree of freedom ν for the t-distribution is as follow:

$$\nu = \frac{\left(\frac{s_x^2}{n} + \frac{s_y^2}{m}\right)^2}{\frac{\left(\frac{s_x^2}{n}\right)^2}{n-1} + \frac{\left(\frac{s_y^2}{m}\right)^2}{m-1}}.$$

2. Preliminaries

The null hypothesis could than be rejected in the following rejection region:

$$\{t \mid t < -t_{1-\alpha/2;\nu}\} \quad \text{or} \quad \{t \mid t > t_{1-\alpha/2;\nu}\}.$$

With the level of significance α and the student's t-distribution $t_{1-\alpha/2;\nu}$ with degrees of freedom ν .

In the analysis we use the p-value calculated with the intrinsic Welch's t-test method of R programming language. The p-value represents the probability of receiving a result like this under the assumption of the null hypothesis. That is why we want to have a small p-value to reject the null hypothesis and show a significantly difference of the expected values.

2.3. Related work

Cognitive map

The cognitive map is a concept introduced and analyzed by E.C. Tolman [17]. From his experiments with rats he deduced that rats are not simply navigating by stimuli and response but rather discover the space and store their acquired knowledge in a cognitive map like structure. This so called cognitive map enables rats to make decisions and logical conclusion.

B. Kuipers later analyzed the cognitive map from a more technical point of view [9] and [10]. In his work he described that the cognitive map aggregates information from observations to route description and fixed features which later are integrated in topological and the metric relations. An overview of the interaction between those five types of information can be seen in figure 2.1 from [10, p. 11].

Individual behavior

Individual behavior in pedestrian dynamics simulation is handled in different ways. Braun et al implemented individual behavior in the operational level by introducing new parameters like dependence (need of help) and altruism (willingness to help) and by introducing groups (like families) [3]. Those new parameters change the force calculation in the under laying social force model by Helbing [6]. Pelechano et al used the operational level too but also took individual knowledge of the building into consideration in the way finding [12]. Pan implemented a modular framework for human and social behavior which features typical behavior like queuing and

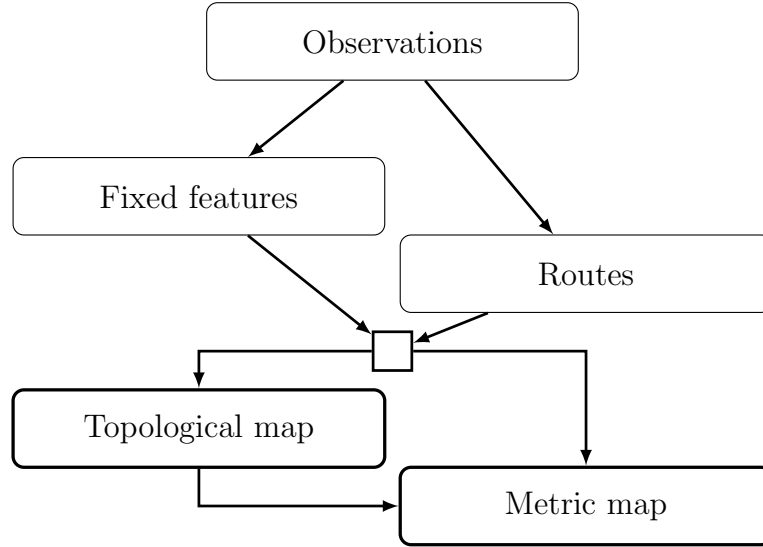


Figure 2.1.: **Cognitive map**

Five different types of information of a cognitive map according to Kuipers [10, p. 11]

leader following [11]. This new framework is also analyzed and compared with other evacuation simulations.

Other related work

There are several works about way finding. The book of Arthur and Passini [1] gives a good overview of the way finding in general and especially the process of finding a way in chapter 5. In [2] the route choice during a fire is discussed. They pay attention on the decision process and the way people choose the emergency exit (for example closed or open doors). In addition they discuss the influence of evacuation signs and the delay time after fire alarm. This delay time is also discussed in [13].

The influence of panic is discussed by Sime in several publications [15], [16]. His conclusion is that the delay of informing the public is of more importance than panic. Especially the typical behavior of panic could not be proved in the studies of a number of major fires.

METHODOLOGY

Modeling human behavior, perception and cognition is a complicated task. It is not the goal of this work to reach realistic human behavior or even to understand human cognition, but to emulate the behavior simplified enough to reach adequate simulations. With the created framework a powerful set of tools was build to model and emulate realistic behavior.

For achieving individual behavior and basic reasoning in pedestrian dynamics simulations it is necessary to have a versatile spatial knowledge representation. For this reason a simplified version of the cognitive map proposed by Tolman [17] was implemented and used for each agent separately (section 3.1). To model the information gathering of pedestrians a sensor structure was build to enrich the information stored in the cognitive map (section 3.2). Moreover the stored knowledge is used for individual decision making (section 3.3). The aforementioned three modules are encapsulated in the new created cognitive map router (section 3.4). Figure 3.1 shows an overview of the build modules which are described in detail in the following sections.

3.1. Representing knowledge with a cognitive map

The knowledge representation is the central module for the new routing mechanism. It has to fulfil several requirements (section 3.1.1). One of them is the possibility to represent different spatial knowledge. Another one is the possibility to store knowledge individually to achieve the goal of individual behaviour.

For the data structure we use the concept of the cognitive map proposed by

3. Methodology

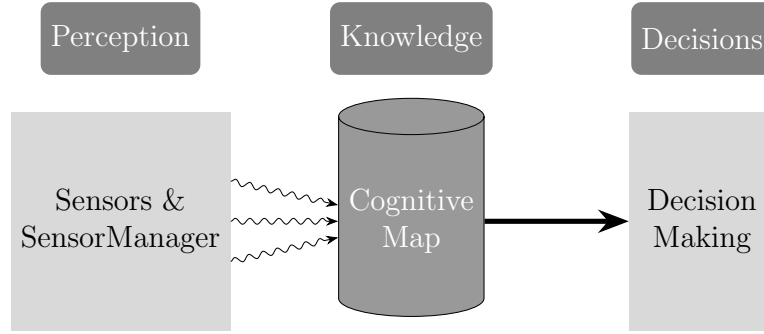


Figure 3.1.: **Modules interaction overview**

Overview of the interaction between the modules.

Tolman [17] which is widely used in robotics navigation. Due to the fact that the navigation is used for agents in evacuation simulations inside a building we omit some parts of the cognitive map concept. A metric map (section 3.1.2) and a memory of used routes (section 3.1.3) constitute our new cognitive map (section 3.1.4). The metric map represents the notion which the simulated pedestrian has about the building whereas the memory of used routes constitutes a simple remembrance.

3.1.1. Requirements and preconditions

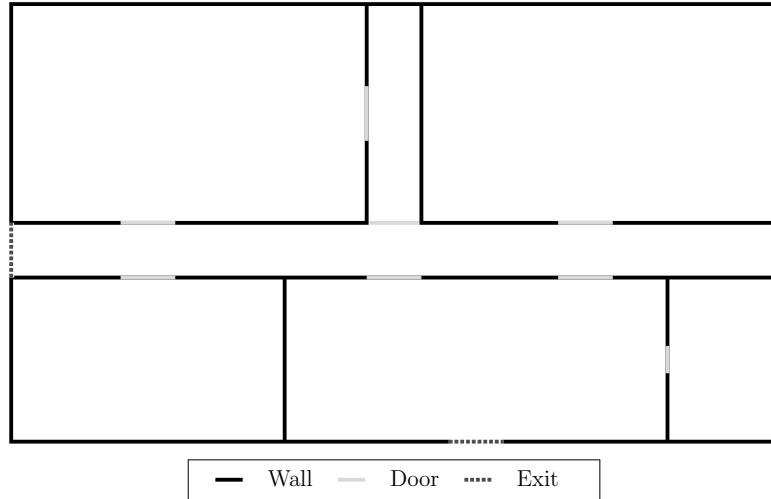


Figure 3.2.: **Floor plan example**

An example of a floor plan used for simulations.

Figure 3.2 shows a simple floor plan with five rooms and two corridors. In the

3. Methodology

simulation software those floor plans have to be converted to another structure. This structure is important.

The simulation software divides the building in several rooms which are divided in further sub rooms. The rooms and sub rooms are bounded by walls, crossings and transitions (for example doors). A crossing is the intersection between two sub rooms among the same room and a transition is the intersection between two different rooms or a room and the outside. Figure 3.3 show the relation diagram of the described geometry structure. Since the difference between transitions and crossings is from less importance for this work we call them door.

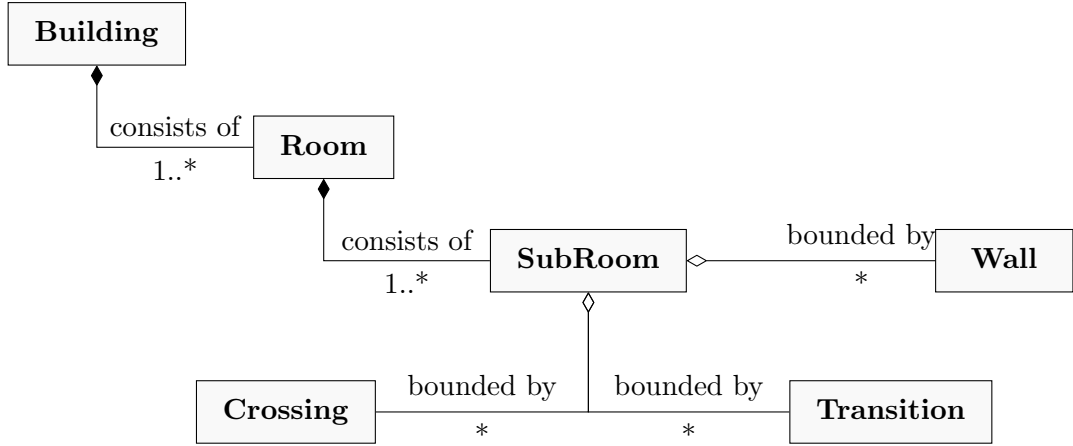


Figure 3.3.: **Geometry structure**

Relation diagram of the geometry structure used in the simulation software.

From the geometry structure we deduced the possible spatial knowledge and classified it into two classes. The basic structure an agent could be aware of are sub rooms and doors. The knowledge of existence of sub rooms and doors are then classified as first order knowledge. The second order knowledge are additional information an agent could have about sub rooms or doors (e.g. the blockage of a door or the smoke emergence in a sub room). With this classification second order knowledge is always related to elements of the first order. For example the awareness of a congestion in front of a door is knowledge of second order whereas the knowledge of the door itself belongs to the first order knowledge. Another advantage is the possibility to store knowledge of second order as a property of an element of first order. Therefore the modeling of the new data structure has to consider the first order knowledge primarily. In table 3.1 some illustrative examples are shown.

First order knowledge item	Corresponding second order knowledge examples
Sub room	Smoke in the sub room. Type of the sub room (e.g. corridor or normal sub room).
Door	The people density in front of the door. A congestion at the door. The blockage of a door.

Table 3.1.: **Knowledge classification examples**

Examples for first and second order spatial knowledge.

Even if our first order knowledge is bound to sub rooms and doors the second order knowledge is really versatile. We just mentioned some examples but a lot of other information would be imaginable. The first order knowledge is a robust spatial representation whereas the second order knowledge gives us the versatility.

3.1.2. The metric map (NavigationGraph)

The metric map constitutes the notion of the building an agent has. Therefore it has to represent the knowledge of first order as well as the knowledge of second order. As aforementioned elements of second order could be modeled as properties of elements of first order. Therefore we have to care about first order elements mostly.

For the first order knowledge we decided to use a graph based structure. That is why we had to identify sub rooms and doors with vertices and edges. In contrast to some former routing algorithm we identify the sub rooms as vertices and the doors as edges.

This is reasonable in order to have a versatile structure for adding information to a certain edge. In our representation an edge represents the intersection between sub rooms, which is needed to guide agents from room to room instead of guiding them from door to door. Another advantage is the possibility to store different information for different edges directions. For example leaving a room towards a corridor is rated better than the other direction. This structure gives us the possibility to have an idea about leaving the sub room in the first place, which would be difficult if doors are vertices. The chosen structure has some downsides as well. When it comes to accurate distance calculations some problems appear.

3. Methodology

Definition 3.1 (Navigation graph):

Let B denote the set of sub rooms of a building. Then we call the directed graph

$$G_i = G(V_i, E_i) \quad (3.1)$$

with vertices

$$V_i \subseteq V := B$$

and edges

$$E_i \subseteq E$$

$$E := \{e = (e_1, e_2) \in B \times B \mid e_1 \neq e_2 \wedge \exists \text{ Intersection between SubRoom}(e_1) \text{ and SubRoom}(e_2)\} \quad (3.2)$$

the **navigation graph** of the pedestrian P_i .

The navigation graph is called **complete** if $G_i = G(V, E)$ otherwise it is called **sparse**.

Lemma 1 The navigation graph is in general not acyclic.

PROOF Figure 3.5 shows a navigation graph constructed from the floor plan in figure 3.4 which includes cycles. \square

Lemma 2 The navigation graph could have multiple edges.

PROOF Given a building \overline{B} which has two different sub rooms which are connected by two different doors.

\Rightarrow The complete navigation graph of \overline{B} has multiple edges between this two sub rooms. Figure 3.4 shows an example building (room 2 and corridor B). \square

Lemma 3 The navigation graph is always loop free.

PROOF Assume we have a loop l .

$\Rightarrow dest(l) = src(l)$

\Rightarrow Contradiction to definition 3.1 (3.2). \square

3. Methodology

It is possible to create a sparse graph to model pedestrians with incomplete knowledge of a building. Figure 3.4 shows the floor plan from figure 3.2 after labeled with vertices and edges. Figure 3.5 shows the finished graph used inside the simulation software after processing the geometry. In this example the navigation graph contains all sub rooms and doors thus it is complete.

3. Methodology

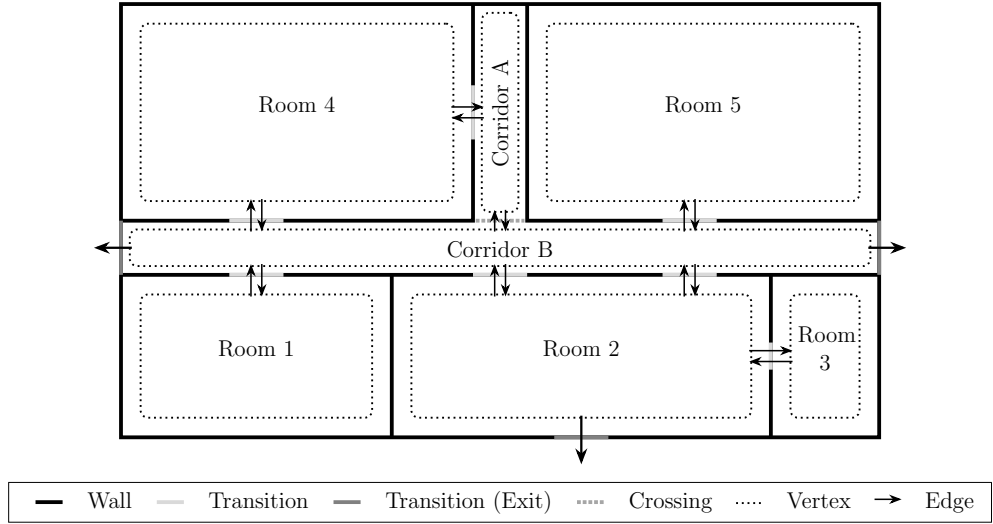


Figure 3.4.: **Floor plan with graph**

The floor plan used in simulations with assigned complete navigation graph structure.

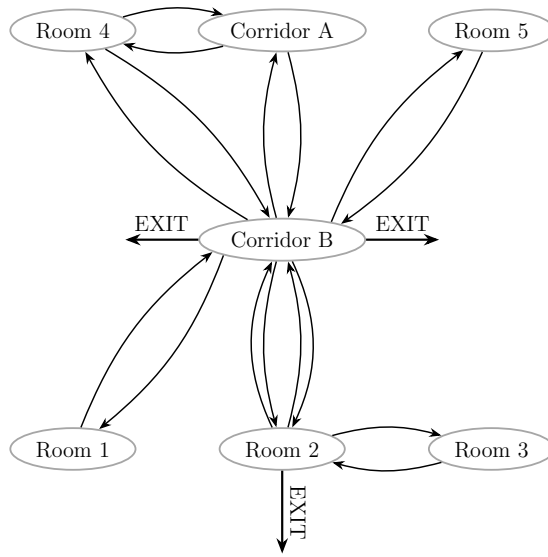


Figure 3.5.: **Graph from floor plan**

The complete navigation graph generated from the floor plan of figure 3.2.

Implementation

Figure 3.6 shows the UML diagram of the navigation graph. The implementation of the navigation graph consists out of three classes the **NavigationGraph**, the **Vertex** and the **Edge** class. The **NavigationGraph** class exists out of a collection of pointers to **Vertex** objects and several method to manipulate or read parts of the graph. The **Vertex** class consists out of a pointer to the corresponding **SubRoom** and a collection of **Edges** starting in this vertex. The **Edge** class has a source and a destination pointer to the corresponding **Vertex** and a pointer to the **Crossing** or **Transition**. The **Edge** has a collection of factors which are used to calculate the weight of an edge. This weight is later used for decision making.

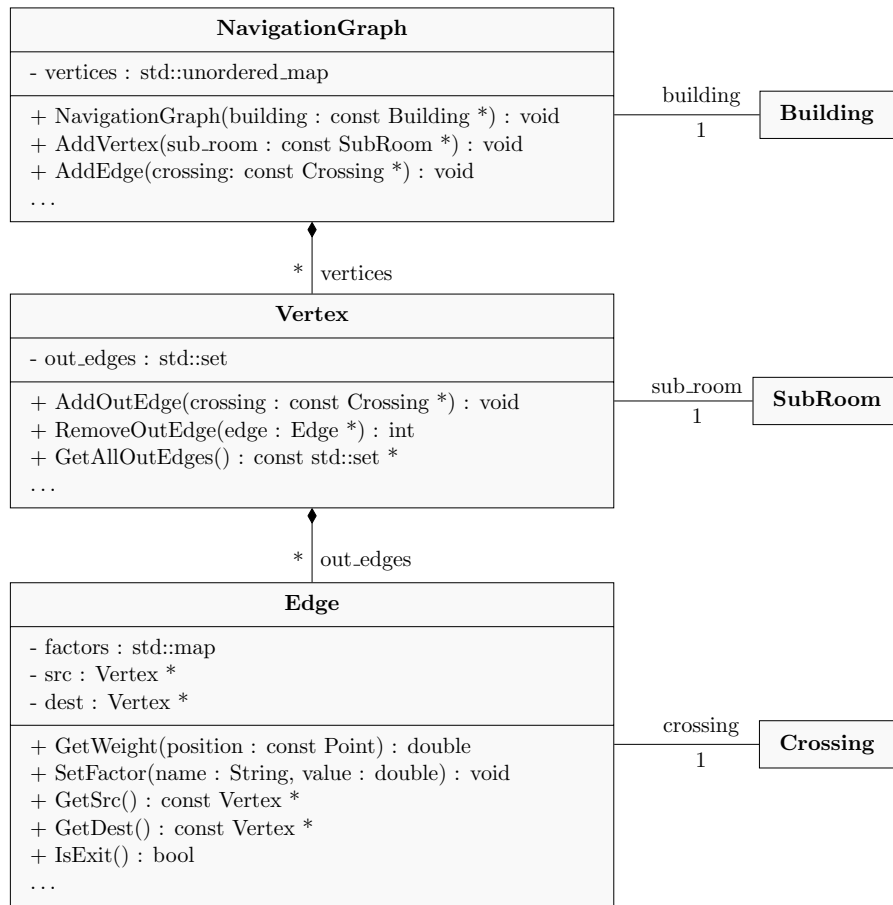


Figure 3.6.: UML diagram of the NavigationGraph

The UML class diagram of the implemented navigation graph.

Edge-weight calculation

The calculation of optimal routes is based on the weight of each edge. These depend primarily on the distance and other factors which represent the second order knowledge.

Definition 3.2 (Edge-factor):

Let $e_i \in E$ be an edge of the navigation graph $G = G(V, E)$.

$$F_i := \{f_k \in F_i \mid f_k \in \mathbb{R} \wedge f_k > 0\} \quad (|F_i| < \infty)$$

is the **set of corresponding factors** for e_i . The elements of F_i are called **edge-factors**.

$$f^{(i)} := \prod_{f_k \in F_i} f_k$$

is called the **accumulated edge-factor**.

Definition 3.3 (Edge-weight):

Let x_i be the length of the edge e_i and F_i the set of corresponding factors for e_i . Then is

$$w_i := x_i \cdot \prod_{f_k \in F_i} f_k = x_i \cdot f^{(i)} \quad (3.3)$$

the **edge-weight** of e_i .

Edge-factors and sensors are highly related to the decision making process. The decision making is based on the edge-weight, to decide for an optimal route. Thus the edge-factors have a high influence on the chosen route.

3.1.3. Used routes memory

The smaller part of the cognitive map is the memory of used routes, which is for the purpose of representing the pedestrians remembrance. We store every edge which was chosen by the decision making in the same order. With this we can reconstruct the chosen path. An application could be a sensors which avoids the agent from going backwards (section 3.2.2).

The implementation of the used routes memory is a simple `std::vector` which stores the edges in the right order.

3.1.4. Putting it all together: the cognitive map

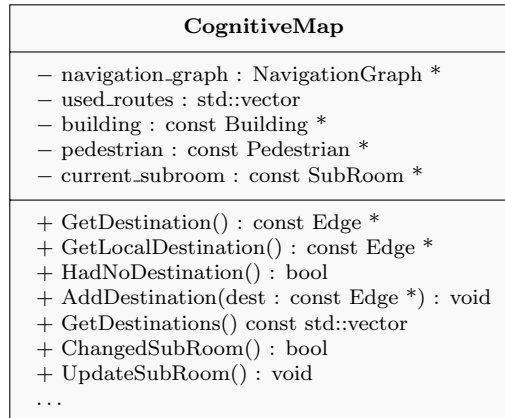


Figure 3.7.: UML diagram of the CognitiveMap

The UML class diagram of the CognitiveMap.

The resulting **CognitiveMap** class (figure 3.7) consists out of the described **NavigationGraph** and the **std::vector** of used **Edges**. Even if this cognitive map is a drastic simplification of Tolmans cognitive map it is less complex and still fits our needs.

In the simulation each agent has its individual cognitive map. These maps are accessed through the **CognitiveMapStorage** class which also takes care of the creation of the initial cognitive maps. The creation itself is done by **CognitiveMapCreator** classes which are passed to the **CognitiveMapStorage** and executed when needed. With these **CognitiveMapCreators** it is possible to create different cognitive maps, in matter of information content, to simulate pedestrians with different knowledge. It is also possible to use different creators for different agents. The current implementation features two creators the **CompleteCognitiveMapCreator** which creates a complete cognitive map and an **EmptyCognitiveMapCreator** which creates an empty cognitive map. Further creators can be easily implemented. Figure 3.8 shows the UML diagram of the creation and storage part.

3.2. Gathering information

WAS KANN ICH WISSEN?

Logik (p. 25)

IMMANUEL KANT

3. Methodology

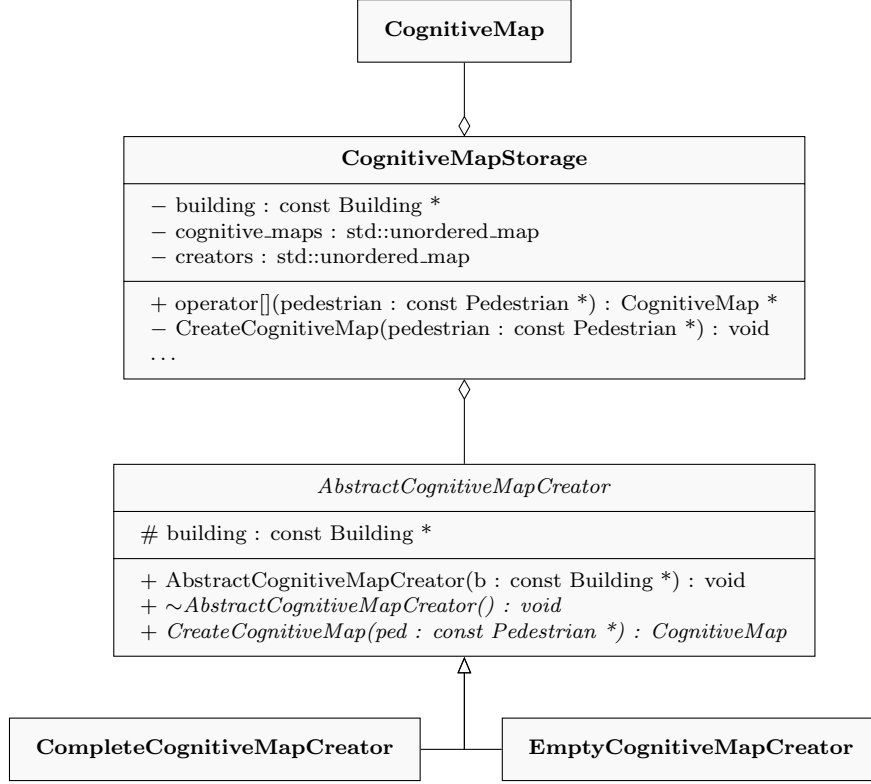


Figure 3.8.: **UML diagram of CognitiveMapStorage**

The UML class diagram of the CognitiveMapStorage and CognitiveMapCreators.

With the proposed cognitive map we have a versatile structure for the later discussed decision making. The decision making is based on the edge weight and thereby on the edge factors. Therefore it is important that the cognitive map in general and the edge factors in particular are up to date to make current decisions.

The information gathering module is responsible for this update process. It provides a framework for reproducing simplified human perception. It is able to manipulate edge factors as well as an entire edge or vertex.

For gathering information a sensor structure was build. These sensors are managed and executed by an event driven sensor manager. Those events could be triggered in every time step during the execution of the router. Therefore the sensors are executed individually for each agent.

3.2.1. Sensors and sensor manager

The sensor system consists of **Sensors** and the **SensorManager** class.

SensorManager and process

The sensor system was build with the observer design pattern in mind. The **SensorManager** is the observing object which observes the **CognitiveMapRouter** object. An object inherited from **AbstractSensor** could be registered to the **SensorManager** for certain events. When the **CognitiveMapRouter** reaches an event state it notifies the **SensorManager**, which then executes the **Sensors** registered for this event.

Table 3.2 shows the available events. The integration of further events is possible.

Event name	Description
INIT	This event is triggered during the initialization of the cognitive map of the respective agent.
CHANGEDROOM	This event is triggered when an agent changed the sub room.
NEWDESTINATION	This event is triggered after the agent got a new destination from the router.
NOWAY	This event is triggered when the agent could not find a way to a known emergency exit.

Table 3.2.: **Available events**

Available events for registering **Sensors** to the **SensorManager**.

With the **SensorManager** it is possible to register **Sensors** as needed. So its the task of the user to decide which sensors should be activated and used on which event. It is not even required to use one of them. With this, the sensor execution chain is completely adjustable to the needs of the actual simulation.

Sensors

A **Sensor** is a class inherited from **AbstractSensor**. The task of the sensor is the manipulation and creation of first and second order knowledge of the cognitive map. Therefor it could read from the **CognitiveMap**, the **Building** and the corresponding **Pedestrian** object. The **Sensor** could be thought as both, a modeled sensing device or a information manipulator without any physical sensing in mind. It is the main input for the cognitive map. It could add or delete edges and vertices or manipulate edge factors.

3. Methodology

With the versatile sensors it is possible to add any information to the cognitive map and thereby change the decision making. For example one could add data from a fire simulation to have a better smoke status of a room. This smoke data could then be used to set a defined edge factor to edges heading to the smoked sub room. This will lead to a decision in favour of non smoked sub rooms. So the sensor could be the interface for getting more data into the routing and decision making process.

Implementing a sensor

A new sensor has to inherit from the abstract class **AbstractSensor**. The parent class specifies the implementation of an execute function and a get name function. Furthermore a constructor function is inherited which sets a pointer to the **Building** object. The **GetName** function is mostly used for storing factors and sensors identified by name. The **execute** function is the main function of the sensor. This function is executed when reaching an event for which the sensor is registered. In this function the **CognitiveMap**, **Building** and **Pedestrian** object could be used to manipulate the **CognitiveMap**.

An implemented sensor could then be registered to the **SensorManager**.

3.2.2. Implemented sensors

The results gained by the different sensors are discussed in section 4.2. In this section the implemented sensors are presented.

RoomToCorridorSensor

The **RoomToCorridorSensor** adds an edge-factor to the respective edge depending on the source and destination sub room type. It cares about sub rooms of the type room or corridor. Based on the assumption, that changing a room in direction of an exit corridor is good and leaving an exit corridor in the direction of a usual room is bad, the edge-factor is lower or higher than one. This edge-factor makes agents tend to go in the direction of a corridor or to stay on corridors.

LastDestinationsSensor

The **LastDestinationsSensor** is based on the used routes memory. It sets a penalty edge-factor (> 1) to the corresponding edge in opposite direction if it exists. This way an agent is hindered from going back immediately. However raising the edge

factor only means that the chance of going back is minimal but not zero, since the edge still exists in the navigation graph. The sensor sets the edge-factor which raises the weight but does not delete any edge. If an agent has discovered all possible directions he could decide to go back again if all other edges have even worse weights (for more details on decision making see section 3.3).

DiscoverDoorsSensor

The **DiscoverDoorsSensor** should be used after the agent was not able to find any route to an emergency exit. It emulates the process of discovering an unknown room. This sensor adds all possible out edges to the vertex corresponding to the actual sub room. After the sensor execution the agent can at least start searching for a local route.

SmokeSensor

The **SmokeSensor** sets a smoke edge-factor. This edge-factor should consider the smoke status or fire hazard in a certain sub room. If the smoke status of a sub room is activated, the sensor raises the smoke edge-factor of all edges heading to this sub room. In this way the agent is encouraged to avoid the smoked sub room.

DensitySensor

The **DensitySensor** measures the density in front of a door (crossing or transition) and sets the corresponding edge-factor. If the density is too high the agent tends to take another route and avoid the jam.

3.3. Making decisions

WAS SOLL ICH THUN?

Logik (p.25)

IMMANUEL KANT

The last module of the proposed routing framework is the decisions making module. With the already defined edge-factors (see definition 3.2) and the deduced edge-weight (see definition 3.3) it was nearby to calculate optimal routes in the given navigation graph. But due to the fact, that some agents may have a sparse

navigation graph it is possible that an agent does not know any complete exit route. There for we distinguish between agents with enough knowledge to find a complete exit route and agents with less knowledge. The first group uses a global optimization algorithm and the second uses a local algorithm. Till now there is a strong separation of strategies of this two groups, but for more realistic behavior a combination of both strategies would be advisable.

3.3.1. Global way finding

The navigation graph is a directed graph with strictly positive weights. It is not necessarily acyclic (lemma 1). For the global way finding we use a modification of dijkstras shortest path algorithm [5].

The major problem was the distance calculation for the edges. Due to the structure of the navigation graph the representation of an edge is the corresponding door which has a spatial extent of zero, at least in the simulation. Therefore another definition of the edge distance is introduced.

Distance calculation

The idea behind the graph structure was that an agent navigates from sub room to sub room until the goal is reached. The navigation uses the graph edges for navigating the agent to the next sub room. The edge could be imagined as an undefined path through the sub room to the intersection with the next sub room. Its the distance an agent needs to go in order to leave the sub room heading to the next one. This weakens the geometric imagination, because we do not have an exact edge position and distance anymore, but serves well for calculating optimal paths.

For the calculation that means that we have to consider the actual position of the agent in the sub room. For the starting room this is the obvious offset distance, the distance from the pedestrian to the intersection. But for the further sub rooms the position of the agent is not obvious.

Our first idea to solve this problem was an approximation of the distance needed to leave the sub room. One approximation was the average distance from the center of the room to all available intersections. The other one was the average distance from all intersections to all other intersections. But the major problem with these approximations was, that they over and under estimate the exact distance at the same time. Depending on the actual geometry this could lead to unintended behavior

3. Methodology

since we use the distance for the edge-weight calculation.

Instead of searching for a better approximation we decided to calculate the exact distance an agent would travel. For this the preceding sub room and the used intersection is needed to have an expected position of the agent when reaching the sub room. Unfortunately this causes another problem.

Due to the fact that our edge distance depends on the expected position of the agent the edge distance differs. This effects the total distance calculation and comparability. For example the path A which had higher costs than the path B when reaching the sub room S_k could have lower costs than B when leaving the sub room heading to another sub room S_{k+1} . This happens when the edge connecting A to S_{k+1} is cheaper than the edge connecting B with S_{k+1} . So not only the shortest distance to a sub room is important, but rather the shortest distance to the intersection with the sub room.

In the graph structure this means, that the concrete distance of a certain edge in a analysed path depends on the predecessor and the successor edge.

Excursus: Dijkstra algorithm

In his seminal article from 1959 [5] Dijkstra proposed two algorithms. One for calculating a minimum spanning tree and one for calculating the minimal path between two vertices in a graph $G = G(V, E)$. We use the well known shortest path algorithm.

To calculate the shortest path between two vertices P and Q the algorithm scans vertices starting from P . During the initialization the total distance is set to infinity for each vertex. Now the total distance of P is set to zero and P is added to the queue.

In each iteration the vertex R with minimal total distance is taken from the queue. If the new vertex is Q the shortest path has been found. Otherwise the total distance of all vertices adjacent to R is compared with the total distance via R . If the new path is shorter the total distance is set to the total distance via R and the preceding vertex is set to R . In addition all adjacent vertices which are not already in the queue are added to the queue.

This progresses till Q is reached. If the queue is empty and Q was not reached there is no path from P to Q .

Modification of Dijkstra algorithm

One of our major needs for calculating the exact edge distance starting in a sub room was the knowledge of the expected position of the agent when reaching this sub room.

In the algorithm this could be deduced from the chosen edge between the actual vertex and the preceding vertex. The chosen edge is important anyway because we could have multiple edges (see lemma 2). In addition we need to consider the different distance of a single edge depending on the preceding edge. That is why we have to store the total distance for a vertex depending on the preceding edge. Since the vertex is determined by the edge destination we could exchange vertices and edges at all.

Therefore our algorithm does not discover new vertices but discovers new edges. At the end one could identify the edge with the edge destination vertex.

In addition we do not calculate the shortest path to a certain vertex but the shortest path to one edge in a set of exit edges E_E (emergency exits). This changes the abortion condition of the algorithm.

Modified Dijkstra algorithm

Let G denote a graph with dynamic weight edges depending on the position of the agent in the sub room (vertex). For calculating the cheapest path between a vertex P and an exit edge in E_E in the graph G the algorithm visits combinations of edges and vertices beginning with edges incident with P .

The initialization sets the total weight sum to infinity for each edge. Now the total edge weight sum of edges incident with P is set to the weight calculated with the distance from P to the corresponding intersection. All edges starting in P are added to the queue.

In each iteration the edge e_k with minimal total weight sum is taken from the queue. If the new edge is element of the exit edge set E_E the shortest path has been found.

Otherwise we iterate over the edges incident with the destination vertex $v^{dest}(e_k)$ of e_k . For each edge \bar{e}_i incident with $v^{dest}(e_k)$ the total edge weight sum is compared. Therefor the edge weight of \bar{e}_i is calculated with the distance from the intersection corresponding to e_k and the intersection of \bar{e}_i and the edge-factors of \bar{e}_i . If the total edge weight sum via e_k is less than the total edge weight sum stored in \bar{e}_i the sum

3. Methodology

and the preceding edge are changed. The edge \bar{e}_i will be added to the queue if it was not already added before.

This progresses till an edge out of E_E is reached. If the queue is empty and no edge of E_E was reached there is no path from P to E_E .

The algorithm is similar to the original algorithm applied to the line graph of an undirected graph G . The major difference is the calculation of the edge weights. The distance is calculated from the line graph edge but the edge factors are taken from the original edges.

Listing A.3 shows the commented implementation of the modified algorithm in

C++. Listing 3.1 shows the pseudo code of the modified dijkstra algorithm.

Code listing

Data: Position of the pedestrian, current vertex v

Result: A destination for the pedestrian.

initialization

```

while  $\exists e \in out\_edges(v)$  do
    calculate distance of  $e$ 
     $destinations\_map[] \leftarrow e$ 
     $queue[] \leftarrow e$ 
while  $queue \neq \emptyset$  do
     $e_{actual} \leftarrow queue.top$ 
     $d_{actual} \leftarrow d_{total}(e_{actual})$ 
    if  $e_{actual} == emergency\ exit$  then
        return path to  $e_{actual}$ 
     $v_{actual} \leftarrow destination\_vertex(e_{actual})$ 
    while  $\exists e_{out} \in out\_edges(v_{actual})$  do
        if  $v_{out} \in visited\_edges$  then
            continue
        calculate  $d_{total}(e_{out})$  via  $e_{actual}$ 
        if  $e_{out}$  was not discovered before then
             $queue[] \leftarrow e_{out}$ 
        if  $d_{total}(e_{out}) < e_{out}.distance$  then
             $e_{out}.distance \leftarrow d_{total}(e_{out})$ 
             $e_{out}.predecessor \leftarrow e_{actual}$ 
     $visited\_edges[] \leftarrow e_{actual}$ 

```

Listing 3.1: Modified Dijkstra algorithm (pseudo code)

Used algorithm for shortest path calculation.

3.3.2. Local way finding

The local way finding chooses an favourable destination out of the edges intersecting with the vertex corresponding to the actual sub room. It expects at least one known intersecting edge, otherwise the cognitive map could not propose any destination. The edges could be set by an invocation of sensors prior the local way finding.

For the selection of an edge the local way finding first compares the accumulated edge factors of all intersecting edges. (The accumulated edge factor is the product of the edge factors and is independent from the distance. See definition 3.2.) After this double the smallest accumulated edge factor is taken as the upper bound. All edges with an accumulated edge factor lower than this upper bound are taken into consideration. The edge with the shortest distance is chosen as the new destination. The local algorithm tries to use all information a pedestrian has. But at the end the distance is an important decision criteria after selecting the edges with the best accumulated factors. This should reflect a pedestrian choosing the next destination based on a bigger information basis than just the distance. For example jams in front of a door or smoke in the next sub room, could be reflected in an edge-factor and is important for the local route choice.

3.4. Routing the pedestrians

The cognitive map router encapsulates the routing process in the pedestrian dynamics simulation. The router is invoked in every time step for every agent separately. Basing on the described modules above a cognitive map router was implemented. The main function call is the `FindExit` method required by the `Router` interface. This method is invoked in every time step and gets passed a pointer to the certain `Pedestrian` object. Figure 3.9 shows the UML activity diagram of our implementation and listing A.1 shows the actual implementation in `C++`.

As first we check if the pedestrian ever had a destination before. If he did not had a destination before we invoke the sensors registered for the `INIT` event. Those sensor could set initial values like previous knowledge or starting conditions.

After this step we check if the pedestrian changed the sub room. This is realized with the cognitive map, which also cares about the actual and old position of a pedestrian. The changed sub room check is also true for people without a destination. Therefore it is effectively checking if the pedestrian needs a destination.

If the pedestrian changed the sub room or needs a new destination by other reason

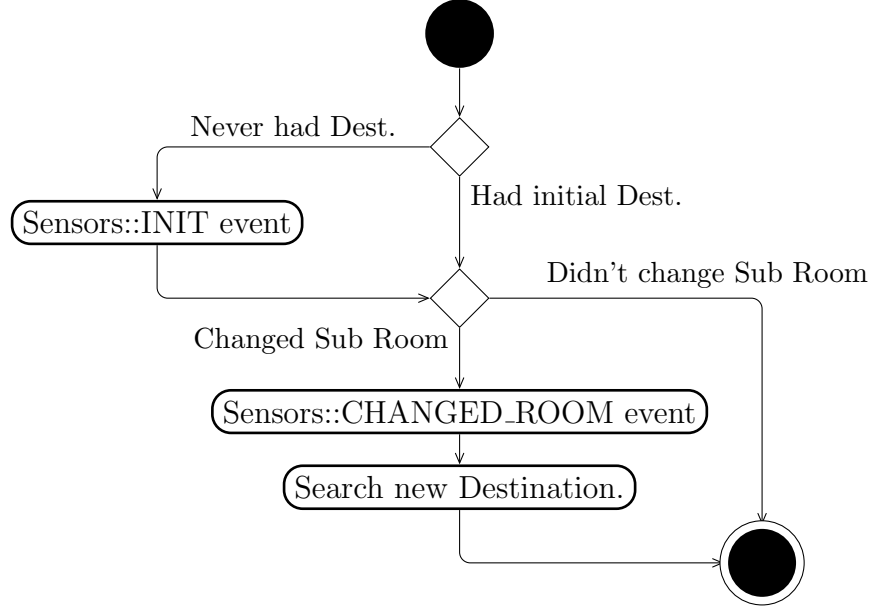


Figure 3.9.: **Router FindExit()** activity diagram

The UML activity diagram for the CognitiveMapRouters FindExit method. It describes the main routing algorithm used in this router. The mentioned sensor events are executed. The searching for a new destination is described separately.

we invoke the sensors with the certain event and start the destination search.

The search for a new destination is done in a separate method called **Find-Destination**. Figure 3.10 shows the UML activity diagram whereas listing A.2 show the actual implementation in C++.

For the retrieval of a new destination the global way finding is invoked as first (see section 3.3.1 for more details on the global way finding algorithm). If the global way finding could not find a route to an emergency exit the sensors registered for the **NO_WAY** event are invoked. Those sensors could add new information to the cognitive map. For this reason the global way finding is invoked again to check if the pedestrian knows a route. If the pedestrian is not successful again the local way finding is called finally (see section 3.3.2).

At the end we check if we got any destination from this procedure. In case we did not get any destination we could not lead the pedestrian to any goal. That is why we print out an error message and the pedestrian is deleted from the simulation later. Usually there should be a sensor which discovers the actual sub room of the pedestrian. At least the local way finding should offer a valid destination.

3. Methodology

If a destination was found the sensors registered for the `NEW_DESTINATION` event are called. As last the pedestrian is update to head the new destination.

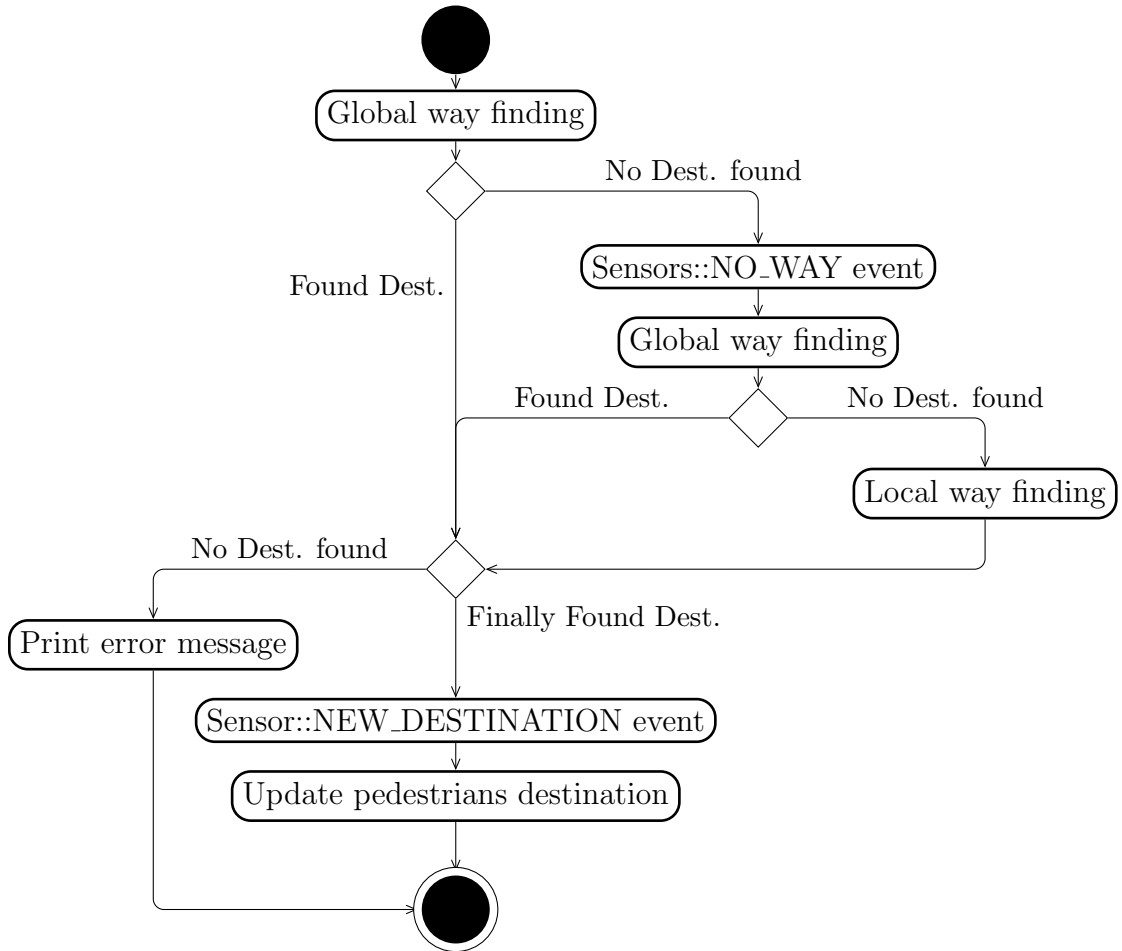


Figure 3.10.: **Router FindDestination()** activity diagram

The UML activity diagram for the CognitiveMapRouters **Find-Destination** method. This method is called for searching a new destination. The decision whether to search a new destination or not is done before. The mentioned sensor events are executed.

SIMULATIONS AND RESULTS

WAS DARF ICH HOFFEN?

Logik (p. 25)

IMMANUEL KANT

To show the flexibility of our new routing framework we analyze the impact of different configurations on the route choice and thus on the whole simulation. The implemented routing framework is adjustable in many different ways. This leads to a lot of possible configurations, which we can not analyze in all aspects. That is why we choose some configurations for the analysis.

The first option is the choice of the cognitive map creators and their distribution. This option determines how the initial cognitive maps are built and look like. The available cognitive map creators are the `CompleteCognitiveMapCreator` and the `EmptyCognitiveMapCreator` (see section 3.1.4 for more details). The second option is the selection of sensors (see section 3.2.1 for more details). The sensors have to be registered and thus change the modeled perception. In addition every sensor has some parameters which could be calibrated too.

In order to achieve comprehensible results we analyze sensors separately in small buildings. We combine different sensor selection with different cognitive map creators. Table 4.1 shows the conducted simulations and analysis for the different sensor-creator combinations. As first we study the impact of single sensors when used with complete cognitive maps. After this we analyze sensors with empty cognitive maps.

Cognitive map creator	Sensors	Section	
CompleteCognitiveMapCreator		4.2	(p. 34)
	No Sensor	4.2.1	(p. 35)
	LastDestinationsSensor	4.2.1	(p. 35)
	RoomToCorridorSensor	4.2.2	(p. 38)
	DensitySensor	4.2.3	(p. 40)
	SmokeSensor	4.2.4	(p. 45)
EmptyCognitiveMapCreator		4.3	(p. 47)
	No Sensor	4.3.1	(p. 47)
	RoomToCorridorSensor	4.3.2	(p. 50)
	SmokeSensor	4.3.3	(p. 52)

Table 4.1.: **Simulation configurations overview**

Overview of the conducted simulations and their configurations. See the mentioned sections for the result of the analysis.

4.1. Setup and preparation

For the analysis we conduct 50 simulations for each configuration of interest with different initial conditions. The initial conditions influence mainly the position of pedestrians in a certain sub room. The initial conditions are different among the 50 simulations of one configuration but equal for different configurations. With this setup we want to minimize the influence of random effects.

For some configurations we compare the total evacuation time with the total evacuation time obtained using the global shortest path router [8]. The total evacuation time is the duration until the last agent has left the building. Furthermore we qualitatively analyze some simulations by inspecting selected snapshots. In those snapshots the pedestrians are colored from red (slow) to green (fast) depending on their current speed.

For comparing evacuation times we use Welch’s t-test [18]. This test has the null hypothesis that the expected values of the distributions of the two samples are the same using the mean value. Rejecting this null hypothesis means that the expected values are significantly different. All tests are calculated with R’s [14] intrinsic t-test method.

4.1.1. Geometries

The main geometry which is used for the analysis is relatively simple to keep track of particular effects. The geometry is shown in figure 4.1. The corridor as well as the emergency exits have a width of 2 meters to avoid congestions.

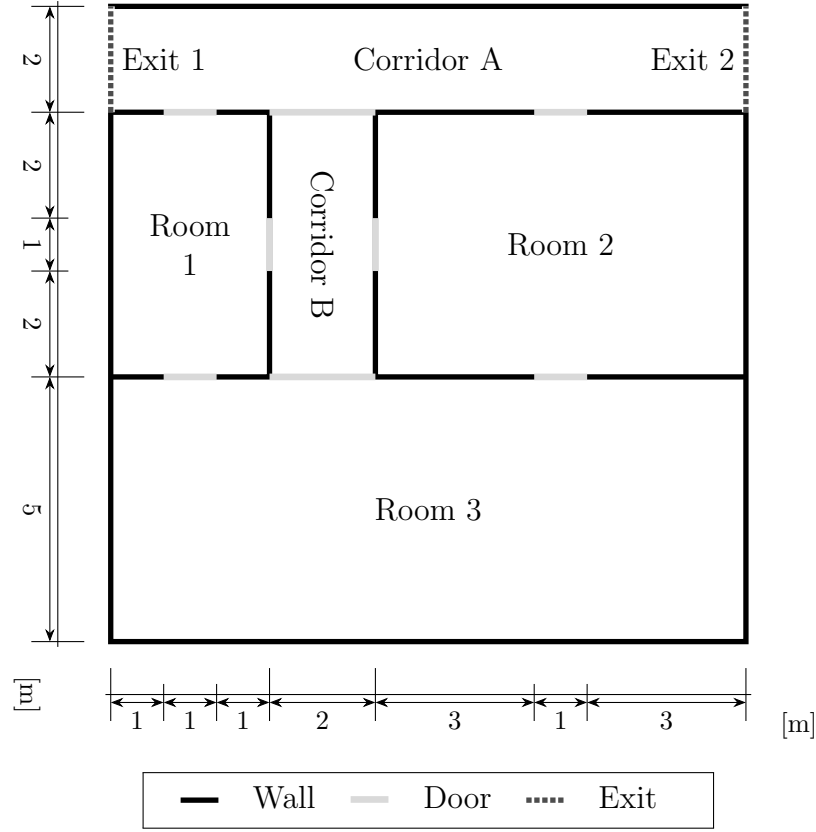


Figure 4.1.: **Geometry for analysis**

A simple geometry used for the analysis of sensors.

For some configurations we use other geometries to point out sensor specific effects.

4.2. Analyzing different sensors with complete cognitive maps

In this section we use the `CompleteCognitiveMapCreator` only. This means that all agents have a complete initial cognitive map and thus know every sub room and door in the building. The only additional information which could be added is

knowledge of second order. The sensors of interest are therefore the `DensitySensor`, the `LastDestinationsSensor`, the `RoomToCorridorSensor` and the `SmokeSensor`. Since the agents have a complete cognitive map the global way finding is used only. We distributed 24 pedestrians in total, two agents in each corridor, five agents in room 1 and 2 each and 15 agents in room 3 (figure 4.1).

4.2.1. LastDestinationSensor and no sensors

The simulations without any sensor is similar to the global shortest path router and hence used as a comparison group for later simulations. The same applies for the `LastDestinationSensor` when used with a complete cognitive map. The total evacuation times of the 50 simulations are shown in table 4.2. The four snapshots in figure 4.2 show, that the pedestrians go straight the shortest path to one of the exits.

Due to the fact that the main floor is 2 meters wide there is no real congestion at any time in front of the emergency exits. That explains why the total evacuation is nearly minimal in this simulations. This has influences on the possible effects of the discussed sensors which could not lower the evacuation time anymore.

t [s]	Frequency	Percentage
16	2	4%
17	19	38%
18	14	28%
19	12	24%
20	1	2%
21	2	4%

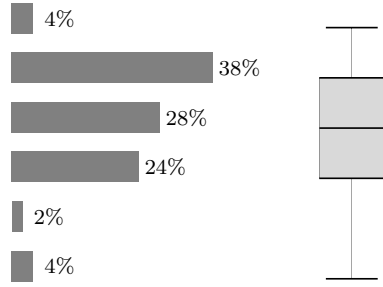
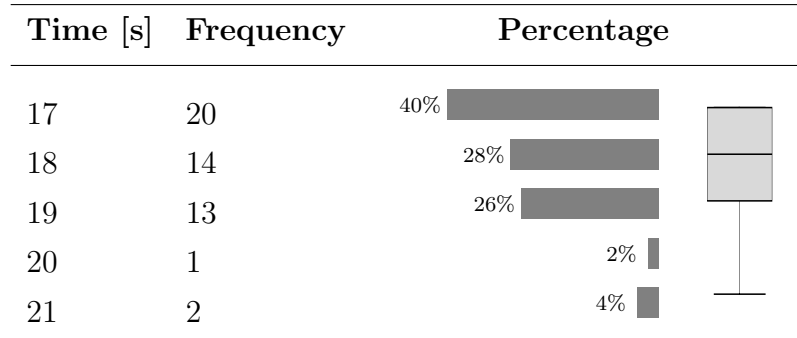


Table 4.2.: **Complete cognitive map no sensors**

Evacuation time frequencies of 50 simulations with `CompleteCognitive-MapCreator` and no sensors.

Table 4.3.: **Complete cognitive map with sensors**

Evacuation time frequencies of 50 simulations with `CompleteCognitive-MapCreator` and `LastDestinationSensor`.

4. Simulations and results

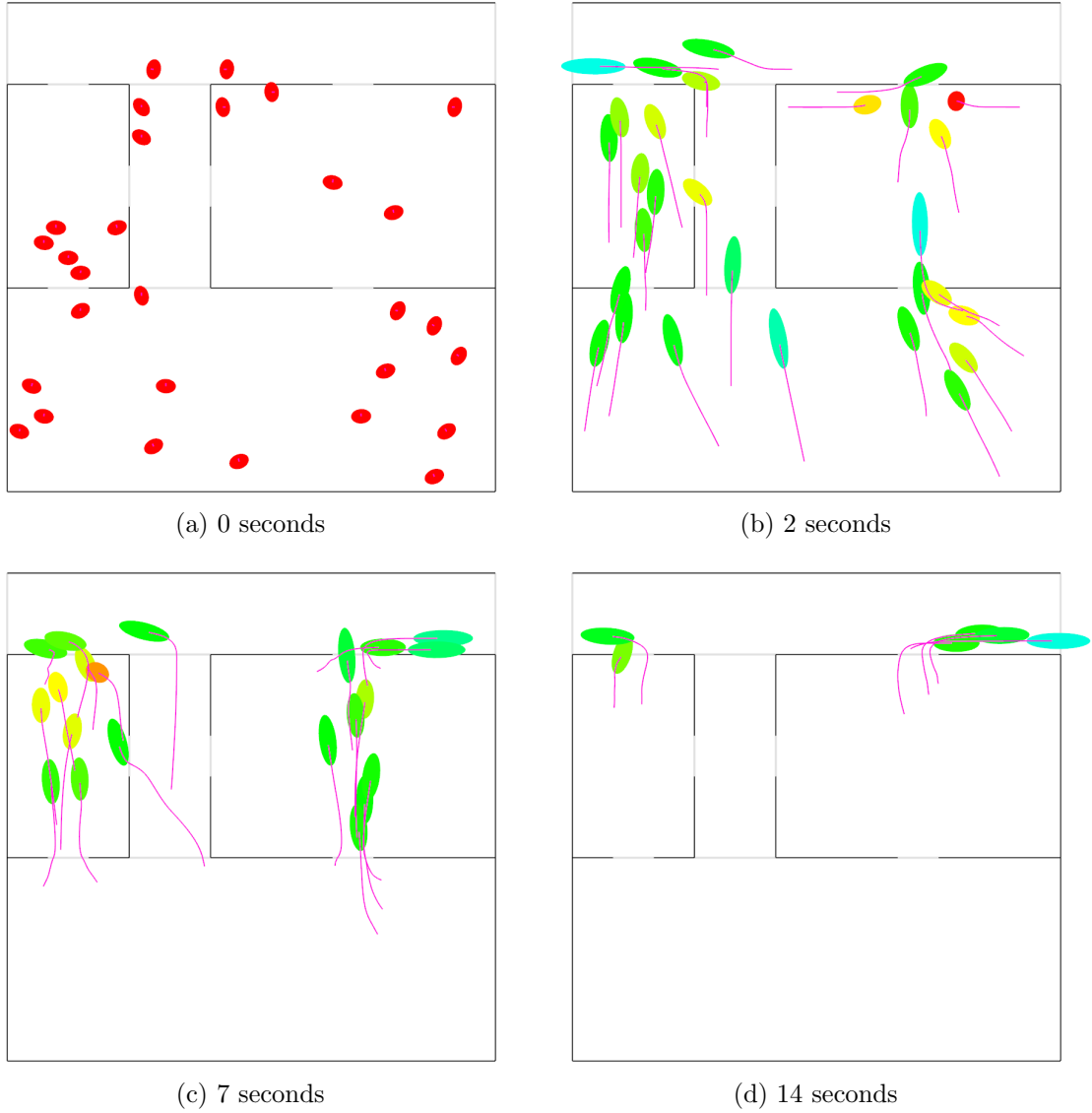


Figure 4.2.: **Simulation with complete cognitive map**

Simulation with complete cognitive map and no sensors.

4.2.2. RoomToCorridorSensor

As aforementioned the `RoomToCorridorSensor` makes the agents tend to go in the direction of a corridor or to stay in corridors. Thus some agents have to go a longer distance to reach the exits and one expects higher evacuation times. Table 4.4 shows the total evacuation time of both configurations, with `RoomToCorridorSensor` and without. The obvious difference of the total evacuation times is verified by a p-value of $2.2 \cdot 10^{-16}$ in Welch's t-test. The total evacuation time with `RoomToCorridorSensor` is significant higher than without.

The snapshots in figure 4.3 illustrate the expected behavior. Most of the agents prefer to head to the next corridor directly even if the distance is slightly higher. Only some agents which are located pretty near the doors go to the next normal sub room.

t [s]	Freq.	With sensor	Without sensor	Freq.
16	0	0%	4%	2
17	0	0%	38%	19
18	0	0%	28%	14
19	6	12%	24%	12
20	8	16%	2%	1
21	12	24%	4%	2
22	12	24%	0%	0
23	9	18%	0%	0
24	3	6%	0%	0
<p>p-value $< 2.2 \cdot 10^{-16}$ (Result of Welch's t-test)</p> <p>mean (Total evac. time): 21.38s 17.94s</p>				

Table 4.4.: **Complete cognitive map with sensors**

Comparison of total evacuation time frequencies of simulations with `RoomToCorridorSensor` and without sensors. We conducted 50 simulations each. The simulations are all done with complete cognitive maps.

4. Simulations and results

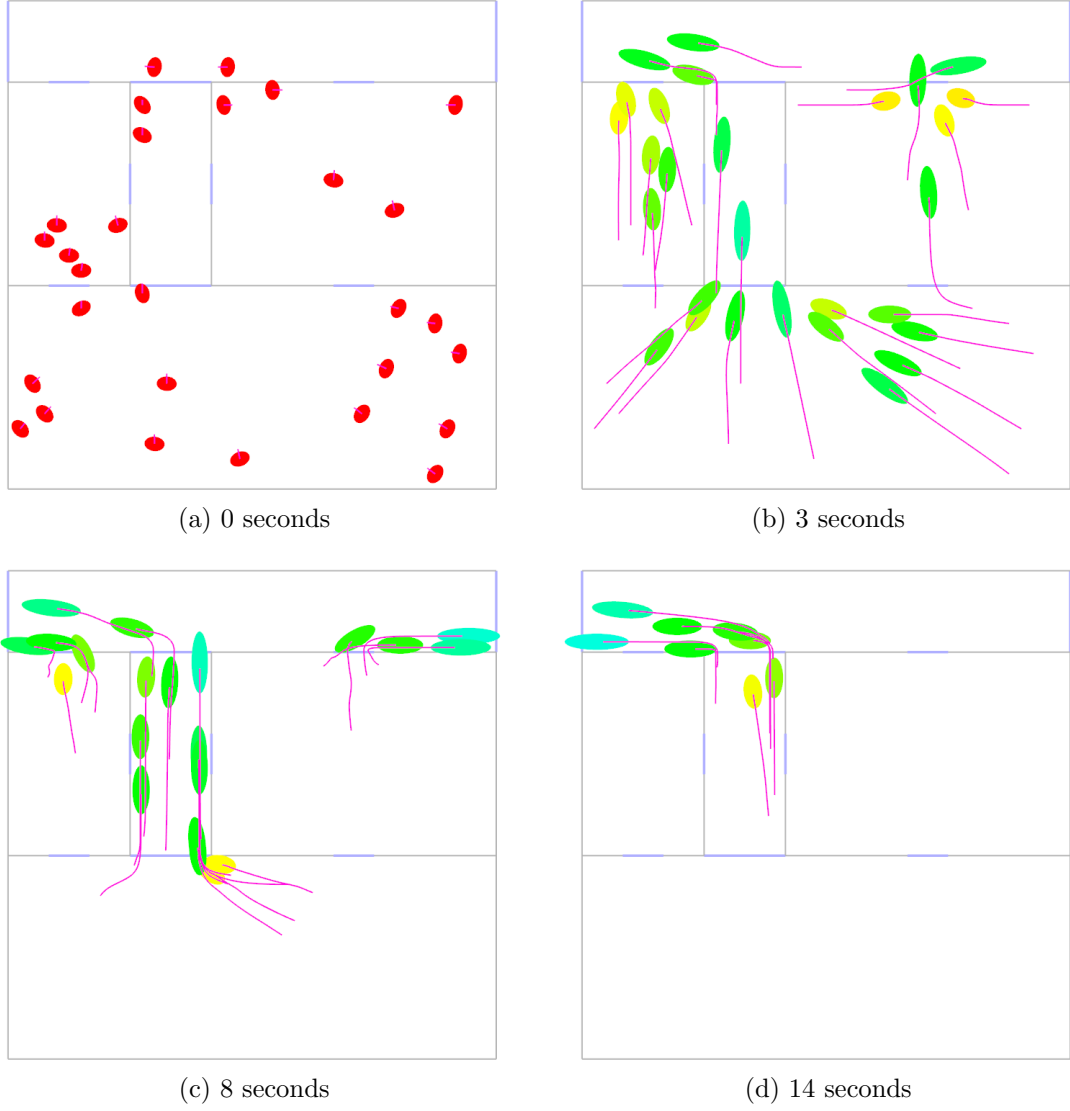


Figure 4.3.: **Simulation with complete cognitive map**

Simulation with complete cognitive map and `RoomToCorridorSensor`. The agents tend to stay on a corridor or change the room in the direction of a corridor.

4.2.3. DensitySensor

The **DensitySensor** is responsible for avoiding congestions. It measures the density in front of doors and rates the door appropriately. The resulting evacuation times are shown in table 4.5. The total evacuation time is significantly higher than the total evacuation time of simulations without any sensor with a p-value of $2.373 \cdot 10^{-8}$. The reason for the higher evacuation time is the absence of any congestion in front of exits during the simulations. Which means, that the distributing of agents balanced by the density in front of doors doesn't improve the evacuation time in this special case. However this does not mean that pedestrians in real live do not behave in this way. In addition the sensor causes new congestions as seen in figure 4.4 at 9 seconds between room one and corridor b (figure 4.1) caused by the bidirectional flow of a few agents. This lead to longer evacuation duration too.






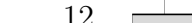













t [s]	Freq.	With sensor	Without sensor	Freq.
16	0	0% 	4% 	2
17	2 	4% 	38% 	19
18	12 	24% 	28% 	14
19	12 	24% 	24% 	12
20	17 	34% 	2% 	1
21	5 	10% 	4% 	2
22	2	4% 	0% 	0
<p>p-value $< 2.373 \cdot 10^{-8}$ (Result of Welch's t-test)</p> <p>mean (Total Evac. Time): 19.34s 17.94s</p>				

Table 4.5.: **Complete cognitive map with sensors**

Comparison of total evacuation time frequencies of simulations with **DensitySensor** and without sensors. We conducted 50 simulations each. The simulations are all done with complete cognitive maps. At 9 seconds evacuation time a congestion between room 1 and corridor b occurs due to the bidirectional flow at the door.

4. Simulations and results

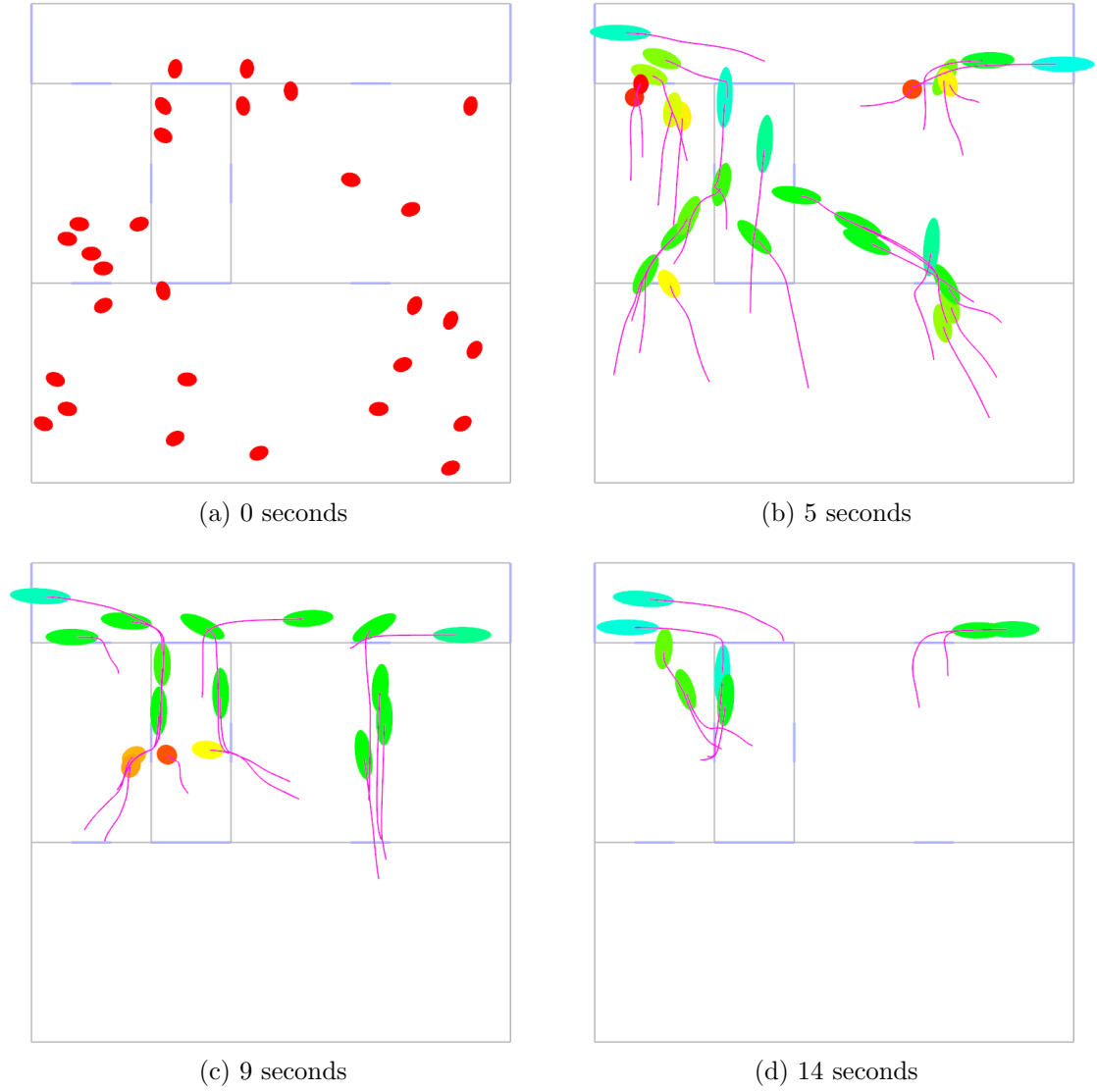


Figure 4.4.: **Simulation with complete cognitive map**

Simulation with complete cognitive map and `DensitySensor`. The agents try to avoid crowded doors.

4. Simulations and results

For the analysis of the **DensitySensor** we conducted further simulations with the geometry shown in figure 4.5. Exit 1 is smaller than exit 2 and should cause some congestions. In addition the way from room 1 to exit 1 is shorter than the way to exit 2. We distributed 140 agents in room 1. Without any sensors the agents take the direct path to the exit 1. This leads to a high density and a congestion in front of the emergency exit.

With the **DensitySensor** the exit 2 is used too and the agents are distributed better. Depending on the actual density when arriving at corridor A the agents decides whether to go to exit 1 or to exit 2. The total evacuation time is significant lower with the **DensitySensor** than without. Table 4.6 shows the comparison of the total evacuation times.

Figure 4.6 shows some snapshot of a simulation without the **DensitySensor**. The agents are using exit 1 (figure 4.5) only. Figure 4.7 shows the same simulation with equal initial conditions but with the **DensitySensor**.

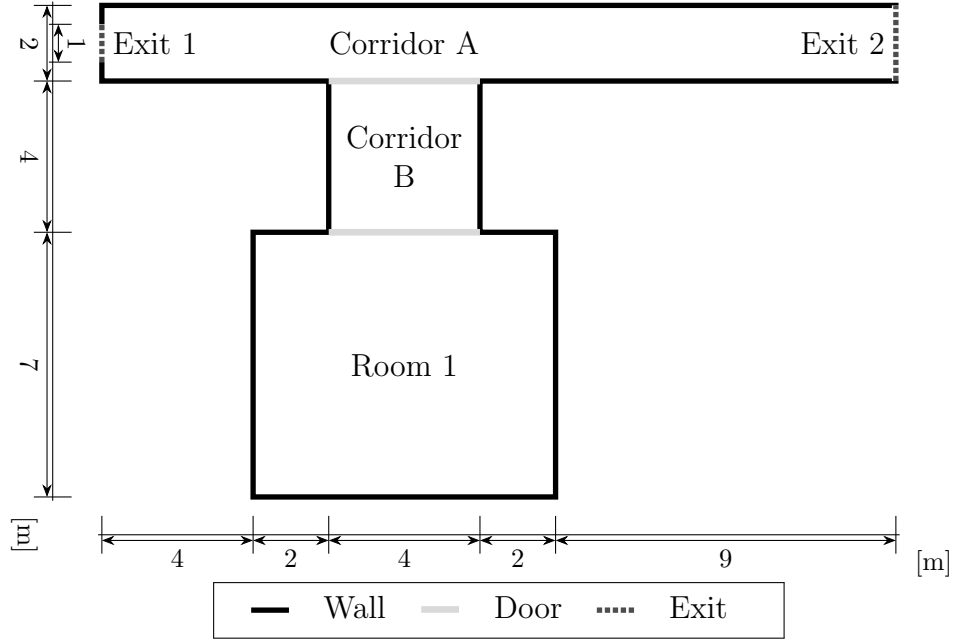


Figure 4.5.: **Geometry for analysis of DensitySensor**

The geometry (T-Junction) used for analyzing the **DensitySensor**. The distance to exit 1 is shorter than the distance to exit 2 but the estimated travel time could be higher depending on the density.

4. Simulations and results
















t [s]	Freq.	With sensor	Without sensor	Freq.
59	4	8% 	0%	0
60	9	18% 	0%	0
61	8	16% 	0%	0
62	9	18% 	0%	0
63	11	22% 	0%	0
64	7	14% 	0%	0
65	2	4% 	0%	0
...
81	0	0%	2% 	1
82	0	0%	2% 	1
83	0	0%	10% 	5
84	0	0%	28% 	14
85	0	0%	32% 	16
86	0	0%	16% 	8
87	0	0%	8% 	4
88	0	0%	2% 	1
<p>p-value $< 2.2 \cdot 10^{-16}$ (Result of Welch's t-test)</p> <p>mean (Total evac. time): 61.86s 84.76s</p>				

Table 4.6.: **Analyzing the DensitySensor**

Comparison of total evacuation time frequencies of simulations with **DensitySensor** and without sensors. We conducted 50 simulations each. The simulations are all done with complete cognitive maps.

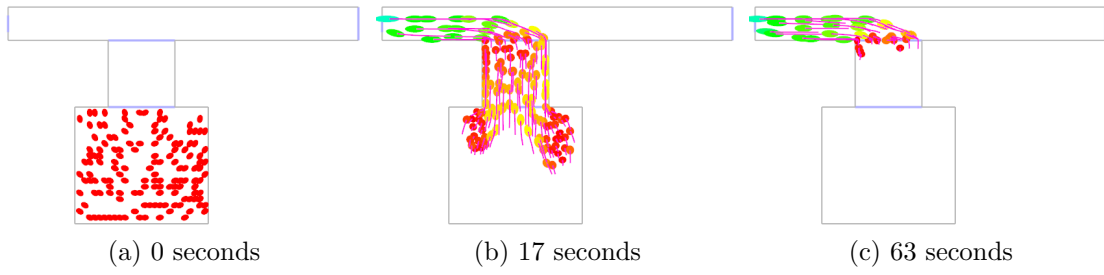


Figure 4.6.: **T-Junction without sensors**

Simulation with complete cognitive map and no sensors. With shortest path router some congestions at corners and doors occur.

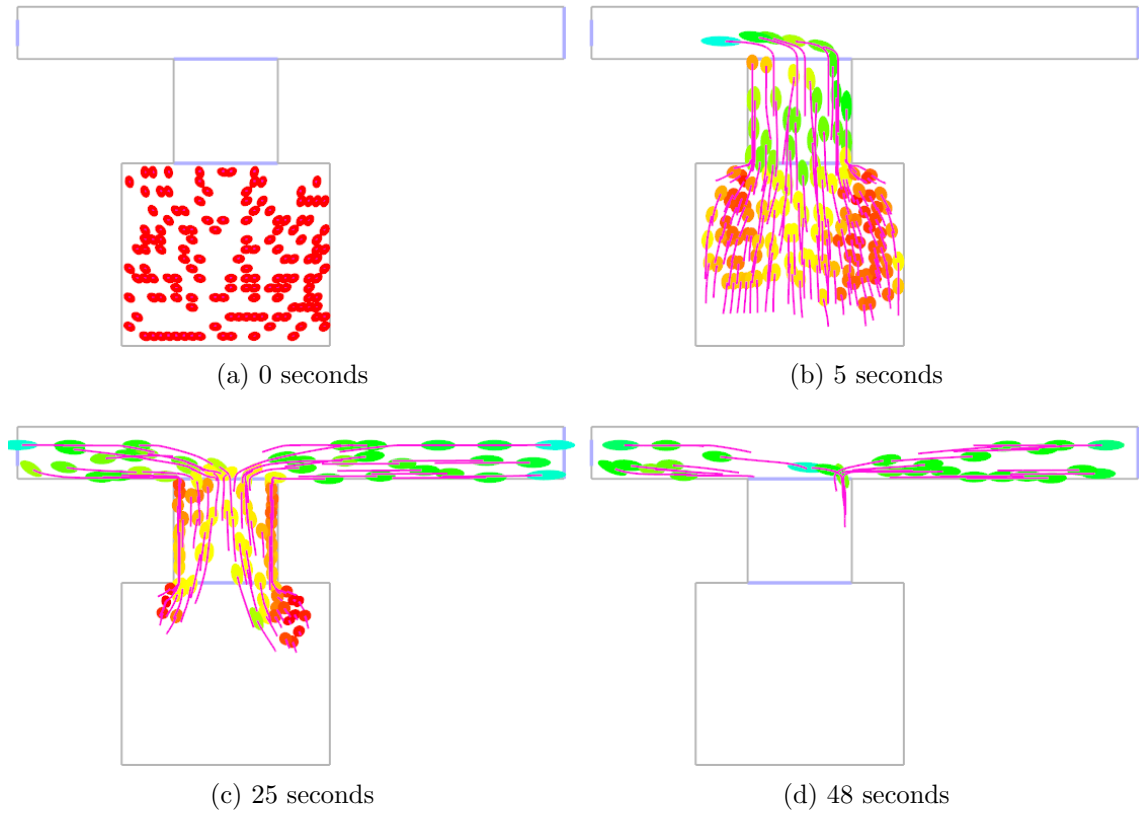


Figure 4.7.: **T-Junction with density sensor**

Simulation with complete cognitive map and `DensitySensor`. Through the `DensitySensor` both exits are used and the agents are distributed depending on the current density.

4.2.4. SmokeSensor

For analyzing the **SmokeSensor** we assumed that the corridor b (figure 4.1) is filled with smoke and thus should not be used. The **SmokeSensor** makes the agents avoid this sub room and willing to make a detour.

The comparison in table 4.5 shows that the total evacuation time is significantly higher with a p-value of $2.205 \cdot 10^{-6}$ than the total evacuation times without sensors. The snapshots from a single simulation in figure 4.8 show that the agents avoid the smoked sub room and take the detour through the two adjacent rooms.

t [s]	Freq.	With sensor	Without sensor	Freq.
16	0	0%	4%	2
17	6	12%	38%	19
18	9	18%	28%	14
19	16	32%	24%	12
20	10	20%	2%	1
21	7	14%	4%	2
22	2	4%	0%	0
<p>p-value $< 2.205 \cdot 10^{-6}$ (Result of Welch's t-test)</p> <p>mean (Total evac. time): 19.18s 17.94s</p>				

Table 4.7.: **Complete cognitive map with SmokeSensors**

Comparison of total evacuation time frequencies of simulations with **SmokeSensor** and without sensors. For the simulations we assumed corridor b (figure 4.1) to be filled with smoke. We conducted 50 simulations each. The simulations are all done with complete cognitive maps. The agents avoid corridor B and try to flee through room 1 and 2 and therefore they need more time.

4. Simulations and results

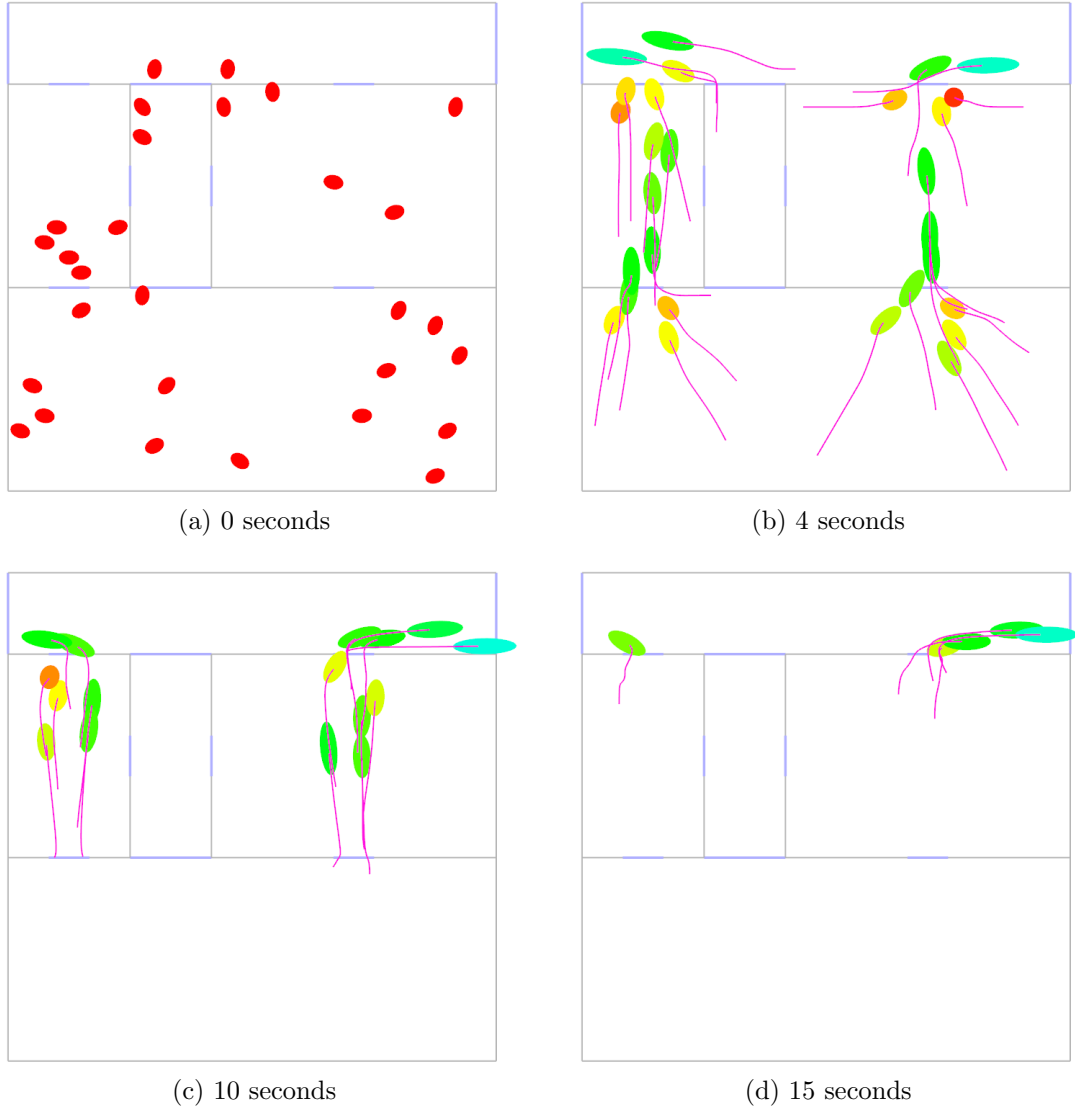


Figure 4.8.: **Simulation with complete cognitive map**

Simulation with complete cognitive map and **SmokeSensor**. In this simulation the vertical corridor (corridor b from figure 4.1) was smoked. The agents avoid to go into this sub room.

4.3. Analyzing different sensors with empty cognitive maps

An empty cognitive map represents the edge case of a pedestrian without any knowledge of the building. This means the orientation of the agent has to be fulfilled locally. A simulation with empty cognitive maps and without any sensor would lead to useless results because the agents would just stay on their initial positions. Under this circumstances we can not choose a new destination simply because the agent doesn't know about any destination. Therefore the `DiscoverDoorsSensor` is mandatory to get meaningful results.

Another important sensor for simulations with empty cognitive maps is the `LastDestinationsSensor`. The local way finding chooses the next destination depending on the edge factor and the distance. If the edge factors are not too different the agent chooses the destination with the lowest distance. This would lead to oscillating agents at the doors because the actual door (the last destination) is always the destination with the lowest distance. The `LastDestinationsSensor` solves this problem and is thus mandatory too.

The simulations for the analysis of empty cognitive maps are done with only one agent because we are mostly interested in the way finding of a single agent and want to avoid side-effects like congestions. The other initial conditions as explained in section 4.1 did not change. The floor plan from figure 4.1 is used. The agent starts in room 3.

4.3.1. Mandatory sensors only

As mentioned before the `LastDestinationsSensor` and the `DiscoverDoorsSensor` are mandatory for simulations with empty cognitive maps. With those two sensors and the `EmptyCognitiveMapCreator` the agent knows nothing at first. After the first run of the `DiscoverDoorsSensor` the agent knows at least the doors of the actual sub room. Since the edge factors do not differ, at least if the agent did not pass the door already (`LastDestinationsSensor`), the agent always chooses the door with minimal distance which extends the average evacuation path length drastically.

This reveals some simulations where the agent explores every door before the corridor with the emergency exits is found. This is not realistic but caused by the absence

4. Simulations and results

of any sensor and thus any valuable information which could provide something like a heuristic. In real life situations humans would rarely evacuate without any heuristic. For demonstration purposes we conducted simulations without additional sensors anyway to show the change in evacuation time with and without additional sensors.

Table 4.8 shows the evacuation time frequencies of one agent in 50 simulations. Figure 4.9 shows the described behavior with some snapshots from one simulation.

t [s]	Frequency	Percentage
18	1	2%
19	6	12%
20	11	22%
21	4	8%
22	10	20%
23	8	16%
24	8	16%
25	1	2%
26	1	2%

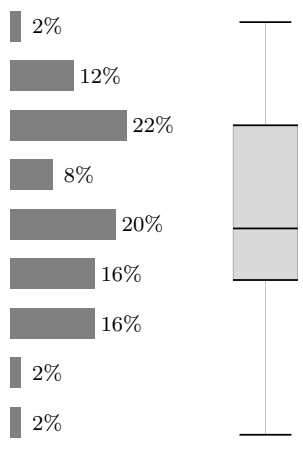


Table 4.8.: **Empty cognitive map no additional sensors**

Evacuation time frequencies of 50 simulations with `EmptyCognitive-MapCreator` and no additional sensors. Simulations are done with one agent and `LastDestinationsSensor` and `DiscoverDoorsSensor`.

4. Simulations and results

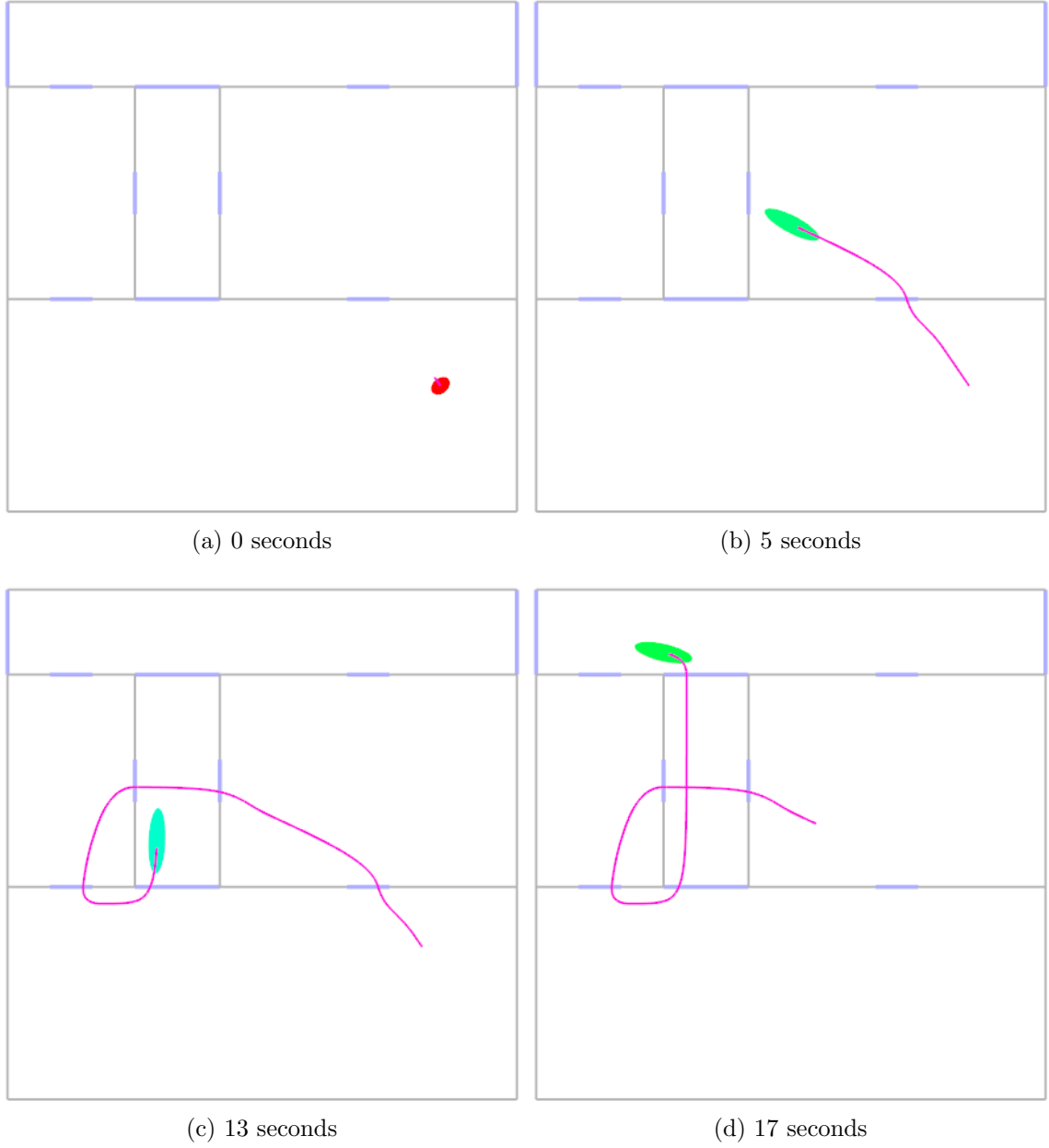


Figure 4.9.: **Simulation with empty cognitive map**

Simulation with empty cognitive map and no additional sensors.

4.3.2. **RoomToCorridorSensor**

With the `RoomToCorridorSensor` the agent uses a heuristic to search for an exit path by preferring corridors over normal sub rooms. This heuristic is quite simple but strongly changes the success of the evacuation in terms of evacuation times. The evacuation times with the `RoomToCorridorSensor` are much lower than without. It should be mentioned, that conducting simulations without any additional sensors presumes that the agent does not have any knowledge as well as no heuristic which could lead to better orientation. That is why the comparison of evacuation times may not be that meaningful. Table 4.10 shows the comparison of the total evacuation times anyway. Figure 4.10 shows snapshots from one simulation. It can be seen that the pedestrian leaves the building over the corridor. This is not the shortest path but this could not be expected from pedestrians without any knowledge of the building.

4. Simulations and results

t [s]	Freq.		With sensor	Without sensor	Freq.
8	2		4%	0%	0
9	7		14%	0%	0
10	10		20%	0%	0
11	10		20%	0%	0
12	8		16%	0%	0
13	8		16%	0%	0
14	4		4%	0%	0
15	1		2%	0%	0
...
18	0		0%	2%	1
19	0		0%	12%	6
20	0		0%	22%	11
21	0		0%	8%	4
22	0		0%	20%	10
23	0		0%	16%	8
24	0		0%	16%	8
25	0		0%	2%	1
26	0		0%	2%	1

Table 4.9.: **Empty cognitive map with additional sensors**

Total evacuation time of a single agent with empty cognitive map and `RoomToCorridorSensor` compared with simulations without additional sensors. The `LastDestinationsSensor` and the `DiscoverDoorsSensor` are mandatory for simulations with empty cognitive maps (see section 4.3).

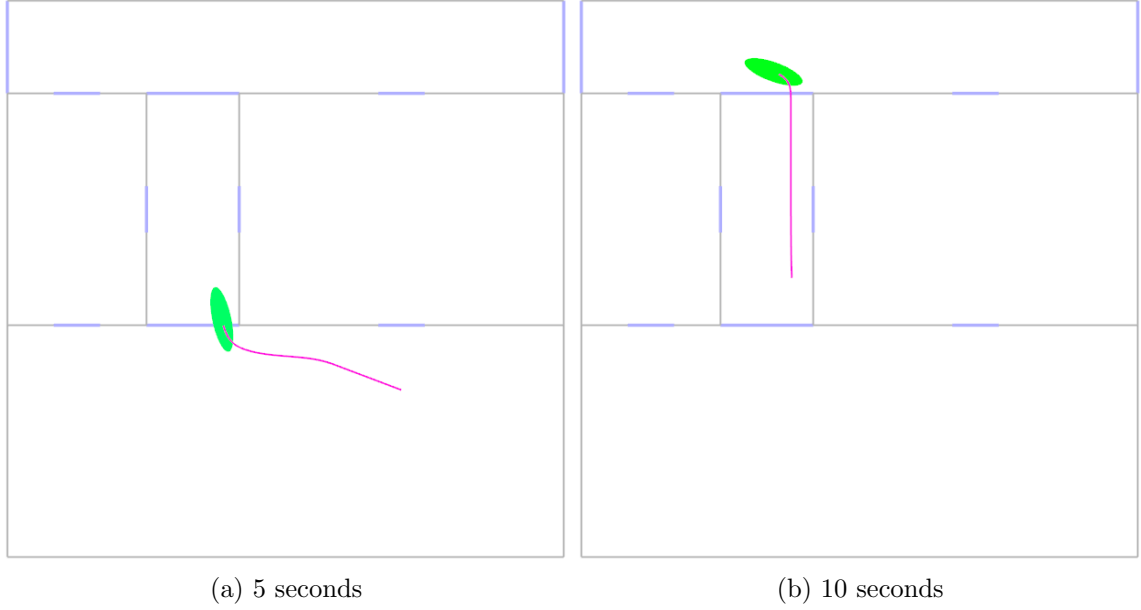


Figure 4.10.: **Simulation with empty cognitive map**

Simulation with empty cognitive map and `RoomToCorridorSensor`.

4.3.3. SmokeSensor

The `SmokeSensor` is used to simulate fire in corridor B (figure 4.1). In contrast to section 4.2.4 the agent has no knowledge at the beginning of the simulation. In this case the evacuation has to be done over room 1 or 2 since the edge from room 3 to corridor B is rated bad and therefore not used during the local path finding. When the agent arrives in room 1 or 2 the only left choice is to go to corridor A since the edge leading from room 1 or 2 to corridor B is rated bad too. In the most cases the described path is the shortest one and thus leads to lower evacuation times. This is an effect of the chosen floor plan and not a characteristic of the combination of an empty cognitive map combined with the `SmokeSensor`. Table 4.10 shows the comparison of the total evacuation times with `SmokeSensor` and without. Snapshots from one simulation and the described route choice can be seen in figure 4.11.

4. Simulations and results



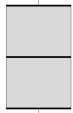














t [s]	Freq.		With sensor	Without sensor	Freq.
6	1		2% 	0%	0
7	2		4% 	0%	0
8	12		24% 	0%	0
9	12		24% 	0%	0
10	15		30% 	0%	0
11	5		10% 	0%	0
12	3		6% 	0%	0
...
18	0		0%	 2%	1
19	0		0%	 12%	6
20	0		0%	 22%	11
21	0		0%	 8%	4
22	0		0%	 20%	10
23	0		0%	 16%	8
24	0		0%	 16%	8
25	0		0%	 2%	1
26	0		0%	 2%	1

Table 4.10.: **Empty cognitive map with additional sensors**

Total evacuation time of a single agent with empty cognitive map and the **SmokeSensor** compared with simulations without this sensor and empty cognitive maps. The **LastDestinationsSensor** and the **DiscoverDoorsSensor** are mandatory for simulations with empty cognitive maps (see section 4.3). For the simulations we assumed corridor B (figure 4.1) to be filled with smoke.

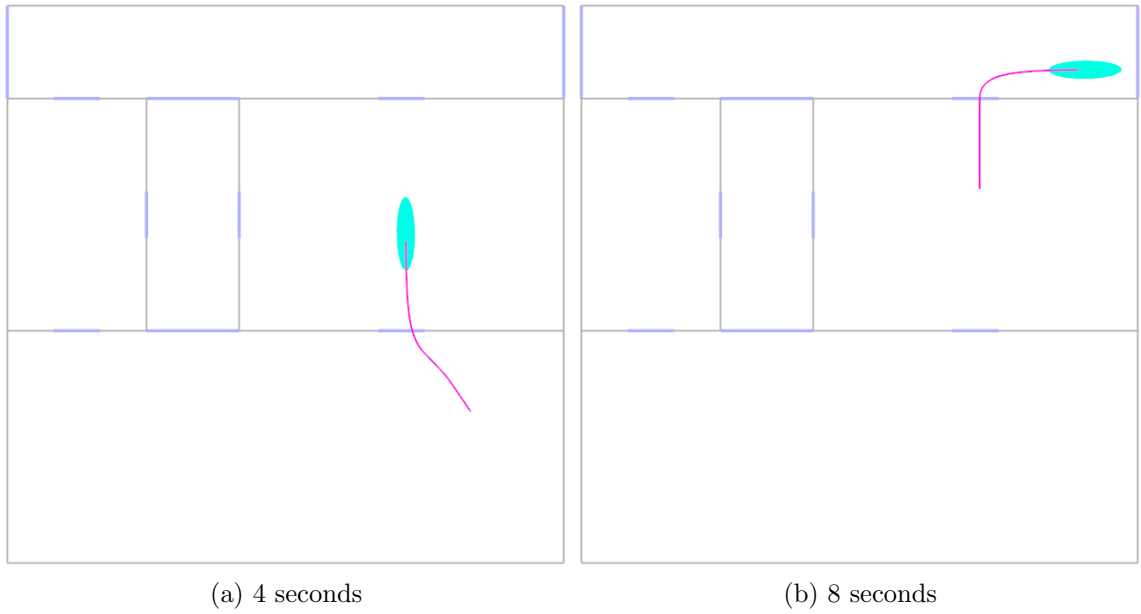


Figure 4.11.: **Simulation with empty cognitive map**

Simulation with empty cognitive map and **SmokeSensor**. For the simulations we assumed corridor B (figure 4.1) to be filled with smoke.

CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this work we implemented a versatile and knowledge based routing framework for pedestrian dynamics. With this framework we propose a adjustable method for emulating human knowledge, perception and decision making. This work does not claim investigating and understanding the nature of human behavior, rather its goal is to create tools to ease the implementation of new behavioral models.

The framework consists out of three modules: the perception module (**Sensors** and **SensorManager**), the knowledge module (**Cognitivemap**) and the decision making module. For the knowledge representation we proposed a simplified cognitive map which reduces the complexity of the model but represents all needed knowledge. For the perception module we implemented a sensor structure and for the decision making we provide a local and a global optimization algorithm. In chapter 4 we showed that the sensors have a high impact on the simulation and are suitable for reproducing human behavior. Especially the sensor module is highly extendable and could thereby fulfill even additional requirements later. For all module the cognitive map is the central module to read information from or to write information into. Due to the object oriented design and the modularization it is possible to exchange or adjust modules independent from each other.

With the proposed framework we build the tools needed to simulate different scenarios and circumstances.

5.2. Future work

There are several extensions and improvements which could be part of future works. Some of them are related to problems which arose during the implementation or the analysis and some are suggestions for possible extensions. One of the most important tasks would be a verification of simulations with empirical data. This task is not just important for this routing framework but for the pedestrian dynamics simulations in general. The next tasks are directly related to our new routing framework and suggestions for further improvements.

Sensors

The perception module offers a lot of further extension possibilities. We already implemented several sensors and showed their impact on the result of simulations. For simulating certain situations and circumstances sensors are a good option. That is why it is advisable to design and implement further sensors. They enrich the information and help to emulate realistic behavior.

Those advantages do not come without a downside. We implemented 5 sensors with different tasks. The `LastDestinationsSensor`, the `DensitySensor`, the `SmokeSensor` and the `RoomToCorridorSensor` are manipulating edge factors of the metric map of the certain cognitive map. With setting an edge factor the sensors have a direct impact on the way finding which is based on the edge weight (section 3.3). Therefore the calibration of parameters becomes complicated depending on the numbers of sensors used. With an increasing number of sensors the decision making becomes less comprehensible and the user has to decide which sensor and circumstances weights more than other. Especially with more than two sensors combinations of circumstances become important too and thus the calibration becomes more complex and difficult. For example a smoked room should be rated as the worst destination even if this means that the agent has to go back and go through a door with a congestion in front of it. This means even a combination of factors from the `LastDestinationsSensor` and the `DensitySensor` need to be lower than the sole `SmokeSensors` factor.

With this work we offer a framework and some sensors but aforementioned those sensors are more a proof of concept than the result of psychological studies on human behavior and route choice. Especially psychological insights on human behavior become more important when it comes to the conceptual design of new sensors. This

is a wide area of work for psychologists and technicians separately and together.

Information sharing and collaboration

One important sensor is the information sharing sensor. When it comes to simulations with empty cognitive maps the absence of any information sharing and collaboration between agents becomes most obvious. A simulation with more than a few agents and empty cognitive maps will cause bidirectional flows. This leads to unnecessary and unrealistic congestions. But also with complete cognitive maps the unrealistic behavior due to the absence of information sharing could become observable. For example if one room in the building is smoked the information is not shared and every pedestrian has to discover this individually. It is assumable that pedestrians in evacuation share certain information about a fire for example or flee in small groups.

The information sharing could be done with the sensor structure but this is maybe not as easy as it sounds. In order to design a sensor with this task several questions arise. We just want to mention some problems related to this task.

All pedestrians have different cognitive maps with different amounts of information content. The classification of knowledge we used in this work could be a good starting point for the information sharing. But besides deciding which kind of information should be shared the distance to the information and actuality is important too. Furthermore it is not quite realistic to have a complete information exchange during a situation like an evacuation. It sounds quite more realistic that a pedestrian with more knowledge points into a direction instead of explaining the building. In addition the certainty of information and the decrease of the certainty during propagation should be recognized during the information sharing.

This leads to the task of investigating information propagation in groups of pedestrians.

Decision making

The actual decision making distinguishes between agents who are able and those who are not able to find a complete exit path. The first group optimizes over the navigation graph whereas the second group just uses the edges of the certain sub room. Thereby it is impossible for agents of the first group to find a shorter path because it is not considered if the agent has a complete or a sparse navigation graph.

5. Conclusion and future work

In contrast to the first group the second group only uses the local knowledge of edges belonging to the actual sub room. This could lead to situations where agents ignore further knowledge they have and make decisions only using a subset of their actual knowledge.

The decision making should always use all knowledge which is available for the certain agent. Additional the strategies could be mixed up to emulate more realistic behavior.

LISTINGS

Code Listing

```
1 int CognitiveMapRouter::FindExit(Pedestrian * p)
2 {
3     //Check for former destination.
4     if((*cm_storage)[p]→HadNoDestination()) {
5         sensor_manager→execute(p, SensorManager::INIT);
6     }
7     /**
8      * Check if the Pedestrian already has a Destination
9      * or changed the subroom.
10    */
11    if((*cm_storage)[p]→ChangedSubRoom()) {
12        sensor_manager→execute(p, SensorManager::CHANGEDROOM);
13        int status = FindDestination(p);
14        (*cm_storage)[p]→UpdateSubRoom();
15        return status;
16    }
17    return 1;
18 }
```

A. Listings

Listing A.1: **CognitiveMapRouter::FindExit** method

The `FindExit` method of the `CognitiveMapRouter` cares about the routing of a single pedestrian in a single time-step.

Code Listing

```

1 int CognitiveMapRouter::FindDestination(Pedestrian * p)
2 {
3     //check for a known exit route
4     const GraphEdge * destination = NULL;
5     destination = (*cm_storage)[p]->GetDestination();
6     if(destination == NULL) {
7         //no destination was found, now we could start the discovery
8         !
9         //1. run the no_way sensors for room discovery.
10        sensor_manager->execute(p, SensorManager::NO_WAY);
11        //check if this was enough for finding a global path to the
12        exit
13        destination = (*cm_storage)[p]->GetDestination();
14        if(destination == NULL) {
15            //we still do not have a way. lets take the "best" local
16            edge
17            destination = (*cm_storage)[p]->GetLocalDestination();
18        }
19    }
20    //if we still could not find a destination
21    //it's an error!
22    if(destination == NULL) {
23        Log->Write(
24            "ERROR: \t Pedestrian (ID: %i) could not find any
25            destination",
26            p->GetID()
27        );
28        return -1;
29    }
30    //Add destination to cognitive map
31    (*cm_storage)[p]->AddDestination(destination);
32    //fire new destination event
33    sensor_manager->execute(p, SensorManager::NEW_DESTINATION);
34
35    //update Pedestrian
36    p->SetExitLine(destination->GetCrossing());

```


A. Listings

```
33     p->SetExitIndex( destination ->GetCrossing()->GetUniqueID() );  
34     return 1;
```

Listing A.2: **CognitiveMapRouter::FindDestination** method

The **FindDestination** method of the **CognitiveMapRouter** cares about the search for a new destination. The decision whether to search a new destination or not is done before.

Code Listing

```

1  const GraphEdge * GraphVertex::GetCheapestDestinationByEdges(
    const Point & position) const
2  {
3      std::set<const GraphEdge*> visited;
4      // map with GrapEdges and their predecessors and distances
5      std::map<const GraphEdge *, std::pair<const GraphEdge *,
        double>> destinations;
6      // priority queue with discovered Edges and their distance.
7      std::priority_queue<
8          std::pair<double, const GraphEdge*>,
9          vector<std::pair<double, const GraphEdge*>>,
10         std::greater<std::pair<double, const GraphEdge*>>
11         > queue;
12     const GraphEdge * exit_edge = NULL;
13
14     // add all out edges from this vertex to priority queue and
        destinations.
15     for(EdgesContainer::const_iterator it = this->GetAllOutEdges
        ()->begin(); it != this->GetAllOutEdges()->end(); ++it) {
16         double new_distance = (*it)->GetWeight(position);
17
18         destinations[(*it)] = std::make_pair((const GraphEdge*)
            NULL, new_distance);
19         queue.push(std::make_pair(new_distance, (*it)));
20     }
21
22     while(!queue.empty()) {
23         const GraphEdge * act_edge = queue.top().second;
24         double act_distance = queue.top().first;
25         queue.pop();
26         //if we discovered an exit edge we are finished (queue
            is distance ordered)
27         if(act_edge->IsExit()) {
28             exit_edge = act_edge;
29             break;
30         }

```

A. Listings

```

31 //discover new edges or shorter paths to old edges
32 const EdgesContainer * new_edges = act_edge->GetDest()->
    GetAllOutEdges();
33 for(EdgesContainer::const_iterator it = new_edges->begin
    ( ); it != new_edges->end(); ++it) {
34     // if the destination edges was visited we already
        have the shortest path to this destination.
35     if(visited.find((*it)) != visited.end()) continue;
36
37     double new_distance = act_distance + (*it)->
        GetWeight(act_edge->GetCrossing()->GetCentre());
38     //check if the destination edge was discovered
        before.
39     if(destinations.find((*it)) == destinations.end()) {
40         //initialize the new discovered vertex with
            distance inifity and push it to the queue
41         destinations[(*it)] = std::make_pair<const
            GraphEdge*, double>(NULL, INFINITY);
42         queue.push(std::make_pair(new_distance, (*it)));
43     }
44     //check if we found a shorter path to the dest
        vertex
45     if(destinations[(*it)].second > new_distance) {
46         destinations[(*it)].second = new_distance;
47         destinations[(*it)].first = act_edge;
48     }
49 }
50 visited.insert(act_edge);
51 }
52 //did we found an exits?
53 if(exit_edge != NULL) {
54     const GraphEdge * act_edge = destinations[exit_edge].
        first;
55     if(act_edge == NULL) {
56         return exit_edge;
57     } else {
58         while(this != act_edge->GetSrc()) {

```

A. Listings

```
59         act_edge = destinations[act_edge].first;
60     }
61     return act_edge;
62 }
63 } else {
64     return NULL;
65 }
66 }
```

Listing A.3: **Modified Dijkstra algorithm**

The modified version of Dijkstra algorithm used for the shortest path calculation.

BIBLIOGRAPHY

- [1] P. Arthur and R. Passini. *Wayfinding: People Signs and Architecture*. McGraw-Hill Book Co., 1992.
- [2] L. Benthorn and H. k. Frantzich. Fire alarm in a public building: How do people evaluate information and choose evacuation exit? Technical Report 3082, Departement of Fire Safety Engineering, Lund University, 1996.
- [3] A. Braun, S. Musse, L. de Oliveira, and B. Bodmann. Modeling individual behaviors in crowd simulation. In *Computer Animation and Social Agents, 2003. 16th International Conference on*, pages 143–148, May 2003.
- [4] M. Chraïbi, A. Seyfried, and A. Schadschneider. The generalized centrifugal force model for pedestrian dynamics. *Physical Review E*, 82:46111, 2010.
- [5] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1 959):269–271, 1959.
- [6] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.
- [7] S. P. Hoogendoorn, P. H. L. Bovy, and W. Daamen. Microscopic Pedestrian Wayfinding and Dynamics Modelling. In M. Schreckenberg and S. D. Sharma, editors, *Pedestrian and Evacuation Dynamics*, pages 123–155, 2002.
- [8] A. U. Kemloh Wagoum. *Route choice modelling and runtime optimisation for simulation of building evacuation*. Dr., Jülich, 2013.
- [9] B. Kuipers. Modeling Spatial Knowledge*. *Cognitive Science*, 2(2):129–153, Apr. 1978.

BIBLIOGRAPHY

- [10] B. Kuipers. The cognitive map: Could it have been any other way? In H. L. Pick Jr. and L. P. Acredolo, editors, *Spatial Orientation*, pages 345–359. Springer US, 1983.
- [11] X. Pan. *Computational modeling of human and social behaviors for emergency egress analysis*. PhD thesis, Stanford University, 2006.
- [12] N. Pelechano, K. O’Brien, B. Silverman, and N. Badler. Crowd Simulation Incorporating Agent Psychological Models, Roles and Communication. In *1st Int’l Workshop on Crowd Simulation*, 2005.
- [13] G. Proulx. The Time Delay to Start Evacuating upon Hearing a Fire Alarm. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 38(14):811–815, Oct. 1994.
- [14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.
- [15] J. D. Sime. The concept of panic. *Fires and human behaviour*, 1980.
- [16] J. D. Sime. Crowd facilities, management and communications in disasters. *Facilities*, 17(9/10):313–324, 1999.
- [17] E. C. Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189–208, July 1948.
- [18] B. Welch. The generalization of student’s’ problem when several different population variances are involved. *Biometrika*, 34(1/2):28, Jan. 1947.
- [19] W. J. Yu, L. Y. Chen R. Dong, and S. Q. Dai. Centrifugal force model for pedestrian dynamics. *Phys. Rev. E*, 72(2):26112, Aug. 2005.

ERKLÄRUNG

Hiermit erkläre ich, dass ich die am heutigen Tag eingereichte Diplomarbeit zum Thema "A knowledge-based routing framework for pedestrian dynamics simulation." unter Betreuung von Jun.-Prof. Dr. Kathrin Padberg-Gehle und Dr. Mohcine Chraïbi selbstständig erarbeitet, verfasst und Zitate kenntlich gemacht habe. Andere als die angegebenen Hilfsmittel wurden von mir nicht benutzt.

Datum

Unterschrift