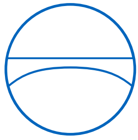

Technische Universität München

Ingenieurfakultät Bau Geo Umwelt

Lehrstuhl für Computergestützte Modellierung und Simulation

Integration of BIM-based pedestrian simulations in the early design stages

Bachelorthesis

für den Bachelor of Science Studiengang Umweltingenieurwesen

Autor:	Janik Scholl
Matrikelnummer:	
1. Betreuer:	Prof. Dr.-Ing. André Borrmann
2. Betreuer:	M. Sc. Jimmy Abualdenien

Ausgabedatum:	12. February 2019
Abgabedatum:	02. August 2019

Abstract

The advanced development of pedestrian-flow simulations enables Architects and Engineers in cases, which are difficult to calculate by conventional manual calculation methods, to proof if a building fulfils its fire-safety-requirements. However, these simulations are most times conducted in the later in a buildings' design phases, since the manual creation of such simulations is labour intensive and costly. Recent studies though have proven the beneficial impact of early stage simulations. Thus, this thesis examines the development of a concept and implementation of an automated creation of "Early Stage Pedestrian Simulation". The field of Energy Management, where such concepts have already been developed, functions as prime example and basis for the research concept. As a case-study, the thesis' concept is developed for the pedestrian-flow simulation "crowd:it".

Zusammenfassung

Der fortgeschrittene Entwicklungsstand von Personenstrom-Simulationen ermöglicht Ingenieuren und Architekten in, mit üblichen Handrechenverfahren schwer zu berechnenden Fällen, das Erreichen von Brandschutzzielen nach zu weisen. Diese Simulationen werden jedoch meist erst in späteren Entwicklungsphasen eines Bauprojekts durchgeführt, da das manuelle Anfertigen solcher Simulationen sehr arbeitsintensiv und teuer ist. Neuste Studien belegen jedoch den positiven Einfluss von Simulationen in frühen Entwicklungsphasen. Infolgedessen beschäftigt sich diese Arbeit mit der Entwicklung eines Konzepts und Implementierung zur automatisierten Anfertigung sogenannter “Early Stage Pedestrian Analysis”. Als Grundlage dafür dienen bereits Entwickelte Konzepte aus dem Feld des Energie Managements, wo solche Analyse Konzepte schon entwickelt wurden. Das Konzept der Arbeit wird als Fallstudie für die Software zur Simulation von Personen Strömen „crowd:it“ entwickelt.

Table of Contents

Table of figures VI

Table of tables VIII

Table of Abbreviations IX

1 Introduction and Motivation 10

1.1 Motivation ... 10

1.2 Research Goal and structure ... 11

2 Literature Review: Early Stage-of-Development Simulations 12

2.1 Introduction ... 12

2.2 Level of Development ... 12

2.3 Fuzziness ... 13

2.3.1 Early Stage Simulation Evaluation Methods ... 14

2.4 Conclusion ... 15

3 Pedestrian-Flow Simulations 16

3.1 Crowd:it ... 17

3.1.1 Geometric representation ... 18

3.1.2 Creating Paths ... 23

3.2 Flood-field ... 24

3.2.1 Type of Simulations ... 25

4 Concept 26

4.1 Setup ... 26

4.2 Level of Development ... 27

4.3 Determination of input Parameters ... 28

4.3.1 The “.floor”-file ... 29

4.3.2 Transform-Operations ... 31

4.3.3 Fuzziness ... 35

4.4 Manipulating the floor-file ... 35

4.4.1 The Crowdit File ... 35

5 Implementation 37
5.1 File Structure....37
5.1.1 File....38
5.2 Classes....39
5.2.1 Geometry....39
5.2.2 Parser.cs....41
5.2.3 UncertaintyManger.cs....43
5.2.4 Manipulator.cs....43
5.2.5 Program.cs....44

6 Evaluation 46
6.1 Setting up the Simulation....46
6.2 Input Parameters....46
6.3 Results....47

7 Conclusion 50

8 Table of Literature 51

Appendix A 53

Appendix B 54

Table of figures

Figure 3.1: (Plum & Jäger, 2011) Evacuationmodels 16
Figure 3.2: Unassigned simObj 18
Figure 3.3: Origin 19
Figure 3.4: Destination....19
Figure 3.5: Stair....20
Figure 3.6: Escalator....21
Figure 3.7: WaitingZone 22
Figure 3.8: Elevator and Elevator Matrix....22
Figure 3.9: A complex path's “Path-Tree” 23
Figure 4.1: BDL stages 1-5. Abualdenien and Borrmann (2018)....28
Figure 4.2: Sample-Project....29
Figure 4.3: XML-.floor-file 30
Figure 4.4: Possible outcomes of moving and scaling neighbouring walls....32
Figure 4.5: Scaling and moving a door 33
Figure 4.6: The three different cases of neighbouring walls defined for the concept .34
Figure 4.7: The ".floor"-file 35
Figure 5.1: The concepts folder structure, pre early stage simulation 37
Figure 5.2: Input Xml-File 38
Figure 5.3 CalculateMidpoint() Function: 40
Figure 5.4: The Parser....41
Figure 5.5: The read status 42
Figure 5.6: The write status 42
Figure 5.7: The input status....43
Figure 5.8: The program class's process 45
Figure 6.1: dxf-File before (left) and after (right) the setup....46
Figure 6.2: The buildings floorplans in crowd:it. 47

Figure 6.3: Diagram of the evacuation times for each variation, sorted in order: Parameter A, B, C, D,E 48

Figure 6.4: Diagram of the evacuation times for each variation, sorted in order: Parameter B, A, C, D,E 48

Figure 6.5: Diagram of the evacuation times for each variation, sorted in order: Parameter C, A, B, D,E 49

Figure 6.6: Diagram of the evacuation times for each variation, sorted in order: Parameter D, A, B, C,E 49

Figure 6.7: Diagram of the evacuation times for each variation, sorted in order: Parameter E, A, B, C, D 49

Table of tables

Table 4.1: List of Simulation and pedestrian settings available in crowd:it.................27

Table 6.1: Evacuation times and the uncertain parameters.......................................48

Table of Abbreviations

BIM	Building Information Modelling
simObj	Simulation Object
LOD	Level of Development
BLD	Building Development Level
MCM	Monte Carlo Method

1 Introduction and Motivation

1.1 Motivation

When public buildings are being developed, a major concern is the users' safety. Fire-safety regulations require many different standards which must be fulfilled. Multiple categories of fire-safety measures do exist. Some apply to the materials which must be fire resistant or fire-retardant. Other laws specify the number of people which can stay in given areas, or the maximum evacuation time for the pedestrians in those areas. Modern pedestrian-flow simulations offer Architects and Engineers additional tools to verify if these requirements are met. Currently however, those simulations are carried out at the end of the planning cycle, when only minor changes to the building's layout can be done and changes to a buildings design, if needed, might be seen as setback and costly to achieve.

The ability to analyse a building in its early stages could spot bottlenecks and structural weakness whilst the design can be altered without much of an additional effort. It enables planners to test the performance of different designs or to detect highly influential key parameters for successful evacuations. One could analyse several early designs and improve its layout early on, instead of having to proof a designs effectiveness in the end of the development progress. This workflow of simulating early design is more desirable as it is proved to be beneficial and, essentially, reduces costs whilst also improving critical performance aspects.

It is technically possible to conduct early stages simulations, much like it is common practice in the energy management sector to use methods of early design uncertainty analysis. However, customers are most times unwilling to pay for such analysis, as pedestrian simulations up to this day are mainly manually developed and the early stages uncertainty of information would require for many, slightly differing simulations to be created.

1.2 Research Goal and structure

This thesis' goal is to develop a concept for an implementation of early stage pedestrian simulation. The research for the concept's development will be derived from concepts of the use of uncertainty and sensitivity analysis of early stage building projects conducted in the energy sector.

The thesis is divided in the following chapters:

- In Chapter 2 we review latest literature on the topic of uncertainty and sensitivity analysis. We examine several papers approach to state-of-the-art early stage simulations. The knowledge gained in this chapter will be used for the creation of our own concept, applied to early stage pedestrian simulation.
- In Chapter 3, different concepts of pedestrian-simulations are briefly studied. The software crowd:it is introduced, as it will be used as this thesis' platform for the development and implementation. Therefore, its functionalities are explained in more detail.
- In Chapter 4 we develop our concept of early stage pedestrian simulation. Therefore, the knowledge gathered from Chapter 2 and 3 is synthesised. Simplifications, boundary conditions and assumptions are set. Crowd:its geometrical representation is examined and rules established to manipulate its data. The concept of level of development and possible in- and output parameters are discussed.
- Chapter 4 summarizes the implementation methods used and describes the key parts of code.
- In Chapter 5 to prove the concepts functionality, a sensitivity analysis is conducted on a buildings early stage of design.
- In Chapter 6 the results of this thesis are critically discussed. Concepts and ideas for future research are proposed.

2 Literature Review: Early Stage-of-Development Simulations

Early stage uncertainty analysis has seen a lot of attention in the building sector as of recent years. Especially concepts to predict a designs energy consumption has been in focus of current research, as buildings pose for a third of the world's energy consumption (IPCC, 2018). Therefore, as part of the Paris-Agreement, the EU, aiming to be CO2 neutral by the year 2050 (European Commission, 2019), has ruled the building industry to vastly reduce newly designed buildings' energy needs to contribute towards achieving the 20/20/20 Goals. This led to the development of several concepts for early stage calculation of energy consumption.

The following Literature review examines recent researches conducted and terminology being used for early level of development (LOD) analysis. The knowledge gained in this chapter will be used as basis of a concept of pedestrian simulations in an early LOD.

2.1 Introduction

It is being acknowledged that conducting early stage analysis has high impacts on a building's performance (Bogenstätter, 2000). Krygel (2018) claims that: "Energy simulation to provide feedback during the early stages of design is often not done, even though decisions at this stage have the largest impact on energy and cost". However, it is noteworthy that, as Singh (2018) states: "A lot of modelling efforts required to make physical simulation models, also an automatic translation of BIM data to BEM *(Building energy management)* data hasn't proved much reliable (J. B. Kim et al., 2015)".

2.2 Level of Development

A buildings design progress is split up in different Levels of Development (LOD). The American BIMForum defines five stages of LOD (BimForum, 2019) with different levels of information available, increasing at each design stage (Singh, Singaravel, & Geyer, 2018).

Harter, Schneider & Lang (2018) discuss the LOD model to conduct a Life Cycle Assessment of Buildings in early design stages. For this purpose, they extract information at different stages of LOD from BIM models. However, comparing two different stages

of LOD, they find that some parameters lack information in early design stages, which have to be estimated to increase the calculations accuracy.

Abualdenien and Borrmann (2018) criticise the LOD definitions to be informal and imprecise, as they only bring textual and graphical information, which leads to multiple ways of interpretation and different expectation for the detail of information at each level. Additionally, they claim that BIM tools produce too detailed designs even in early stage LODs. Their demand is to precisely define LOD requirements and incorporate their uncertainties to improve the quality of collaborative process.

As concept to solve the problem of the LOD's unclear definition, Abualdenien and Bormann (2018) developed a 5 stage LOD concept for the overall building, with a new term Building Development Level (BDL).:

> BDL1 represents the building as 2D site plan with information about the building's usage, position and orientation. BDL2 defines the buildings height, thus creating a 3D model out of the 2D plan. They add information about the foundation and the buildings external components' midsurfaces. In BDL3 the authors add information about the structural system, construction type and materials. Storeys are introduced and defined. The inner structures are defined in BDL4, creating internal spaces. They also add percentage of opening for each level and allow for estimated loads to be defined. BDL 5 is adds more precise materials, construction type, load and layer structure.

By using this concept, the authors intend to describe the uncertainty of information by explicitly describing the maturity the information available. Furthermore, they state that their approach allows simulations to be conducted on early stage buildings while preventing false impressions of high accuracy through the consideration of fuzziness.

2.3 Fuzziness

"Missing information can only be estimated within a certain range of fuzziness" (Harter, Schneider, & Lang, 2018). The term "fuzziness" is used to describe variability of an elements attribute values due to lack of information or knowledge resulting from early stages of design. Hence, early design simulation always must deal with certain degrees of fuzziness. Models developed require a certain level of information, missing information is supplemented by suitable assumptions and rules (Singh, Singaravel, & Geyer, 2018). For instance, Harter, Schneider and Lang (2018) claim that estimation

of information always lies within a certain fuzziness range, which is based on empirical studies.

2.3.1 Early Stage Simulation Evaluation Methods

Confirming Abualdenien and Borrmann's proposal to further specify the LOD concept, Sing, Singaravel & Geyer (2018) developed an alternative LOD model:

> They named their model adaptive LOD (aLOD). aLOD has three stages of development. Sing et. al. used their model to conduct an Energy Prediction a multilevel model. Each individual aLOD has parameters that have a defined range of fuzziness. After creating the model in aLOD1 all possible combinations of these parameters are generated and analysed. The analysation serves as feedback, which is then used to re-design aLOD and repeat the cycle or design aLOD2. In aLOD2 the specified parameters are again part of a feedback and development loop, until the user is satisfied with the results and moves on to design aLOD3, where the process is repeated.

To generate the variations, Sing et. al. use of the Monte Carlo Method (MCM). In their study, they aimed to estimate a buildings energy consumption for each aLOD. Hence, the annual energy consumption for each of the variations created by the MCM was estimated. They state that the alternatives at each aLOD can be chosen based on lowest mean, min, or max value.

Hygh, DeCarolis, Hill & Ranjithan (2012) concept is an expansion to Sing et al's method, by conducting a linear regression on the database created from the MCM in combination with the energy consumption estimation. Following, the regressions coefficients were normalized to permit comparison. In contrast to sensitivity analysis, this approach enables to predict behaviour when certain components underly uncertainties.

Another evaluation method is the sensitivity analysis- where a systems sensitivity towards the uncertainty of its parameters is being tested. Design decisions could be supported by identifying the most influenced parameter using sensitivity analysis (Rumnici & Abualdenien, 2019)

2.4 Conclusion

Lack of information granted by early LOD models is not to be understood as negative term. This lack of information, also called fuzziness is the driving force behind early stage analysis. After a model to represent the projects level of development is being picked or defined, parameters with certain degree of fuzziness are the research’s basis. One has to assume the min and max range of fuzziness appropriate for his level of development. Once this step is taken, the lack of information provides the possibility to create uncertainty analysis to gradually improve a buildings concept. Another approach is to conduct sensitivity analysis, driven by the input parameter’s fuzziness, to gain greater knowledge of a building’s behaviour.

3 Pedestrian-Flow Simulations

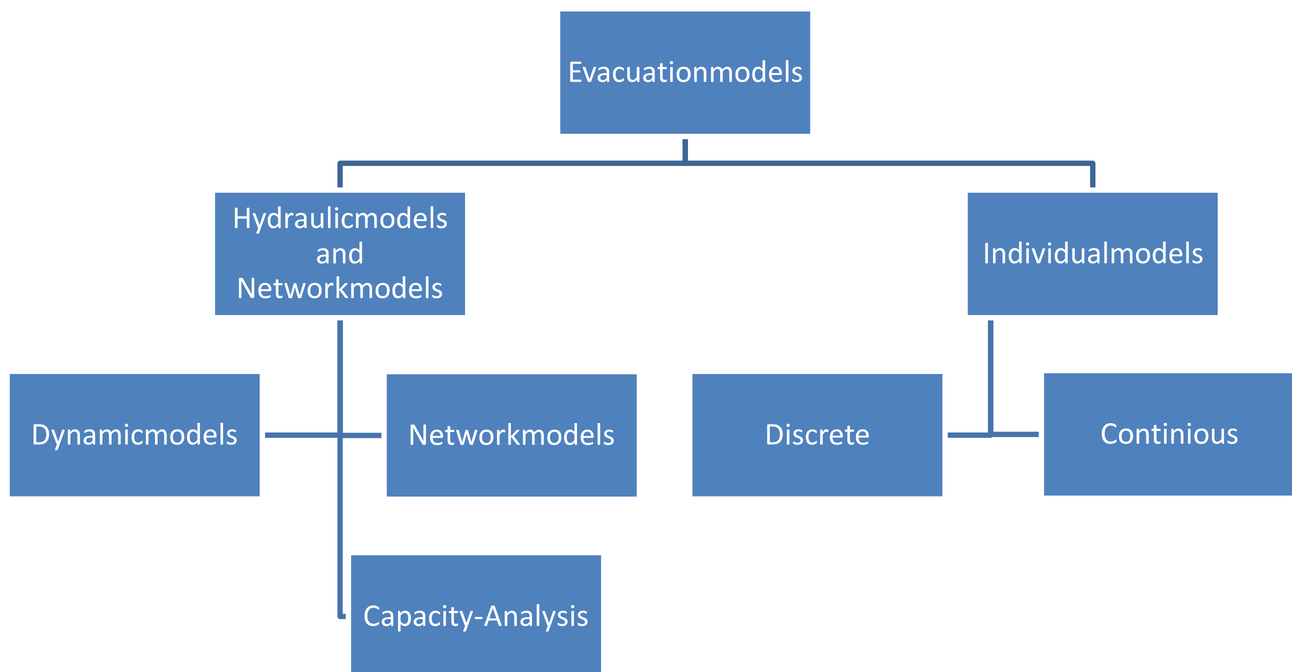

Figure 3.1: (Plum & Jäger, 2011) Evacuationmodels (Translated)

Different types of pedestrian-flow Simulations can be split into two main categories and several subcategories each. These models contrast in approaching the simulation of pedestrian flows. The branch of Hydraulic-models applies empirical data gained from field studies and tries to predict a masses movement similar to a liquids behaviour. In contrast, individual-models are focused on an a single human's behaviour and assign personas to their pedestrian models, aiming to mimic spontaneous, unexpected behaviour of moving crowds caused by an individual's decision.

According to Plum and Jäger (2011), a Hydraulic-modes describe uses empirical data to measure flow rates dependent on spatial pedestrian density. For specific groups, evacuation paths and stationary flowrates, one can calculate an estimation of a building's evacuation time. Dynamic-models focus on pedestrian flow to gain information about move speed. Network-models represent paths as graphs, adding the ability to consider individual parameters, such as immobile pedestrians.

Kneidl (2013) explains Microscopic-approaches to be considering each pedestrian as individual. This allows local events to be simulated, such as congestions in front of

stairs or exits or the development of one-directional pathways. Microscopic models can be divided into either discrete or continuous models, which differentiate in spatial and temporal discretization. According to Kneidl, high granularity in terms of spatial discretization enables a simulation result to be consistent, the downside being the increased demand of computation power needed to conduct such simulation.

3.1 Crowd:it

Crowd:it is a software package for microscopic, agent based crowd simulation, developed on the “Optimal Steps Model” by (Seits & Köster, 2016), as well as on Dr. Angelika Kneidl’s research at Technical University Munich (accu rate, 2019).

Accu:rate (2019) describes crowd:it as based on a three-tier model consisting of:

• *A locomotion layer (how exactly do people move through space: The Optimal Steps Model).*

• *A navigation layer (graphs that map the orientation of people).*

• *A behavioural layer.*

The underlying Optimal Steps Model enables real-world pedestrian stepping behaviour. Agents slow down naturally when faced with dense crowds by taking smaller steps. (Thus, no density-speed relation is needed as input.) Agents reflect real human behaviour, avoid collisions with each other and obstacles, and seek the easiest way to their destination. As a result, congestion, lane formation and inefficient pedestrian routing are depicted realistically (accu rate, 2019).

3.1.1 Geometric representation

Crowd:it defines two kinds of geometry objects:

- Simple geometry, which can be a point, edge or polygon. These objects pose as obstacle and cannot be crossed by pedestrians.
- Simulation objects (simObjs), which can further be divided into the following sub-categories:

Unassigned simulation Objects:

Undefined simObjs (orange) of which all other types of simObjs are created from.

Pedestrians cross these objects as they will not be considered by the floodfield.

Figure 3.2: Unassigned simObj

Origins:

Area in which agents are spawned.

Attributes:

- Min premovement time(s)
  Min time agents wait before starting to move
- Max premovement time(s)
  Max time agents wait before starting to move
- Sorted Birth Cells
  Determines if agents in this cell are to be spawned in a spatially sorted or random distribution
- Interval name
  Name of interval assigned
- Generate from(s)

Point of time where the fist agent is spawned

- Generate to(s)

  Point of time where the last agent is spawned

- Number of Agents

  Number of agents to be spawned

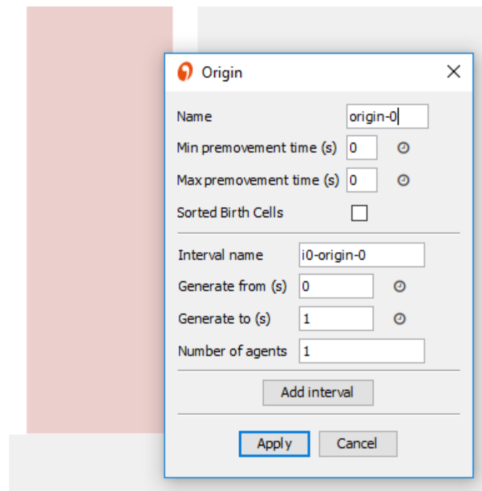

Figure 3.3: Origin

Destinations:

Area which can be used as paths intermediate- or end destination. Agents disappear once they reach their end destination.

Attributes:

- Disable dynamic flooding

  Toggles the dynamic floodfield

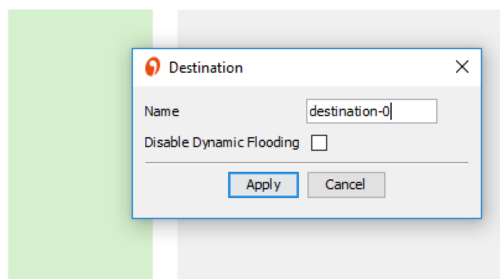

Figure 3.4: Destination

Stairs:

Connector between two floors.

Attributes:

- Number of Treads

  Number of treads

- Tread width

Tread width in meters

- Connects to floor

  Floor that the stair connects to. If empty, the stair ends at the same level as it started

- Direction upwards

  Indicates the stairs upwards direction (arrow in fig. 4.5). Can be changed by pressing "Turn 90°"

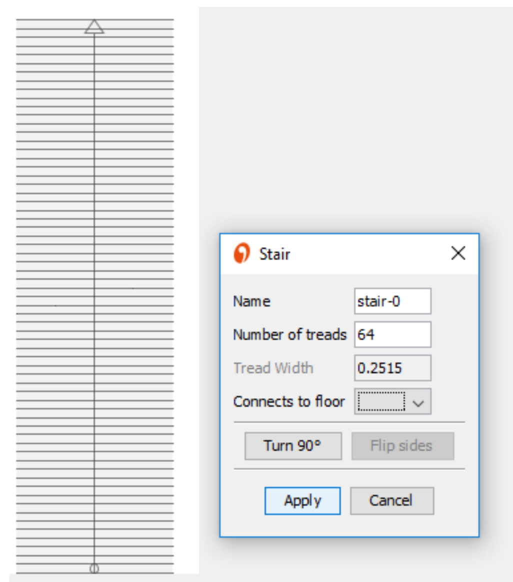

Figure 3.5: Stair

Escalator:

Connector between two floors. All Agents are moving at same speed once they stepped on the escalator.

Attributes (additional to stair's attributes):

- Speed in m/s

  Agents' travel speed on escalator

- Number of Landing Treads

  Number of landing treads

- Travel Direction

  Agents' travel direction indicated by the smaller arrows in the bottom. Changeable by clicking "Flip travel Direction"

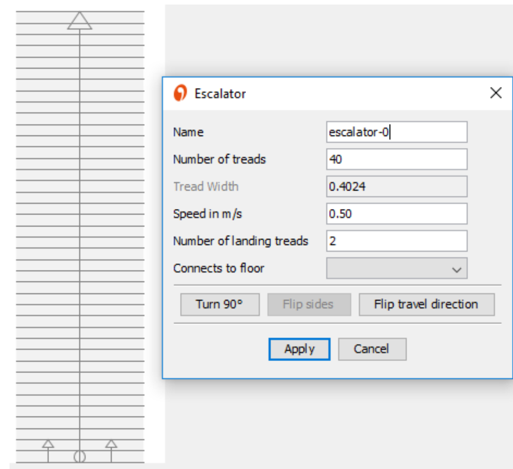

Figure 3.6: Escalator

WaitingZone:

Area in which agents wait for an fixed amount of time, an interval to end or other agents/group members to arrive.

Attributes:

- Capacity

  The waitingZone's capacity
- Deviation

  Deviation of time to wait for agents
- Distribution

  The deviations random distribution. Can be normal, uniform or distributed
- Recurring every (s)

  Determines if the waitingZone's opening window reopens periodically after a certain amount of time
- Time to wait(s)

  Time agents must wait before leaving the waitingZone

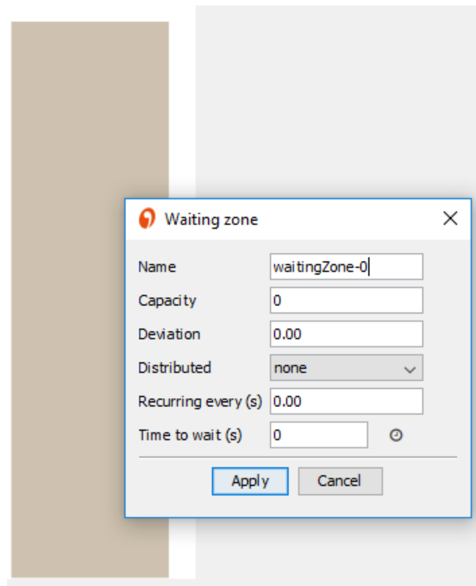

Figure 3.7: WaitingZone

Elevator:

Area connecting two floors. Agents entering this areal are teleported to their destination floor, where they wait until the elevators travel time is over.

Attributes:

- Capacity

The elevators capacity

- Boarding time

Time needed for agents to fully enter the elevator

- Elevator Matrix

Matrix containing the travel times between each floor

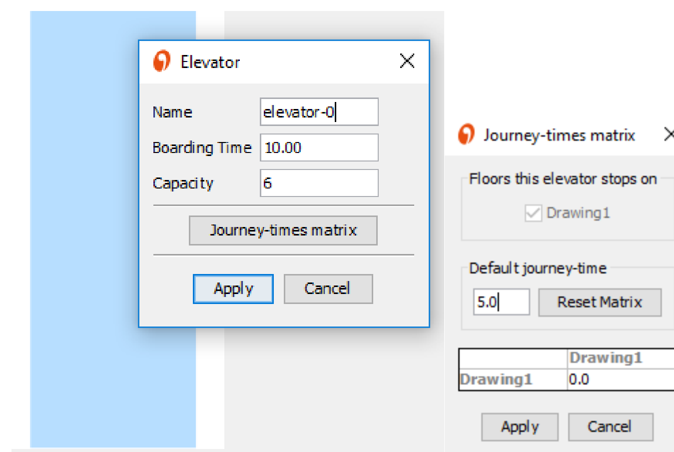

Figure 3.8: Elevator and Elevator Matrix

3.1.2 Creating Paths

In crowd:it, paths represent the only possibility for agents to move from point A to point B. Paths are modelled by hand and can only contain simObjs or a collection of these. Agents moving along a path start at the first object (origin), where they are tasked to try to reach the next object in the paths list. Once they have reached this object, they are assigned their next destination object. This process is repeated until an agent cannot reach its next path-element or the end of the path is reached, where the agent is being de-spawned. Every path must start with an origin (or a set of origins) and end with a destination (or set of destinations). A path may contain unlimited amounts of simObjs, sets and pathsnippets, as well as sets of sets, sets of pathsnippets etc. This structure allows the user to model complex paths.

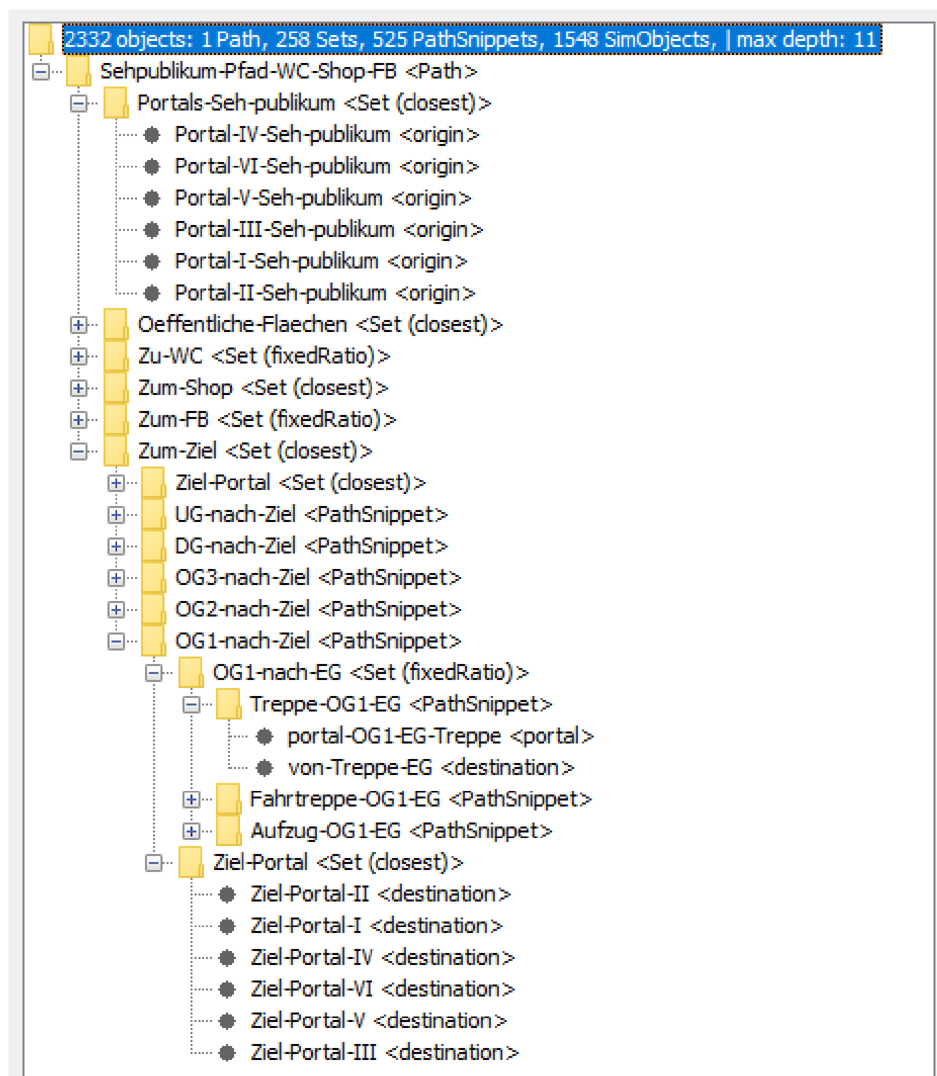

Figure 3.9: A complex path's "Path-Tree"

The "Portals-Seh-publikum" set contains 6 origins. This is the paths entry point. The next members in the list are a mixture of sets and pathSnippets. The last member of the path ends with a set of destinations.

3.1.2.1 Sets

Sets are a collection of at least two different objects. These objects can be simObjs, pathSnippets or other sets. Whenever an agent enters a set along its path it must decide which of the sets objects it chooses to be the next destination. Agents do not know their next destination prior to entering the set, the decision is made “on the fly” while the simulation is taking place. Its decision is dependent on the heuristic assigned to the set.

Each set must have one of the following different heuristics assigned to it:

- closest: Agents seek the closest object within the Set. This is the default heuristic.
- less Crowded: Agents seek the least populated object within the Set, and then the closest of these.
- distributed: Agents select uniformly any object within the Set.
- distributedAndEmpty: Agents select uniformly any object within the Set, provided it has the capacity for them. For instance, you may wish to simulate seats by setting each seat as a Waiting Zone. Each seat must be empty to be selected by an agent.
- shortestQueueLane: Agents choose a queue with the fewest number of people. Naturally, this is only appropriate if a set contains a Waiting Zone.
- fixedRatio: Fixed values can be set, that define the distribution among the set members. The sum of the values per set must sum up to 1

3.1.2.2 Pathsnippets

Pathsnippets are a collection of objects, which agents must go through in the order in which they appear in the pathSnippet. Pathsnippets contain simObjs, sets and other pathSnippets. Other than set, pathSnippets only have one heuristic “InOrder”, leaving agents with no choice to choose from for the next object as their destination.

3.2 Flood-field

The floodfield stores the forces’-values affecting an agent. This force guides the agents along their path. Negative (push-)forces are created by obstacles, for instance walls or, in case of a dynamic floodfield, cross- and counterflows. Positives (pull-)forces are generated by destinations to “pull” the agents. The floodfield can be described as

potential a paths origin and destination, derived from those forces. Its values are stored in floor-cells, whose size is dependent on the chosen spatial granularity.

3.2.1 Type of Simulations

Crowd:it does not distinguish between different simulation approaches. However, in practice, simulations can be split into two different main categories, comfort studies and evacuation simulations. Out of those two kinds, evacuation simulations are more specifically defined- to the degree that accu-rate currently develops an automated report system for evacuation simulations.

Evacuation simulations usually start with a fully occupied building. They either end after the building was successfully evacuated or after a predefined (most times ruled by fire-safety regulations) evacuation time is exceeded. The paths in such simulations tend to be less complex as agents are advised to follow a fire protection concept for evacuation, which directs them to the next closes (emergency-) exit. The process of evacuation can be described as people independently getting in safety from endangered areas (RIMEA, 2019).

The term “Comfort studies” unifies all other kind of simulations. Comfort studies range from simulating the course of a day of in a museum, over a mass’s movement within a football stadium up to the early hours in an office building or peak-hours of a train-stations’ rush-hour. This setup offers infinitely more different path options, as different personas might show different behaviour or get assigned to different paths. In order to conduct comfort studies, a buildings usage must be clearly defined, which is usually beyond an early stage of LOD.

4 Concept

4.1 Setup

The boundary conditions between each testcase must be equal in order to be comparable. Crowd:it offers a variety of settings for the simulation, floor-discretization and agents. It is desirable for simulations results to be as exact and realistic as possible. In contrast, it is necessary to reduce simulation time, since even simulations with small geometry and only a small number of agents may take up to a minute to be computed each. Small projects with e.g. five objects with three uncertainty values each would require 3^5 =243 individual simulations to be carried out. Considering the desired complexity and number of uncertain objects in future projects, some trade-offs in simulation accuracy are unavoidable. Table 4.1 contains a collection of simulation settings and agents' attributes together with an explanation for their function and the value used in our project. Agents' values for movement speed and size are based Weidmann (1993).

Parameter	Function	Value
Cell size [meters]	Describes each cell's size (default: 2)	2.00
Compress output	Will zip the simulation output when checked (default: ON)	TRUE
Distance between two points [meters]	Distance between two points	0.20
Update rate floor field (simulation step)	Specifies the rate at which the dynamic floor field values should be re-calculated. The rate is per time-step, i.e. 1 means one calculation per time-step. By increasing the rate, the simulation is more realistic, but computation time increases (default: 1)	2
Use dynamic floor field	Toggles an algorithm that makes the simulation far more realistic but increases computation time significantly (default: TRUE)	FALSE

Use undirected floor filed	Toggles an algorithm that speeds up simulation calculation when you have many origins or destinations (over 100) (default: FALSE)	FALSE
Cell discretization	Setting of the discretization for the project's floors (default: 0.10)	0.20
Min velocity [m/s]	An agent's minimum walking speed	0,46
Mean velocity [m/s]	Average walking speed of all agents in the scenario	1,34
Max velocity [m/s]	An agent's maximum walking speed	1,61
Deviation for velocity [m/s]	Populations standard deviation of velocities	0,26
Min torso diameter [m]	An agent's minimal possible torso diameter	0,42
Max torso diameter [m]	An agent's maximal possible torso diameter	0,46
Perception radius [m]	An agent's perception radius, used to calculate density (default 2.0)	2.0
Comfort distance for origins [m]	Distance two agents may appear in an origin	0.2

Table 4.1: List of Simulation and pedestrian settings available in crowd:it

4.2 Level of Development

The information needed for pedestrian simulations varies depending on which kind of simulation is planned to be conducted. As described in chapter 3.2.1. Comfort studies usually require a higher LOD then evacuation simulations. We chose to use evacuation simulation for this thesis, hence, the simulations demands towards the buildings LOD shift towards the early stages. The simulation's minimal requirements in order to be functional are:

- Exterior and Interior Walls
- Origins, representing a building's populated rooms
- Destinations. Which serve as the buildings exit points
- In case of multilevel buildings: stairs as floor connectors

This thesis will adapt the Building Development Level (BDL) defined in chapter 2.3.1. Abualdenien and Borman (2018) aimed to provide different specialised fields with a consistent definition of uncertainty concept by creating the BDL concept. The BDL suitable for pedestrian simulation is BDL. Abualdenien and Borman define BDL4 as stage in which a more precise definition of the structure is modelled, leading to a definition of

the internal spaces. In this level, the percentage of opening and estimated load can be specified. BDL4 is the first stage defining the inner walls and rooms.

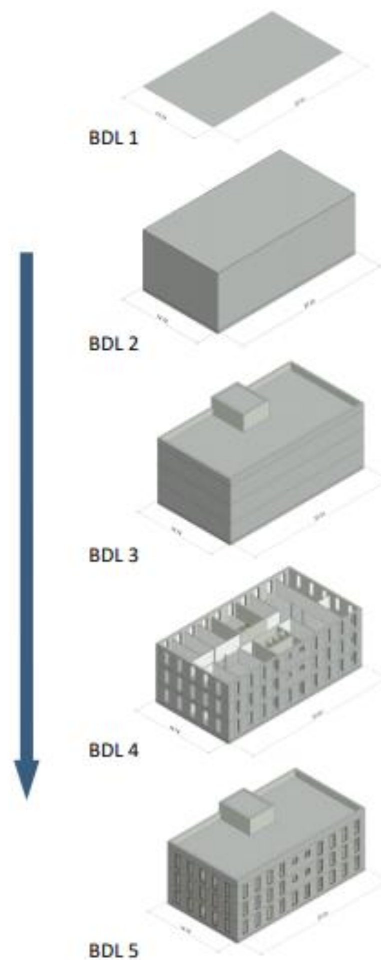

Figure 4.1: BDL stages 1-5. Abualdenien and Borrmann (2018)

4.3 Determination of input Parameters

BDL4 sees the first appearance of the interior's layout and load. Therefore, these parameters underly the highest degree of uncertainty and will be used as input parameters for our simulation. Interior walls, openings and their size and stairs are dominant in dictating an agent's ability to move through the building. It is important to note that the buildings pedestrian load will not be part of our concept of the uncertainty analysis. We assume that it is technically possible to include this parameter as input parameter and assign an uncertainty value to it. However, this would require the manipulation of not only the “. floor” file, but also the “.crowdit” file.

The evacuation simulations key-parameter is the evacuation time. This parameter will function as output parameter. Other possible parameters are the average overall distance travelled by agents or spatial density in simulation objects. Unfortunately, these parameters can not be easily compared by collecting all output data into one large dataset, as they tend to be easily miss-interpreted, since many different factors partake in these results.

4.3.1 The ".floor"-file

Crowd:it's geometrical definition of objects has to be studied before one can define geometric input parameters. Geometry data is stored in the ".floor"-file. The floor-file is stored inside the "geometry" folder, which is part of the "*projectName*_res" directory. For this purpose, the following use-case has been created. The floor-file is of the XML-Document type.

Figure 4.2: Sample-Project

This project consists out of a origin (red-zone) surrounded by five walls (grey-objects). Between the northern walls, a cut has been created (orange). After leaving the room, pedestrians are led to the destination (green) via a hallway, limited by an additional wall in the north.

Drawing1_20190721145102.floor


```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<floor isoDate="2019-07-21T12:51:02.810Z" srcFile="Drawing1.dxf" xmlFormat="0.8">
    <layer id="0">
        <wall closed="true" id="w2">
            <point x="0.157495" y="7.033868"/>
            <point x="1.533224" y="7.033868"/>
            <point x="1.533224" y="0.047573"/>
            <point x="0.157495" y="0.047573"/>
        </wall>
        <wall closed="true" id="w3">
            <point x="1.533224" y="7.033868"/>
            <point x="6.265732" y="7.033868"/>
            <point x="6.265732" y="5.273542"/>
            <point x="1.533224" y="5.273542"/>
        </wall>
        <wall closed="true" id="w4">
            <point x="11.163328" y="7.033868"/>
            <point x="15.895836" y="7.033868"/>
            <point x="15.895836" y="5.273542"/>
            <point x="11.163328" y="5.273542"/>
        </wall>
        <wall closed="true" id="w5">
            <point x="15.895836" y="5.273542"/>
            <point x="14.575136" y="5.273542"/>
            <point x="14.575136" y="0.047573"/>
            <point x="15.895836" y="0.047573"/>
        </wall>
        <wall closed="true" id="w6">
            <point x="14.575136" y="0.047573"/>
            <point x="1.533224" y="0.047573"/>
            <point x="1.533224" y="1.807899"/>
            <point x="14.575136" y="1.807899"/>
        </wall>
        <wall closed="true" id="w7">
            <point x="0.157495" y="10.053227"/>
            <point x="15.895836" y="10.053227"/>
            <point x="15.895836" y="9.181581"/>
            <point x="0.157495" y="9.181581"/>
        </wall>
    </layer>
    <layer id="crowdit">
        <wunderZone id="simObj-129">
            <point x="6.265732" y="7.033868"/>
            <point x="11.163328" y="7.033868"/>
            <point x="11.163328" y="5.273542"/>
            <point x="6.265732" y="5.273542"/>
        </wunderZone>
        <wunderZone id="simObj-133">
            <point x="1.533224" y="5.273542"/>
            <point x="14.575136" y="5.273542"/>
            <point x="14.575136" y="1.807899"/>
            <point x="1.533224" y="1.807899"/>
        </wunderZone>
        <wunderZone id="simObj-155">
            <point x="16.836017" y="9.669702"/>
            <point x="21.544528" y="9.669702"/>
            <point x="21.544528" y="5.973923"/>
            <point x="16.836017" y="5.973923"/>
        </wunderZone>
    </layer>
</floor>
```


eXtensible Markup Language file

Figure 4.3: XML-.floor-file

This file is crowd:it's data-structure derived from the previous geometry. Each floor is represented by one floor-file at a time. A multi-storey building therefore has as many floor-files as storeys. The files root is at "floor" node, containing the file's attributes "isoDate", file-format and "xmlFormat". Its solitary child nodes are of the type "layer". Layer nodes have one attribute named "id", containing the layers names. When reading from ".dxf" or IFC, crowd:it differentiates geometry objects based on their layer's id. Geometry on layers whose names contain the string "crowdit" will be interpreted as

"wunderZone", which represent simObjs. Objects on any other layer which does not contain the "crowd:it" string, is interpreted as type of "wall".

Each object node contains at least one "point" node, consisting of two float values. Each point-node adds one vertex to the object. Two vertices define an edge. Three or more edges define a face.

Wall-objects are created by a single vertex, one edge or a face. These wall-objects are not checked by crowd:it for inconsistencies, crossings or if vertices exist more than once.

SimObjs must be defined by at least one edge (e.g. 2 vertices). Crowd:it also forbids a simObjs to have reoccurring vertices. The only exemption to this is the first and last vertex being allowed but not mandatory to be equal.

4.3.2 Transform-Operations

Importing geometry files from ".dxf" or IFC results in a loss of information, since the floor-file only contains raw geometrical data. Any relation between objects is lost, as it is not necessary for the simulation kernel to know about the relations between different geometry objects. Consequently, relations must be reinterpreted by defining rules for neighbouring objects to behave similar as applications such as Autodesk Revit would cause them to. Moving walls in Revit causes attached objects to be influenced in a defined matter. Objects can be depended on each other, meaning one objects movement moves the other object entirely. They can also be partially depended on each other, causing minor changes when their neighbouring object is moved. And then there are static objects, which can not be influenced by other objects at all. To mimic this behaviour we now have to define the set of rules for inter object relations.

The definition of the Building Development Level provides walls, doors and stairs as geometrical type of objects. Assuming an interior-wall's position had some fuzziness value regarding its position on the y-axis. For instance, the wall could be moved by 0.5 meters in positive y direction. How do neighbouring and or attached walls behave in this case and what should their relations be?

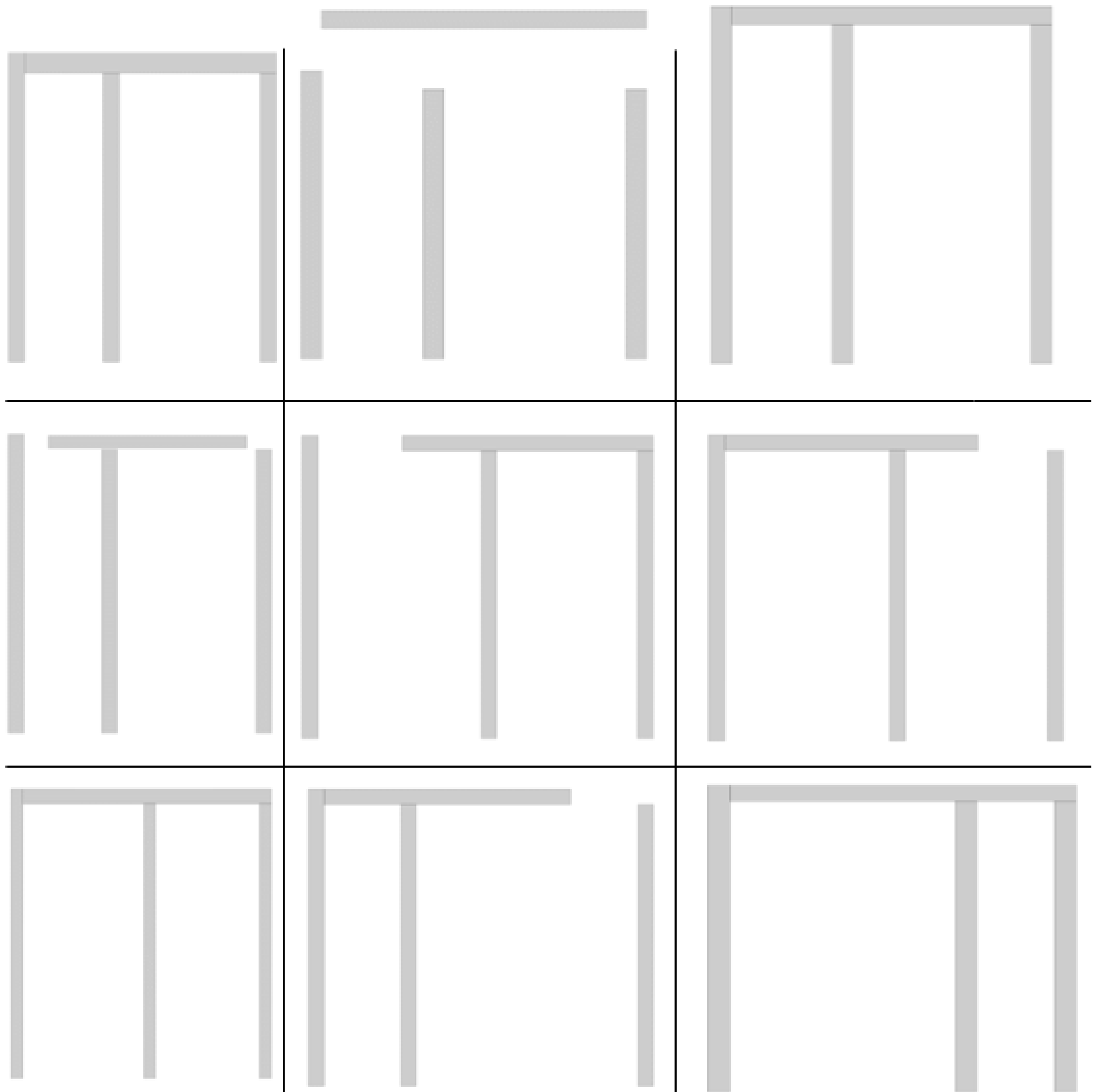

Figure 4.4: Possible outcomes of moving and scaling neighbouring walls

In Figure 4.4 the problematic information loss is being portrayed. The left-top corner shows the original geometry. In the window to its right, the top wall has been moved in positive y direction. The walls neighbouring were interpreted as unattached. This led to holes in the geometry being created. Further to the right, the walls were interpreted as attached, therefore, the attached edges are also moved up to the top. However, it would also be plausible if the attached walls were moved in the positive y direction as a whole.

The second row demonstrates the possible interpretations of uniformly scaling the top wall by 0.8. The last row describes variations of moving the leftmost wall in the negative x direction. Some of these variations are logically more convincing than others, ultimately it is the developer responsibility to define his own set of rules to avoid unexpected behaviour.

In the third row, the leftmost wall was moved in the negative direction of the x axis. Possible outcomes are depicted in the following two windows.

The first step is to define different operations that can be applied to input objects. Computer graphics defines 3 types of transformations: translation, scaling, rotation. The latter is excluded from further discussion, as this would create additional problems and increase complexity. Scaling also has many different variations- as one could scale an object in any direction or uniformly and be using any possible point as basepoint. Nevertheless, scaling is included in our thesis concept, considering a doors or stairs is of major interest for influence on pedestrian flow-rates. Transformation leaves less room for different approaches and is also of high interest, hence it is also included into the concept.

In any further consideration, all edges are assumed to be parallel to either the *x* or *y* axis, which vastly reduces complexity of our concept. This in mind, the following ruleset was developed.

4.3.2.1 Scaling

Scaling is only to be allowed for doors, stairs and free-standing walls. The scaling direction is depended of the object.

Doors will be scaled along their long axis, which is derived from the floor-file since no such information is existent in crowd:it. The scaling will affect the wall which is cut through by the door. The edges/vertices shared between wall and door will be moved

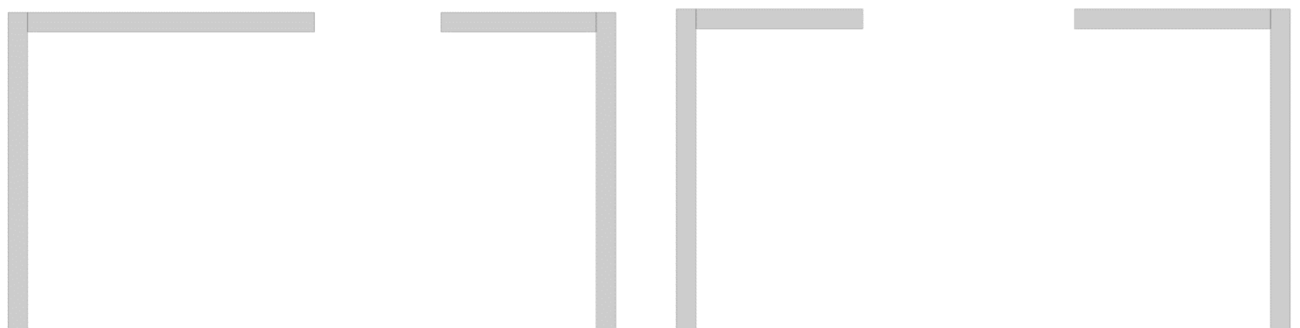

Figure 4.5: Scaling and moving a door

by the same distance, as demonstrated in figure 4.5. Doors also are assumed not to be positioned next to neighbouring walls running orthogonally to them.

Stairs could be scaled along both axis, and potential demand for both exists. Nonetheless, stairs will only be scaled against their walking direction. Otherwise the crowd:it file would be required to be manipulated as well, since the tread-depth or number of treads would be likely to change. Stairs which are moved or scaled are assumed to be free-standing, not directly neighbouring other simulation or geometry objects.

Freestanding walls are scaled in the direction of their long axis, as scaling along the short or “thickness” axis would only result in minor increases, unlikely to have effects on simulation results.

4.3.2.2 Translating

Translation is allowed for all objects. Stairs once again must be freestanding. Translating doors functions like scaling, influencing the cut-through walls.

Three rules were developed, to created expectable behaviour of attached and neighbouring objects, similar as to the ones seen in Revit. When a wall is moved all its vertices will be moved by the same amount and in the same direction.

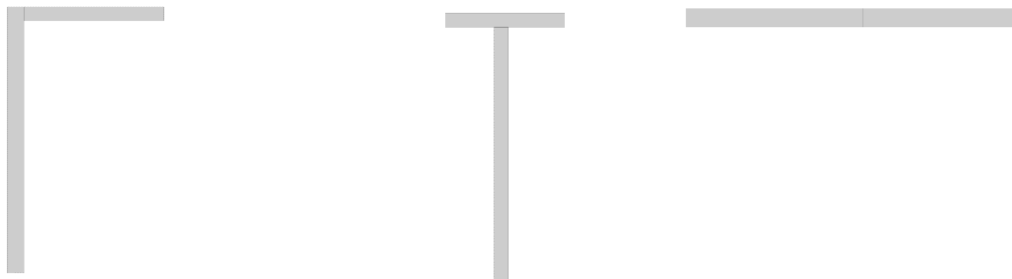

Figure 4.6: The three different cases of neighbouring walls defined for the concept

- If any wall B shares an identical edge with the moved wall A, wall B will also be moved. This procedure is then to be rerun, to check if wall C shares an edge with B, until all walls which are in a row are detected.
- If wall B’s edge is contained in one of the moving wall A’s edges and the edge which is contained is orthogonal to the movement of wall A, then the contained edge will also be moved by the same amount.
- If wall B shares an vertex with the moving wall A, then the edge which is positioned at this vertex and also orthogonal to wall A’s movement is moved by the same amount as wall A.

4.3.3 Fuzziness

It is noteworthy that the floors discretization dictates the granularity of the simulation's interpretation of geometry. This can cause small spans of uncertainty to not affect appear in the simulation at all. The min and max fuzziness values of an object are chosen by the end-user. Fire-Safety regulations or law such as the German "Versammlungsstätttten Verordnung" could be used as guidelines for the span of fuzziness. The fuzziness values are of constant probability and discrete distribution. We introduce the term "Depth of uncertainty" to define the increment used between min and max value.

4.4 Manipulating the floor-file

4.4.1 The Crowdit File

The crowd:it file is built up as shown in the graphic.

```
simpleSimulation.crowdit
1  <?xml version="1.0" encoding="UTF-8"?>
2  <scenario xmlFormat="0.8" name="simpleSimulation" isoDate="2019-07-21T10:01:04.549Z">
3      <spatial>
4          <floor id="Drawing1" floorAt="geometry/Drawing1_20190721145102.floor" />
5      </spatial>
6      <meta>
7          <intervals>
8              <interval id="i0-origin-0" value="0,1:1" />
9          </intervals>
10         <intervalMatrix>
11             <row>origin-1,i0-origin-0</row>
12         </intervalMatrix>
13         <paths />
14         <backgroundImages />
15         <pedCharacters>
16             <type name="FPLMWalking" ratio="1.0" />
17         </pedCharacters>
18         <morphosis>
19             <destination id="destination-1" wunderZone="simObj-155" disableDynamicFlooding="false" />
20             <origin id="origin-1" wunderZone="simObj-133" minPremovementTime="0" sortedBirthCells="false" maxPremovementTime="0" />
21         </morphosis>
22         <sets />
23         <pathSnippets />
24         <floorProps>
25             <floorProp floor="Drawing1" cellDist="0.1" discretizationType="1" elevation="0.0" height="2.0" />
26         </floorProps>
27         <groups>
28             <row>1,1.0</row>
29         </groups>
30         <elevatorMatrices />
31     </meta>
32     <settings />
33     <evaluations>
34         <measurements />
35     </evaluations>
36     <visualization>
37         <colorings />
38     </visualization>
39 </scenario>
40
```

Figure 4.7: The ".floor"-file

The crowd:it file contains all simulation related information. The root-node is the "scenario node, containing information about the XML-Format, the projects name, and isoDate.

The next node is the “spatial” node. It has two attributes: “id”, which got the name of the origin “.dxf” or the imported “IFC” file, and “floorAt”, which links the Floor-XML and the simulation.

This is followed by the “meta” node and its children, as well as the “settings”, “evaluations” and “visualization” nodes.

A relevant node is the “meta” node, and its child node “morphosis”. This node assigns functionality to SimObjs which were defined in the “.floor” file. The first of its nodes, called “destination”, links the SimObj “destination-1” to the wunderZone=”simObj-155”, found as last object in the “.floor” file. The linking between the geometry and its functionality as SimObj goes by the objects id, which allows for the geometry of simObjs to be manipulated even after they are created. To not destroy the connection between the simObjs in the “.floor”file and the “.crowd:it” file the objects name must not be changed, as this would cause the simulation to crash.

The linking between the “.floor-“ and “.crowdit”file allows for the creation of projects with the same crowdit file but differing floorfiles, as long as the floors and simObjs names are not changed. To create multiple variations the complete project folder hasto be copied for each individual iteration. This ensures, that neither the simulation settings or the floor-files name changes. After the files are copied the geometry can be changed at desire, provided that the simObjs remain unchanged.

5 Implementation

The following implementation is based on the concept from previous chapter. The implementation was created using Visual C# 4.0 .NETFramwork Version v.4.6.1 . The application includes the System, System.Collections.Generic, System.Reflection, System.Numerics, System.Xml, System.IO, and System.Globalization namespaces.

5.1 File Structure

The application folder structure is shown in figure 5.1. Notable for usage are the “inputXML” folder and the “projects” folder. Inside the projects folder is a folder called: “baseProject”. At this location is the project which holds the floor-file to be manipulated and the .crowdit file. For every variation of the base simulation the “baseProject” folder is copied into this directory and the copy renamed to the variations name. The names of the simulation and floor-files remain unchanged enabling crowd:it to link the new floor-files to an existing simulation. “project#_res” contains the simulations results inside a folder called “out” and the geometry files. The “intputXML” is introduced as user input file. This file is a basic XML-file. Declaring an object as part of the uncertainty

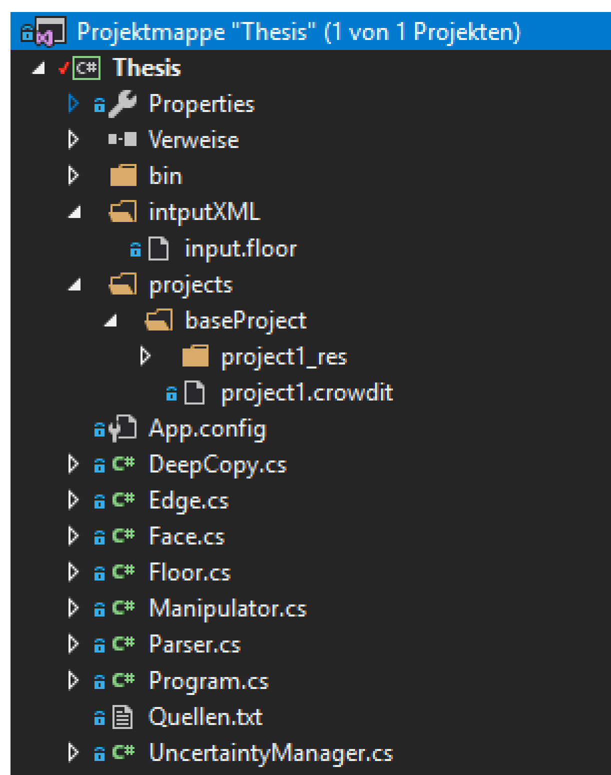

Figure 5.1: The concepts folder structure, pre early stage simulation

analysis requires the objects “id”, which has to match the id in the “.floor”-file, not the “.crowdit”-file. Objects can either have a “Move” or “Scale” attribute.

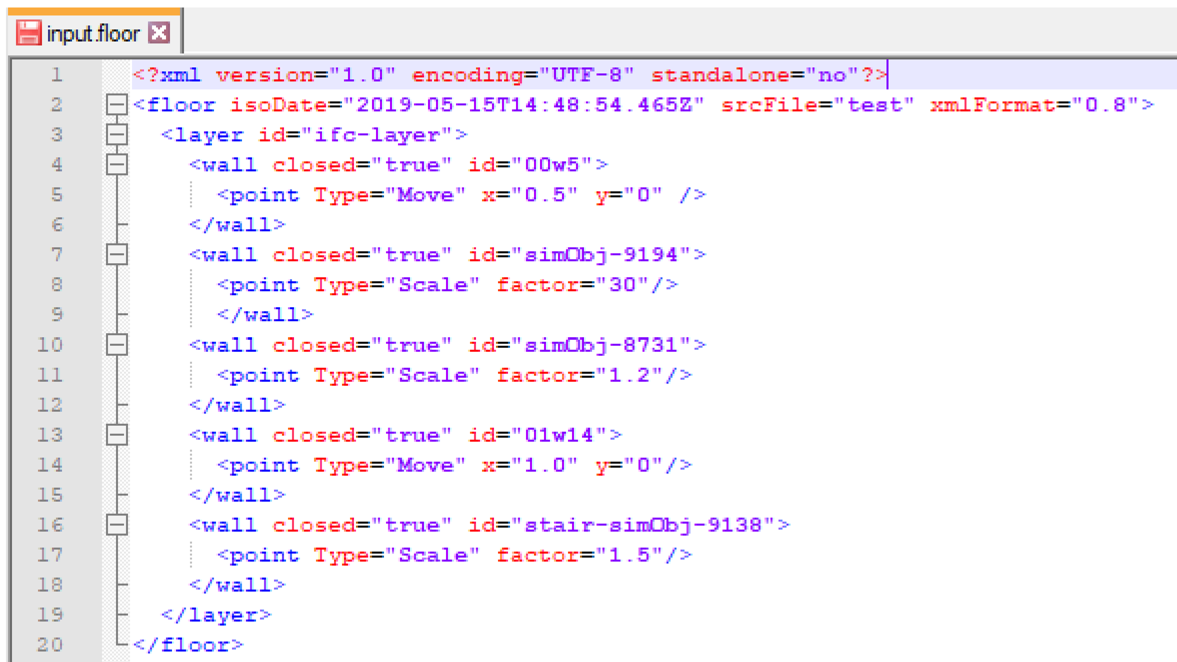

Figure 5.2: Input Xml-File

5.1.1 File

Inside our application we will have a specific location for our base project- containing the floorfile(s) we wish to manipulate. Inside the input folder is the input.XML document, being build up similar to the “.floor”-file. This document has a node for each object of our .floorfile which we plan on altering during our uncertainty analysis. Since every id has to be unique and has to remain unchanged in order for crowd:it to recognise the changed object correct, the linking between the objects will happen the same way as its done by crowd:it. Each object in the input file can have up to two attributes, a point attribute serving as vector, which will be the object’s *moveVector*, and one attribute serving as *scale*. Those value represent the max. uncertainty value, with each iteration moving/scaling the object further until its max value is reached. After reading-in the floor- and input-file, a new “.floor”-file with the exact same name for each iteration of the uncertainty analysis will be created. This file will be placed into an exact copy of our “baseProject” folder and will replace the old floorfile in this folder. After creating the new project with its new “.floor”-files, the crowd:it kernel is called to conduct simulation on this object.

5.2 Classes

The following chapter will describe the applications classes and their intended use and concept behind some implementations. Not all attributes and functions are mentioned but only the most important ones.

5.2.1 Geometry

The implementation uses a modified from of the boundary representation to interpret and represent geometrical objects. Faces are described by edges and vertices. Each face has a list of vertices, and each vertex has a list of faces it is referenced by. Faces must consist out of at least three edges, respectively three vertices.

5.2.1.1 Vertex.cs

The vertex class handles the pure geometric information. Vertices are constructed by a DoubleVector2. The C# *System.Numerics* namespace does not support double Vectors. But CAD applications such as AutoCAD store geometric information in double precision. Hence, the *System.DoubleNumerics* was added to the project.

Vertices are designed to keep track of affected objects. Each Vertex has a list of faces which contain this vertex. when reading the “.floor”-file, the parser will ensure that each vertex is unique. When one object is moved, it also automatically affects other objects sharing the same vertex. In rare cases this can lead to undesired behaviour. Most times when a room is modelled as simObj and shares a common vertex with a wall that is moved.

5.2.1.2 Edge.cs

The Edge class is designed to hold two vertices for each edge. Edges are part of the attribute of the face class. They were implemented to enable checking, whether faces are attached to each other and if two faces share a common edge.

5.2.1.3 Face.cs

Faces are made up by 3 or more vertices. Each face contains a list of its vertices. Faces can be of different element types: “wall, room, stair, door”. These element types mainly determine face’s behaviour when scaled. Walls and doors are scaled along their long axis, stairs in direction of their short. Rooms do not have special implementation,

neither are there any possibilities to directly influence their shape. However, rooms can indirectly be reshaped by dragging out our pulling in their limiting outer walls.

The Face class contains the *CalculateMidpoint()* class. The midpoint is needed to calculate the directional scale. A polygons midpoint can be determined by calculating the geometric centre of mass.

$$s = \frac{1}{m} * \sum_{i=1}^{m} x_i$$

With *m* being the number of unique points (vertices) and *x* each vertices coordinates.

```
1 reference | Janik Scholl, Vor 7 Stunden | 1 Autor, 2 Änderungen
public Vector2 CalculateMidPoint()
{
    Vector2 midpoint = new Vector2();
    double sumx = 0, sumy = 0;
    int count = vertices.Count;
    for(int i=0; i<count; i++)
    {
        sumx += vertices[i].ToVector2().X;
        sumy += vertices[i].ToVector2().Y;
    }

    if(vertices[0]==vertices[count-1])
    {
        sumx -= vertices[count - 1].ToVector2().X;
        sumy -= vertices[count - 1].ToVector2().Y;
    }

    midpoint.X = (1.0f / (count)) * sumx;
    midpoint.Y = (1.0f / (count)) * sumy;

    return midpoint;
}
```

Figure 5.3 CalculateMidpoint() Function:

5.2.1.4 Floor.cs

A ".floor"-file's data is equivalent to its corresponding floor object in the code. Each floor holds a list of all its faces, edges and vertices. It also counts the faces which have fuzziness values assigned and has a function to set the *moveVectors* and *scaleValues* of these faces, according to the variation used.

5.2.1.5 Variation.cs

The Variation class is one the project's main classes. For each variation of the early stage simulation, a variation object is constructed. A variation's object contains each

of the building's floors and herby its faces, edges and vertices. The *uncertaintyManager* advices the variation object which of the variation is represented by it. With this information, the variation object then modifies the *moveVector* and *scale* of the objects which underlay a degree of uncertainty in the current variation.

5.2.2 Parser.cs

The Parser class is a static class. Its prime function is the *parseFloor* function. Its input parameters are a *path string, filestatus, variation object* and *floor*.

ParseFoor first evaluates if it was set into *read, write* or *input fileStatus*. It then loads the appropriate *XmlDocument* and iterates through its nodes until it reaches the wall or simObjs nodes. At this point a *tempFaces object* is created.

```
4 references | Janik Scholl, Vor 11 Minuten | 1 Autor, 6 Änderungen
public static void ParseFloor(string path, FileStatus fileStatus, Variation variation, Floor floor)
{

    //Floorfiles are red from the xmlDoc object
    //Input files are red from the xmlInput object
    XmlDocument tempXmlDoc = Globals.xmlDoc;
    if (fileStatus == FileStatus.input)
        tempXmlDoc = Globals.xmlInput;

    //iterate for each objectType seperatly
    foreach (string objectType in Globals.objectTypes)
    {
        //iterate through all layer-nodes, only read from the layer node containing the current objecType
        XmlNodeList genericNodes = tempXmlDoc.SelectNodes("//floor/layer/" + objectType);
        {
            foreach (XmlNode genericNode in genericNodes)
            {

                //Create a tempFace object with the nodes "id"
                Face tempObj = new Face(genericNode.Attributes["id"].Value, ref variation, SetObjectType(genericNode.Attributes["id"].Value));

                //iterate through each genericNode's childNodes
                XmlNodeList pointNodes = genericNode.ChildNodes;
```

Figure 5.4: The Parser

A switch case structure then evaluates the next step. If *fileSatus=read,* the application enters the *read* state.

Firstly, a *vertexList* is created to hold all vertices parsed from the *pointNodes.* In this state the parser iterates trough all *pointNodes.* In each iteration a *tempVertex* object is created. The *pointNodes* attribute values are parsed into two double variables. If the current floor does not already contain a vertex with these coordinates, a new vertex is created and added to the *vertexList.* If such vertex already exist, the existing vertex is added to the *vertexList.* After iterating through each *pointNode,* a new face object is created. The face then gets assigned to *vertexList* and is added to the variations face list. Once all nodes are parsed, the application breaks and returns to the *main* function.

```
case FileStatus.read:
    {
        //Create list to contain all vertices created by an wall- or simObjs node
        List<Vertex> vertexList = new List<Vertex>();

        //Iterate through each point Node
        foreach (XmlNode pointNode in pointNodes)
        {
            //Create tempVertex object
            Vertex tempVertex = new Vertex();

            //Parse the attributes x and y values
            double x = double.Parse(pointNode.Attributes["x"].Value, Globals.culture);
            double y = double.Parse(pointNode.Attributes["y"].Value, Globals.culture);

            //Check if a vertex with these x/y values already exists
            //if yes, add the existing vertex to the vertexList
            if (variation.FindVertex(x, y) != null)
                vertexList.Add(variation.FindVertex(x, y));
            //if not, create a new Vertex, then add it to the vertexList
            else
            {
                tempVertex = new Vertex(x, y);
                variation.AddVertex(tempVertex);
                vertexList.Add(tempVertex);
            }
        }
        //Create new face object, asign the vertex list to it //add the face to the variations and floors faceLists
        Face face = new Face(genericNode.Attributes["id"].Value, ref variation,SetObjectType(genericNode.Attributes["id"].Value));
        face.AddVertex(vertexList);
        variation.SetFacesVertices(face);
        floor.AddFace(face);
        break;
```

Figure 5.5: The read status

In the write status, the parser iterates through all *pointNodes* and assigns them their new value. To keep track which vertex is to be red, an iteration variable is incremented every time a *pointNode* has been iterated through. The iteration variable is then used as index, to call the corresponding vertex from the *VertexList*. The function exits once all *pointNodes* are parsed.

```
case FileStatus.write:
    {
        //keep track of the iteration
        int i = 0;
        foreach (XmlNode pointNode in pointNodes)
        {
            //Parse the vertexValues at index i to the pointNodes
            pointNode.Attributes["x"].Value = floor.GetFace(genericNode.Attributes["id"].Value).GetVertices()[i].ToVector2().X.ToString().Replace(",", ".");
            pointNode.Attributes["y"].Value = floor.GetFace(genericNode.Attributes["id"].Value).GetVertices()[i].ToVector2().Y.ToString().Replace(",", ".");
            i++;
        }
        break;
    }
```

Figure 5.6: The write status

Lastly, the parser can be put into the *inputStatus.* In this status, the parser searches the *genericNodes* for matches with a floor's faces via their names. If a match is found, the floor's *facesToMove* counter is incremented by one. Then either the face's *moveVector* or *scale* get their new value assigned.

```
case FileStatus.input:
    {
        //Iterate through each floor's face
        foreach (Face face in floor.GetFaces())
            //ckeck if one of the genericNodes ids is euqal to the faces name
            if (face.GetFaceName() == genericNode.Attributes["id"].Value)
            {
                //If positive, increment the facesToMove counter
                floor.AddFacesToMove();

                //iterate through the pointNodes
                foreach (XmlNode pointNode in pointNodes)
                {
                    //if the attributs value is "Move", set the faces moveVector
                    if (pointNode.Attributes["Type"].Value == "Move")
                    {
                        float x = float.Parse(pointNode.Attributes["x"].Value, Globals.culture);
                        float y = float.Parse(pointNode.Attributes["y"].Value, Globals.culture);
                        face.SetMoveVector(new Vector2(x, y));
                    }
                    //if the attributes value is "Scale" set the faces scaleVector
                    if (pointNode.Attributes["Type"].Value == "Scale")
                    {
                        float scaleFactor = float.Parse(pointNode.Attributes["factor"].Value, Globals.culture);
                        face.SetScale(scaleFactor);
                    }
                }
            }
        break;
    }
```

Figure 5.7: The input status

5.2.3 UncertaintyManger.cs

This static class handles the variation problem. It takes the depth of the given uncertainty and *numberOfFacesToMove* to calculate the number of possible variations.

$$possibilities = depth^{numberOfFacesToMove}$$

It also calculates and sets up all possible variations of the Monte Carli Method.

The uncertainty manager also gets called to set the *moveVector* and *Scale* for each object accordingly to the current variation.

5.2.4 Manipulator.cs

The manipulator is the other main class after the parser class. It is called by the program class. The manipulator is handed the variation object, which at this point has a list of all floors, each floor's faces and vertices as well as the modified move and scale values set by the uncertainty manager in the previous step.

Calling the *Translate()* function, the manipulator iterates through each object which has a *scaleValue* or *moveVector* and calls the *Translate*() or *Scale()* function accordingly.

To translate, a list of *verticesToMove* is created. The object's (face A) vertices, which is to be moved, are then added to this list.

The *AddAttachedToMove* function is called, which iterates through all other faces on the same floor. It checks if any face (face B) from the floor's *faceList*:

1. Was already moved. If so, the application returns and the next face is checked.
2. For each of face's B edges it is checked if:
    a. The current edge is parallel to the *moveVector* of face A.
        i. If true, the application determines if face A shares an edge with face B.
            1. If true, all of face's B vertices are added to the *vertices-ToMove* list. And *AddAttachedToMove* is called by face B, repeating the procedure of checking if any face C is attached to B.
    b. The current edge is orthogonal to the *moveVector* of face A.
        i. If true, it the function tests if this edge is contained by any of face's A edges.
            1. If true, only the contained edge's vertices get added to the *verticesToMove* list.

After this, the function returns and all vertices in the *verticesToMove* list are moved by the same vector.

5.2.5 Program.cs

The program class is the interface for mainly the Manipulator, UncertaintyManager, Variation and Manipulator classes. Its input data is the *inputXML* and the simulation and its data inside the *baseFolder*. The Parser is called to parse the input both input elements. Next, Variation and the UncertaintyManger are instantiated. The *baseFolder* is copied and renamed after each Variation instance. The Variations *moveVectors* and *scaleFactors* are set by the UncertaintyManger and finally the Manipulator gets called to transform the different variations. The Parser then saves the newly created before the simulations are started. After reaching this point, program terminates the application.

Figure 5.8: The program class's process

6 Evaluation

6.1 Setting up the Simulation.

The Revit file of the HWH meets the standards of BDL4.The file was exported from Revit into AutoCAD. Any objects which are not of interest for the simulation were removed. The file was then manually adjusted according to the concept's definition from chapter 3. This mainly resulted in the removal of columns between walls and editing door objects. In the buildings northern area, a stair to be scaled and moved was added.

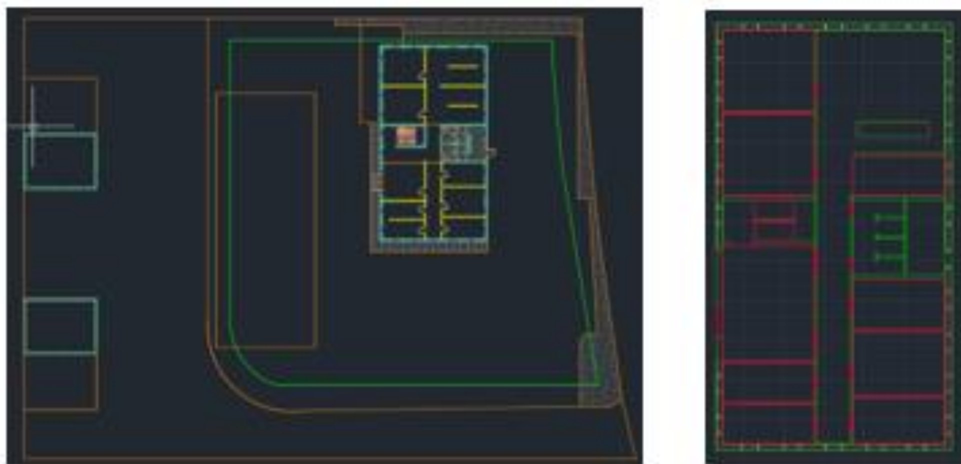

Figure 6.1: dxf-File before (left) and after (right) the setup

Each room in the simulation is occupied by 8 agents.

- Agents from the second-floor search for the closest stairway to the first-floor
- Agents on the first-floor search for the closet stairway to the ground-floor
- Agents on the ground floor search for the fastest way to the closest exit

6.2 Input Parameters

We will examine the systems sensitivity towards the fuzziness of five operations.

- One the ground floor, we will examine the impact of the "WesternDoor's" (1) size. The range of uncertainty spans from 2.0 meters up to 2.6 meters (±20%)
- Also on the ground floor, we create an additional exit which we will call "North-Door", its uncertainty range spans from 0.1 to 1 meters (2)
- In addition, we narrow the "Lvl00-HallwayWidth" (3). Its min, max fuzziness spans from 1.50m to 2.00m.

- At the first floor, the Lvl01-HallwaysWidth (4) lays between 1.0 and 2.0 meters (±33%)
- The width of “Stair-lvl02-lvl01” ranges between 0.8 and 1,2 meters (±20%).

The output value is the buildings evacuation time. Each variation is calculated 5 times. This results in 1035 simulations runs. On the authors machine, this calculation took about 5 hours.

Figure 6.2: The buildings floorplans in crowd:it.

6.3 Results

The input parameters were changed every simulation after the following scheme:

- A: “Lvl00-HallwayWidth” was changed every 84th simulation run
- B: The “WestDoor” was changed every 28th simulation run
- C: The “NorthDoor” after the 9th simulation
- D: “Lvl01-HallwayWidth” every 3th simulation
- E: And Stair-lvl02-Lvl01 every simulation

Type	Time [m]:	Lvl00-HallwayWidth	WestDoor	NorthDoor	Lvl01-HallwayWidth	Stairl-lvl02-lvl01
Slowest	2,08	1.5m	2.00m	0.50m	1.00m	1.20m
	2,07	1.75m	2.00m	0.10m	1.00m	0.80m
	2,00	1.75m	2.00m	0.50m	1.50m	1.20m
	1,97	2.0m	2.00m	1.00m	1.00m	1.20m
	1,97	1.5m	2.00m	1.00m	1.00m	1.20m
Fastest	1,43	1.50m	2.60m	1.00m	1.00m	1.20m
	1,45	2.00m	2.60m	0.50m	2.00m	1.20m
	1,45	1.75m	2.60m	0.00m	1.00m	1.20m
	1,45	1.50m	2.60m	0.00m	1.50m	0.80m
	1,45	1.50m	2.60m	1.50m	2.00m	1.20m

Table 6.1: Evacuation times and the uncertain parameters

Table 6.1 shows the five slowest and the five fastest evacuations. "WestDoor's" size is the most influential input parameters in regard to evacuation time. In any of the fastest evacuation runs it was at set to its max uncertainty value, creating the largest opening possible. Additionally, in each of the slowest 5 runs, it was at the minimal value, indicating a bottleneck at the exit door.

Comparing the slowest and fastest evacuation times, the best layout is 25% more efficient than the "slowest" layout.

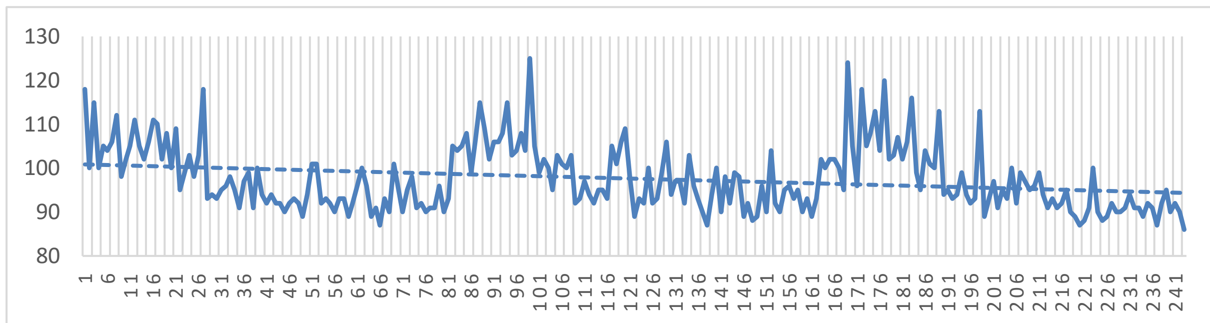

Figure 6.3: Diagram of the evacuation times for each variation, sorted in order: Parameter A, B, C, D,E

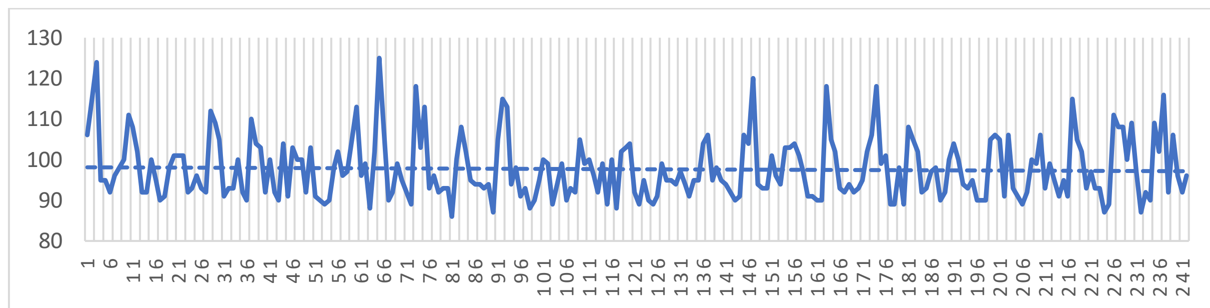

Figure 6.4: Diagram of the evacuation times for each variation, sorted in order: Parameter B, A, C, D,E

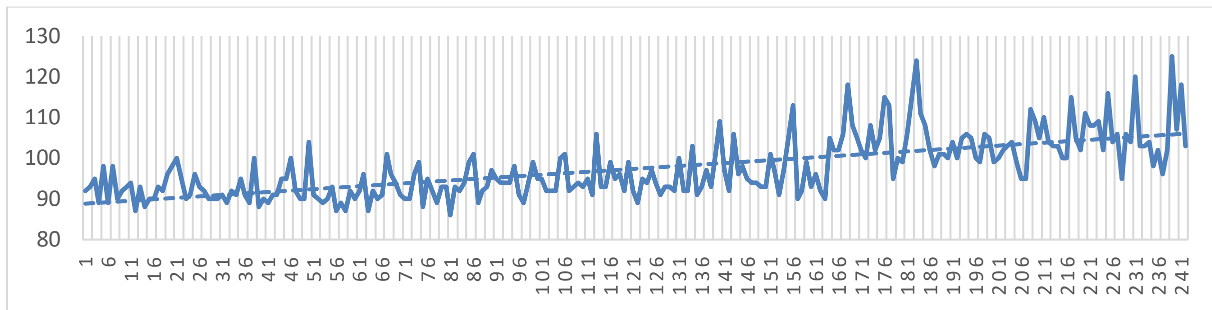

Figure 6.5: Diagram of the evacuation times for each variation, sorted in order: Parameter C, A, B, D,E

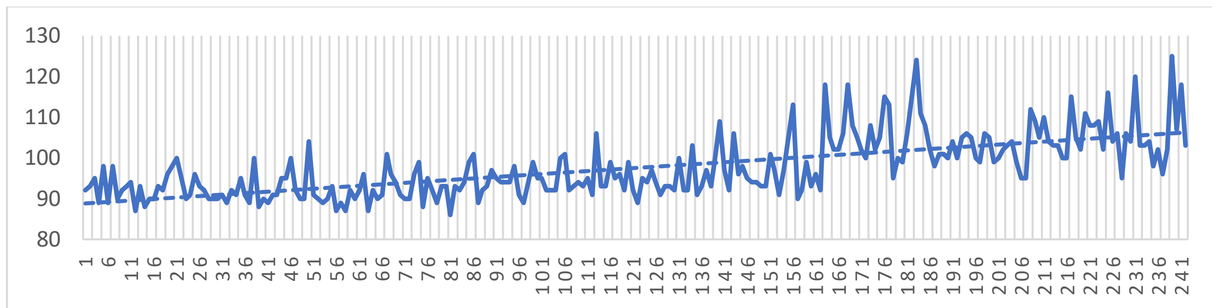

Figure 6.6: Diagram of the evacuation times for each variation, sorted in order: Parameter D, A, B, C,E

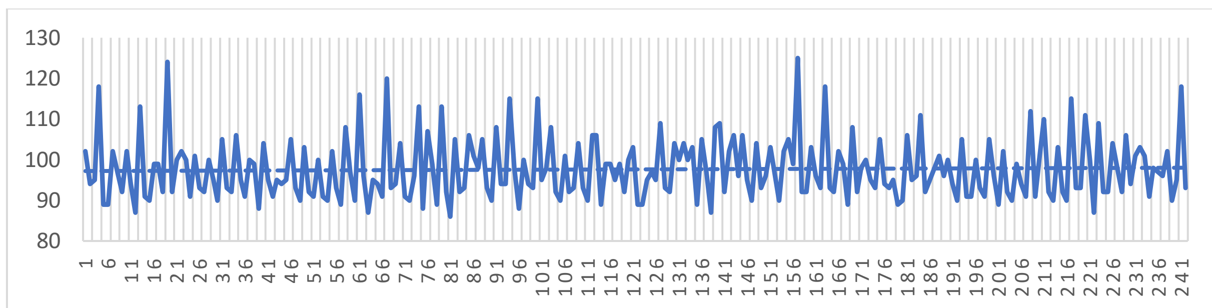

Figure 6.7: Diagram of the evacuation times for each variation, sorted in order: Parameter E, A, B, C, D

According to figure 6.4 and 6.7, parameter B (Lvl00-HallwayWidth) and E (Stair-lvl02-lvl01) do have little impact on the evacuation progress. Their graphs are trend does not significantly change over the different variations.

Parameter C(“NorthDoor”) and D(“Lvl01-HallwayWidth”) have negative impact on evacuation time when approaching max. uncertainty value (Figure 6.5 and 6.6).

Parameter A (“WestDoor”), seen in Figure 6.3 does have positive influence on the building’s evacuation time, judging by its trendline.

7 Conclusion

In previous chapters we developed a concept for early stage pedestrian simulations and studied the various methodologies of uncertainty analysis. After defining our own set of rules and use cases, we were able to design a basic, automated uncertainty analysis tool. We used this tool to conduct a pedestrian simulation of a building's early stages design. We assume the research goal to be fully satisfied.

The concept was successfully tested in chapter 6. We could proof that our concept is able to produce meaningful results by analysing a buildings crucial structural component and their influence on pedestrian movements. With the knowledge gained from the results in chapter 6, we could design the buildings layout after the best variation and push this design to the next design stage. However, here lays the concepts greatest weakness. It is impossible to transfer the variations model back into building design tools such as Revit. We do not believe such a tool will ever be developed, since too much information has been lost parsing the model into crowd:it. To overcome this information loss was the research's greatest single challenge.

Whilst the concept was a success, we do not see a potential for further development of early pedestrian simulation directly inside crowd:it. Instead, we propose to focus on creating tools that allow for BIM tools to create different variations which can be directly simulated by the desired simulation software. Future research might also focus on ways to conduct pedestrian simulations directly in BIM software. This would negate any loss of information which otherwise could happen during transferring files from one application to another, as it happened in this thesis with crowd:it.

Additionally, we noticed in chapter 6 that carrying out multiple simulations consumes large amounts of computation resources and time. The number of simulations runs grows exponentially with each additional simulation object and degree of uncertainty, increasing calculation time even more. Hence, we propose research to examine if simpler simulation models, such as the Hydraulic-model introduced in chapter 3 could be used in early stage simulations.

8 Table of Literature

Abualdenien, J., & Borrmann, A. (2019). A meta-model approach for formal specification and consistent management of multi-LOD building models. *Advanced Engineering Informatics*, pp. 135-153.

accu rate. (2019, 08 01). *accu:rate GmbH, Institute for crowd simulation*. Retrieved from https://www.accu-rate.de: https://www.accu-rate.de/en/software-crowd-it-en/

BimForum. (2019, 08 02). Retrieved from 2018 level of development specification guide: https://bimforum.org/lod/

Bogenstätter, U. (2000). Prediction and optimization of life-cycle costs in early design. *Buildijng Reasearch & Information*, pp. 376-386.

buildingSMART e.V. (2019, 07 31). *buildingSMART.* Retrieved from buildingSMART know-how standarts: https://www.buildingsmart.de/bim-knowhow/standards

European Commission. (2019, 07 31). *ec.europa.eu*. Retrieved from https://ec.europa.eu/clima/policies/strategies/2050_de

Harter, H., Schneider, P., & Lang, W. (2018). The Energy Grey Zone – Uncertainty in Embedded Energy and Greenhouse Gas Emissions Assessment of Buildings in Early Design Phases. *IALCCE 2018.* Ghent: International Symposium on Life-Cycle Civil Engineering.

Hygh, J., DeCarolis, J., Hill, D., & Ranjithan, R. (2012, 09). Multivariate regression as an energy assessment tool in early building design. *Building and Envoirement*, pp. 165-175.

IPCC. (2018, 08 03). *https://www.ipcc.ch/.* Retrieved from https://report.ipcc.ch/sr15/pdf/sr15_spm_final.pdf

Kneidl, D. A. (2013). *Methoden zur Abbildung menschlichen Navigationsverhaltens.* München: Technische Universität München.

Krygel E, N. B. (2008). *BIM: successful sustainable design with buliding information modelling.* Idianapolis: Wiley.

NIBS, C. o. (2008). Commitee of the National Institute of Building Sciences.

Plum, A., & Jäger, G. (2011). Evakuierungssimulationen im Rahmen von Sicherheitskonzepten von der Konzeption bis zur Realisierung an Beispielen. In P. Andreas, & G. Jäger, *Evakuierungssimulationen im Rahmen von Sicherheitskonzepten von der Konzeption bis zur Realisierung an Beispielen* (p. 10). Aachen: BFT Cognos.

RIMEA. (2019, 08 01). *RiMEA e.V.* Retrieved from Richtline für Mikroskopische Entfluchtungsanalysen: https://rimeaweb.files.wordpress.com/2016/06/rimea_richtlinie_

Ritter, F. (2011). Untersuchung der Möglichkeiten und Vorteile des modellgestützten kooperativen Planens anhand von Autodesk Produkten. München.

Rumnici, M., & Abualdenien, J. (2019, August 01). The influence of design uncertainties in annual energy consumption. *Proc of the 31. Forum Bauinformatik.* Berlin: TU Berlin.

Seits, M., & Köster, G. (2016). *Simulating pedestrian dynamics: Towards natural locomotion and psychological decision making.* München: Technische Universität München.

Shawney. (2014). *Internation BIM Implenentation Guide (1st ed).* London: Royal Institution of Chartered Surveyors (RCIS).

Singh, M. M., Singaravel, S., & Geyer, P. (2018, August 01). Information Exchange Scenarios between Machine Learning Energy. *The Sixth International Symposium on Life-Cycle Civil Engineering*, pp. 487-494.

Weidmann, U. (1993). *Transporttechnik der Fussgänger: Transporttechnische Eigenschaften des Fussgängerverkehrs.* Zürich: IVT.

Appendix A

Plans and figures

Appendix B

Auf der beigefügten CD befindet sich folgender Inhalt:

- Der schriftliche Teil der Arbeit als Worddokument
- Die Daten des Projektes
- Der Source-Code der Anwendung

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 11. August 2019

Janik Scholl

Janik Scholl