



DISSERTATION

Agentenbasierte Simulation von Personenströmen mit unterschiedlichen Charakteristiken

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften unter Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker
Institut für Analysis und Scientific Computing

eingereicht an der Technischen Universität Wien
bei der Fakultät für Mathematik und Geoinformation

von

Dipl.-Ing. Martin Bruckner

Matrikelnummer: 0226452

Guglgasse 8/4/19

1110 Wien

Datum

Unterschrift

Kurzfassung

Die dynamische Nutzung von Gebäuden oder ganzer Stadtteile durch den Menschen wird immer wichtiger. Gebäude und Städte werden nicht mehr statisch, sondern als dynamisches System wahrgenommen und genutzt. Aus diesem Grund werden in gleichem Maße Simulationsmethoden immer wichtiger, die das Zusammenspiel einer großen Zahl an Individuen und den gebauten Strukturen zu simulieren hilft.

Sowohl für die tägliche Nutzung wie auch in Evakuierungssimulationen von Gebäuden werden deshalb bessere Methoden zur Fußgängerflusssimulation und darauf basierende Personenstromanalysen benötigt. Tragische Ereignisse in der Vergangenheit zeigten auf, wie rasch sich große Menschenansammlungen zu tödlichen Fallen entwickeln können, die dutzende Menschenleben fordern, falls sie eine unkontrollierte Dynamik entwickeln. Um solche Ereignisse zu vermeiden, benötigen wir genauere Kenntnisse über die Gesetze von Personenströmen. Bereits einfachste und kostengünstige Mittel - wie das Errichten von „Einbahnen“, die zu einer Homogenisierung des Flusses führen, oder „Wellenbrecher“ zur Reduktion des Druckes - können zu einer erheblichen Verbesserung der Sicherheit beitragen.

Ziel der vorliegenden Dissertation ist eine Erweiterung der bisherigen Modellierungsansätze von Personenströmen in Gebäuden, um deren Planer bei der Umsetzung solcher Lösungen zu unterstützen. Die entwickelten Methoden dienen dazu, Verkehrsflüsse zu simulieren und so etwaige Engpässe und Problemzonen in Gebäuden vorab zu lokalisieren und gegebenenfalls zu entschärfen. Es werden dabei spezielle Ansätze entwickelt, um unterschiedliche Personengruppen zu unterscheiden, etwa Menschen, die auf den Gebrauch von Gehhilfen angewiesen sind und geeignete Routing Algorithmen benötigen.

In einer „state-of-the-art“ Analyse wurden zu diesem Zweck etablierte Verfahren evaluiert, neue Ansätze entworfen sowie eine Kombination von beiden verwendet, um Schwachstellen der bestehenden Verfahren, wie hoher Speicherbedarf, zu kompensieren

und die Möglichkeiten dieser Verfahren zu erweitern. Die vorgeschlagenen Konzepte, Modelle und Algorithmen wurden in JAVA implementiert und getestet.

Zur Umsetzung wurde in der vorliegenden Arbeit ein agentenbasiertes, raum- und zeitdiskretes System entwickelt. Basierend auf den diskretisierten Gebäudeflächen werden Personen generiert (Agenten), die sich frei auf diesem Raster bewegen können und deren Verhalten dabei dynamisch von vorgegebenen individuellen Zielen, je nach Gruppenzugehörigkeit unterschiedlichen Eigenschaften der Agenten (z.B. führt Rollstuhlgebundenheit zu unterschiedlichen, räumlichen Randbedingungen) und von den umgebenden Agenten abhängt. Dabei wurde speziell auf die Implementierung mehrstöckiger Gebäude Wert gelegt und die entsprechenden Methoden, Algorithmen und Routinen implementiert, um es den Agenten zu ermöglichen, zwischen diesen einzelnen Gebäudeflächen zu wechseln. Dies stellte besondere Herausforderungen an das „Routing“ dar, das vorab einen Weg durch die Gebäude sucht. Um dies umzusetzen, wurde auf die Graphentheorie zurückgegriffen, mit Hilfe derer die Topologie des Gebäudes auf Knoten und Kanten abgebildet wird. Die vorliegende Arbeit zeigt, dass es möglich ist, mit der entwickelten Methode, die in ihren Grundzügen auf dem Suchalgorithmus von Dijkstra basiert, eine Liste zu erstellen, die Wegpunkte enthält, die der jeweilige Agent „abarbeiten“ muss, um an sein Ziel zu gelangen und zwar abhängig von den jeweiligen Eigenschaften des Agenten und äußeren Einflussfaktoren. Zur Orientierung zwischen diesen Wegpunkten wurde auf eine Reihe von vordefinierten Wegen gesetzt, die bei Bedarf automatisch durch ein static floor field ergänzt werden. Durch die Kombination dieser beiden Algorithmen wurden die Vorteile beider Methoden weitgehend vereint.

Die Simulation darf, nach dem aktuellen Forschungsstand, nicht als Evakuierungssimulation verstanden werden, da spezielle Verhaltensweisen, wie menschliches Verhalten in Stresssituationen, nicht berücksichtigt wurden. Sie ist jedoch eine Vorstufe dazu und kann gegebenenfalls durch Hinzufügen einiger Algorithmen dazu erweitert werden.

Abstract

The dynamic use of buildings or entire neighborhoods by people is becoming increasingly important. Buildings and cities are no longer static, instead they are perceived and used as a dynamic system. For this reason, simulation methods that simulate the interaction of a large number of individuals and the built structures are becoming more and more important.

Therefore, better methods of pedestrian flow simulations and pedestrian flow analysis are needed, both for daily usage as well as in evacuation simulations of buildings. Tragic events in the past have shown how quickly large crowds can become to deadly traps that take dozens of lives as toll, if they develop uncontrolled dynamics. To avoid such accidents in the future, it is necessary to gain a profound knowledge of the laws of pedestrian flows. Even simple and cost-effective resources inferred from such knowledge - such as the installation of "one way streets" that lead to a homogenization of the flow, or "breakwater" to reduce the pressure - can contribute to a significant improvement in security.

The aim of this dissertation is to extend the previous modelling approaches of pedestrian flows in buildings to assist the planner in providing better solutions. The developed methods are used to simulate pedestrian flows and thus to locate possible bottlenecks and problematic areas in buildings in advance and to resolve them if necessary. Therefore special approaches were developed to distinguish different groups of people who, for instance, are dependent on the use of walking aids and require appropriate routing algorithms.

For this purpose established "state-of-the-art" methods have been evaluated. A com-

bination of these and new approaches has been used to overcome the weaknesses of existing methods, such as high memory requirements, and to expand their capabilities. The proposed concepts, models and algorithms have been implemented and tested in JAVA.

The implementation presented in this work is an agent-based, space- and time- discrete system. Based on the discretized floor planes people are generated (agents), which can move freely on this grid. Their behavior is predetermined by individual destinations and different attributes (e.g. wheelchair user) which vary according to group membership and depend on the surrounding agents. Special emphasis was put on the implementation of multistorey buildings, and different methods, algorithms and routines were implemented to allow the agent to switch between these floors. Therefore special routing algorithms have been developed. The graph theory was used to map the topology onto the building on nodes and edges. This work shows that it is possible to create, with the developed method which is based mainly on the search algorithm of Dijkstra, a list of waypoints, which are used by the agents to reach their destinations. For orientation between waypoints predefined paths are used which are automatically supplemented by a static floor field when needed. The combination of these two algorithms joins the advantages of both methods.

The simulation may, in its current state, not to be understood as an evacuation simulation, because specific patterns of behaviour, such as human behaviour in stressful situations, was not considered. However, it is a precursor and may potentially be extended to be so by adding necessary algorithms.

Danksagung

Ein besonderes Wort des Dankes möchte ich an meine Familie und im Speziellen an meine Eltern richten, die die Vollendung dieser Arbeit leider nicht mehr miterleben durften, mir aber stets mit Rat und Tat zur Seite gestanden sind.

Des Weiteren danke ich meinen beiden Betreuern Professor Felix Breitenecker und Professor Georg Franck-Oberaspach, für die hervorragende Betreuung und ständige Diskussions- und Hilfsbereitschaft.

Besonderer Dank gilt auch der gesamten Arbeitsgruppe für die freundschaftliche Arbeitsatmosphäre, die maßgeblich zum Gelingen dieser Arbeit beigetragen hat, hierbei insbesondere Niki Popper, Michael Landsiedl und Gabriel Wurzer.

Last but not least gebührt großer Dank an Xenia Descovich, die in mühevollen Stunden die Arbeit Korrektur gelesen hat.

Inhaltsverzeichnis

1	Einführung	2
1.1	Motivation und Überblick	2
1.2	Schritte zu einer verlässlichen und realistischen Simulation	4
1.3	Zusammenfassung der Arbeit	4
2	State of the Art	6
2.1	Fußgängerfluss Simulation	6
2.2	Makroskopische Simulation	8
2.3	Mikroskopische Simulation	9
2.4	Zelluläre Automaten	10
2.4.1	Theorie	10
2.4.2	Anwendung als Pedestrian Dynamics Modell	12
2.4.3	Floor Fields	13
2.4.4	Ablauf	15
2.5	Agentenbasierte Modellierung	18
2.6	Social Force Model	19
2.7	Magnetic Force Model	21
2.8	Queuing Network Model	24
3	Das Modell	25
3.1	Einführung	25
3.2	Die Umgebung	26
3.3	Der Agent	28
3.4	Gehgeschwindigkeit	30
3.4.1	Abhängigkeiten von Alter und Geschlecht	30
3.4.2	Geschwindigkeit auf Stiegen	32

3.4.3	Abhängigkeit von Geschwindigkeit und Dichte	33
3.4.4	Geschwindigkeit von Rollstuhlfahrern	35
3.4.5	Bestimmung der momentanen Geschwindigkeit	35
3.5	Routing	36
3.5.1	Globale Wegsuche	36
3.5.1.1	Graphentheorie	37
3.5.1.2	Modell Graph	38
3.5.1.3	Gewichtungsfunktion	39
3.5.1.4	Dijkstra Algorithmus	41
3.5.1.5	Erstellen der Zwischenziel List	43
3.5.2	Lokale Wegsuche	43
3.5.2.1	Static Floor Field	43
3.5.2.2	Wegpunkte	50
3.5.2.3	Kombinierte Suche	53
3.6	Aufzüge	53
3.6.1	Aufzugssteuerung	54
3.6.2	Modellierung der Steuerung	55
3.6.2.1	Mathematisches Modell	55
3.6.3	Wartezeiten vor dem Aufzug	58
3.6.3.1	Abschätzung der Transportzeit via Aufzug	58
4	Implementierung	67
4.1	Einführung	67
4.2	Simulationsprojekt	67
4.2.1	Project.csv	68
4.2.2	*.png	72
4.2.3	*.txt	74
4.3	Umgebung	75
4.3.1	Initialisierung	76
4.3.2	Elemente	77
4.4	Agent	81
4.4.1	Initialisierung	81
4.5	Routing	83
4.5.1	Initialisierung Modell Graph	83

4.6	Aufzüge	85
4.6.1	Graphische Darstellung	85
4.6.2	Funktionsweise der implementierten Aufzugssteuerung	87
4.6.3	Programmierparadigma	88
4.7	Diskretisierungsfehler	89
4.7.1	Fehler durch Zeitintervall Δt	89
4.7.1.1	Fehlerberechnung anhand eines Beispiels	90
4.7.1.2	Kompensation des Δt Fehlers	91
4.7.2	Fehler durch Rasterung des Raumes	91
4.7.2.1	Fehlerberechnung	93
4.7.2.2	Kompensation des Fehlers durch Rasterung des Raumes	95
5	Simulationsablauf	97
5.1	Update-Prozess	97
5.1.1	Aufzüge	98
5.1.2	Einfügen neuer Agenten	98
5.1.3	Erstellen der Trajektorien	99
5.1.3.1	Einfügen Wartender Agenten	99
5.1.3.2	Neuer Routing-Vorgang	99
5.1.3.3	Bestimmung der besten Richtung	100
5.1.3.4	Umgebung Scannen	100
5.1.3.5	Interaktion mit Agenten	101
5.1.3.6	Bewegen der Agenten	105
5.1.3.7	Visualisierung	106
5.2	Szenario	106
5.2.1	Simulationsumgebung	107
5.2.2	Durchführung	107
5.2.3	Ergebnis	108
6	Zusammenfassung und Erkenntnisse	110
6.1	Zusammenfassung	110
6.2	Erkenntnisse	111
	Zusammenfassung	113

Inhaltsverzeichnis	1
Abbildungsverzeichnis	113
Tabellenverzeichnis	115
Literaturverzeichnis	116
Curriculum Vitae	120

Kapitel 1

Einführung

1.1 Motivation und Überblick

Fußgängerflusssimulationen erlangten in den letzten Jahrzehnten immer größere Bedeutung, sei es in der Gebäudeplanung in Architekturbüros oder bei der Planung und Gestaltung ganzer Stadtteile. Den größten Nutzen hat die Personenstromanalyse in Evakuierungssimulationen von Gebäuden oder Fahrzeugen (Zügen, Kreuzfahrtschiffen, Flugzeugen, ...).[18, S. 18f] Tragische Ereignisse, wie die *Loveparade 2010* in Duisburg, zeigten auf, wie rasch sich große Menschenansammlungen zu tödlichen Fallen entwickeln können, die dutzende Menschenleben fordern, falls sie eine unkontrollierte Dynamik entwickeln.

Um solche Katastrophen in Zukunft zu vermeiden ist es von Nöten, profunde Kenntnisse über die Gesetze von Personenströmen zu erlangen. Denn bereits einfachste und kostengünstigste Mittel können zu einer erheblichen Verbesserung der Sicherheit beitragen. Das Errichten von „Einbahnen“ führt zu einer Homogenisierung des Flusses, „Wellenbrecher“ zu einer Reduktion des Druckes, der sich durch Menschenmengen aufbauen kann. Diese Vorrichtungen haben sich vor allem bei Sportereignissen, Konzerten und anderen Großveranstaltungen bewährt, bei denen Menschenansammlungen auf einer weitläufigen Fläche kontrolliert werden müssen. Anders hingegen verhält sich die Situation in engen Gebäuden und Fahrzeugen, bei denen vermehrt Aufmerksamkeit

auf die richtige Dimensionierung der Fluchtwege und der Beschilderung der Fluchtwege gelegt werden muss, vor allem wenn man die rasante Entwicklung im Bereich der Kapazitäten betrachtet. So sind etwa Kreuzfahrtschiffe mit 8000 Personen und mehr bereits Realität. Zum Vergleich, die RMS Titanic hatte Platz für etwa 2200 Passagiere und Besatzungsmitglieder. Im Bereich Luftfahrt, wo der neue Airbus A380 mit einer zugelassenen Kapazität von 853 Passagieren eine noch nie dagewesene Kapazität bietet und trotzdem in der von der internationalen Luftfahrtbehörde vorgeschriebenen Zeit zu evakuieren sein muss, oder Fußballstadien, die mit bis zu 100.000 Sitzplätzen aufwarten. All dies benötigt ein durchdachtes Gebäude- bzw. Fahrzeug-Layout, das eine ausreichende Dimensionierung der Fluchtwege, Stiegen und Türen vorsieht, um die Gefahren bei einer Evakuierung so gering wie möglich zu halten.

Da die ausreichende Dimensionierung zumeist auf Kosten der nutzbaren und somit gewinnbringenden Fläche entfällt, sei hier auch noch der ökonomische Gesichtspunkt erwähnt, der hier natürlich nicht außer Acht gelassen werden darf. Bei all diesen Aufgaben stellt die Fußgängerflusssimulation ein Werkzeug dar, mit deren Hilfe man die Realität, unter Zuhilfenahme von mathematischen Modellen, so weit wie möglich nachstellt und somit Daten und Vorhersagen bei der Räumung von Fahrzeugen und Gebäuden erlangen kann. Die so erhaltenen Ergebnisse können im Anschluss herangezogen werden, um bauliche Veränderungen des Layout vorzunehmen und so, bereits im Vorfeld, kritische Bauabschnitte wie zum Beispiel Bottlenecks zu entschärfen. Dies führt nicht nur zu einer erheblich Reduktion von Kosten durch eventuell auftretende Nachbesserungen, sondern minimiert auch das Risiko bei etwaigen realen Experimenten, die, wie bereits angesprochen, von verschiedenen Organisationen vorgeschrieben werden.

Ein weiterer Aspekt, in der Fußgängerflusssimulationen einen wichtigen Beitrag leisten können, ist im Bereich des Barrierefreien Bauens. In einer Definition des Bundesministeriums für Arbeit, Soziales und Konsumentenschutz in Österreich steht, *„Barrierefrei sind bauliche und sonstige Anlagen, Verkehrsmittel, technische Gebrauchsgegenstände, Systeme der Informationsverarbeitung sowie andere gestaltete Lebensbereiche, wenn sie für Menschen mit Behinderungen in der allgemein üblichen Weise, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe zugänglich und nutzbar sind“*. (BGBl. I Nr. 82/2005, Art. 2 Z 7, ab 1.1.2006) Die Modellierung und Simulation dieser Barrierefreiheit wird ein nicht unwesentlicher Bestandteil dieser Arbeit sein.

1.2 Schritte zu einer verlässlichen und realistischen Simulation

Es bedarf einiger Schritte, die auf den Weg zum fertigen Simulationsmodell zu beschreiten sind. Nachfolgend ein kurzer Überblick der Punkte, die Tobias Kreutz in seiner Arbeit *Pedestrian Traffic – Simulation and Experiments* [14, S. 1] definiert, um ein valides Modell zu erhalten.

1. Beobachtung von Phänomenen im Verhalten von Fußgängerbewegungen.
2. Identifizieren der entscheidenden Elemente, die maßgeblichen Einfluss auf das Verhalten haben.
3. Erstellen eines Modells, das in der Lage ist, die Phänomene im Verhalten zu reproduzieren.
4. Eine Implementierung des Modells in einer geeigneten Programmiersprache.
5. Experimentieren und Beobachten, um die Modellparameter zu kalibrieren.
6. Durchführen von standardisierten Tests.
7. Überprüfen, ob die Rechtlichen Regularien erfüllt werden.
8. Zertifizieren des Simulators.

Hierbei sei jedoch erwähnt, dass der Prozess des Beobachtens, Modellierens und Testens keine Einbahnstraße ist, sondern ein sich ständig wiederholender Prozess, bei dem erlangte Erkenntnisse und Fehlschläge zu einer Überarbeitung und somit zu einer stetigen Verbesserung des Modells führen.

1.3 Zusammenfassung der Arbeit

Die Arbeit beginnt mit der Motivation und dem Überblick in Kapitel 1, gefolgt von der Zusammenfassung der Arbeit.

Kapitel 2 gibt einen kurzen Überblick über die momentan gängigen Methoden und Verfahren, die in der Modellierung von Fussgängerbewegungen üblich sind. Hierbei wird im Speziellen auf die Methodik der zellulären Automaten und agentenbasierten Systeme Bezug genommen, da sie im weiteren Verlauf eine wichtige Rolle spielen.

Das Modell, das im Zuge der Arbeit entstanden ist, wird im Detail in Kapitel 3 behandelt. Es soll hierbei ein genauer Überblick über folgende Punkte vermittelt werden.

- Modellierung der Agenten und ihrer Umgebung, in der sie interagieren.
- Individuelle Geschwindigkeiten.
- Routing und Wegsuche.
- Aufzüge.

In diesem Abschnitt der Dissertation wird jedoch noch nicht auf die Implementierung in JAVA eingegangen, die zu Testzwecken erstellt wurde, sondern es wird lediglich der mathematische Teil des Modells beleuchtet.

Anders als im vorhergehenden Abschnitt wird dann in Kapitel 4 auf die Implementierung des Modells eingegangen. Hierbei wird auf die Umsetzung der einzelnen Objekte eingegangen sowie auf deren Parametrisierung. Ein weiterer Schwerpunkt ist das Aufzeigen von Fehlern, die durch das angewandte Modellierungsparadigma unweigerlich entstehen, und wie sie umgangen beziehungsweise minimiert werden.

Der letzte Teil, Kapitel 5, beschäftigt sich Schritt für Schritt mit dem Ablauf der Simulation. Es soll hierbei vermittelt werden, wie und warum sich Agenten in der programmtechnischen Umsetzung verhalten.

Den Abschluss bilden eine kurze Zusammenfassung der Arbeit sowie die Schlussfolgerung, die daraus gezogen werden kann.

Kapitel 2

State of the Art

2.1 Fußgängerfluss Simulation

Dieser Abschnitt umfasst einen kurzen Überblick über die momentan gängigsten Methoden und Praktiken in der modernen Fußgängerflusssimulation, wobei hier nicht der Eindruck erweckt werden soll, dass dieses Gebiet in diesem Kapitel vollständig abgedeckt wird. Zu umfangreich und vielfältig sind die Techniken, die in den letzten Jahren aus Forschung und Entwicklung hervorgegangen sind. Die ersten empirischen Studien über dieses Thema wurden bereits vor mehr als 50 Jahren von Hankin und Wirght (1958) und Fruin (1971) durchgeführt und befassten sich hauptsächlich mit den Eigenschaften von Fußgängereinrichtungen. Mit zunehmender Verbreitung der Computertechnik und der Steigerung der Leistungsfähigkeit von Rechnern stieg das Verlangen, mathematische Modelle zu entwickeln, die Evakuierungs- und Panikverhalten möglichst treffend beschreiben, und sie in computerunterstützten Simulationsdurchläufen zu analysieren. Über die Zeit entwickelten sich somit unzählige Modellierungsmethoden und Ansätze, die von *Neil Gershenfeld* in seiner Arbeit *The Nature of Mathematical Modeling* [6, S. 2] zusammengefasst wurde und zur Unterscheidung und Klassifizierung dieser Ansätze dienen.

spezifisch	\longleftrightarrow	allgemein
phänomenologisch	\longleftrightarrow	erste Prinzipien
diskret	\longleftrightarrow	kontinuierlich
numerisch	\longleftrightarrow	analytisch
stochastisch	\longleftrightarrow	deterministisch
quantitativ	\longleftrightarrow	qualitativ
makroskopisch	\longleftrightarrow	mikroskopisch

Tabelle 2.1: *Klassifizierung nach Gershenfeld [6]*

Da in dieser Gegenüberstellung verschiedener Methodiken nicht alle für unser Problem relevant sind, werden im Folgenden nur einige ausgewählte genauer betrachtet.

diskret Bei einem diskreten System können die relevanten Größen nur diskrete Werte annehmen. Einige Beispiele für diese Größen sind Zeit, Ort oder Zustände, die sich durch Eintreten eines Ereignisses ändern.

kontinuierliche Bei einem kontinuierlichen System können die relevanten Größen typischerweise nur kontinuierliche Werte annehmen. Das System wird, für gewöhnlich, mittels Differentialgleichungen modelliert.

numerisch Unter numerischer Simulation versteht man im Allgemeinen Simulationen, welche mittels numerischer Methoden durchgeführt werden.

analytisch Im Gegensatz zur numerischen Simulation werden hierbei analytische Methoden zur Beschreibung des Modelles eingesetzt.

deterministisch In einem deterministischen Modell sind die betrachteten Variablen und Werte eindeutig bestimmt und unterliegen keinem Zufallsprozess. Jeder Systeminput führt zu einem vorhersagbaren Output.

stochastisch Im Gegensatz zu einem deterministischen Modell existieren hier stochastische, also zufallsbehaftete, Prozesse, die trotz gleicher Inputs unterschiedliche Outputs generieren können.

makroskopisch Makroskopische Modelle haben den Fokus nicht auf das einzelne Objekt gerichtet, sondern auf eine Aggregation gewisser Eigenschaften. So ist etwa nicht die Position oder Richtung einer einzelnen Person von Interesse, sondern die Personen-

dichte eines Gebietes und der Fluss darin.

mikroskopisch Im Gegensatz zu makroskopischen Modellen ist hier das einzelne Objekt von Interesse und deren Attribute.

Im Anschluss wird auf die Unterschiede zwischen Mikro- und Makrosimulation näher eingegangen wobei das Hauptaugenmerk auf die Mikrosimulation gerichtet sein wird, da das in den folgenden Kapiteln erörterte Modell dieser Methodik folgt.

2.2 Makroskopische Simulation

Traditionelle makroskopische Modelle vernachlässigen bei Personenströmen das einzelne Individuum. Grundlage der Modellannahme hierbei ist, dass die Größen

- Dichte $\rho(x, t)$
- Geschwindigkeit $v(x, t)$
- Fluss $f(x, t)$

ausreichend sind, um das System zu beschreiben und Aussagen über das Verhalten der Menge zu ermöglichen. Des Weiteren ist aber zu erwähnen, dass es hierbei zu Schwierigkeiten kommen kann, wenn die Dichte des Personenstromes zu gering ist. Zur Modellierung kommen vor allem Differentialgleichungen zum Einsatz, wie sie auch in der Simulation von Flüssigkeiten und Gasen Verwendung finden, da Studien zeigen, dass sich Personenströme ähnlich verhalten. Hierbei finden häufig die aus der Gasdynamik bekannte Boltzmann-Gleichung und die in der Strömungslehre vertretenen Navier-Stokes-Gleichungen [8] ihren Einsatz.

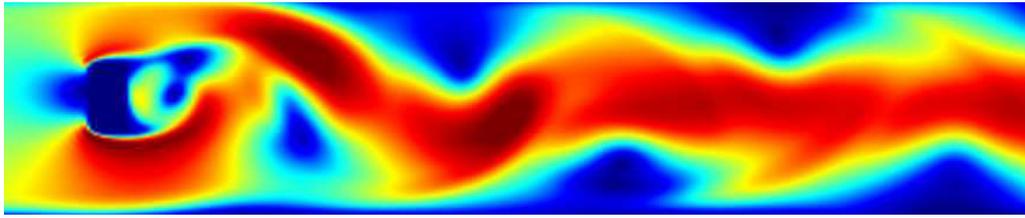


Abbildung 2.1: *Makroskopische Simulation mittels Lattice-Boltzmann Methode [15]*

Abbildung: 2.1 veranschaulicht das Ergebnis der Simulation einer makroskopischen Flüssigkeit mittels Lattice-Boltzmann Methode.

Da es sich bei dem in der Dissertation vorgestellten Modell um keine makroskopische Simulation handelt, werden hier keine State of the Art Methoden zu dieser Technik vorgestellt.

2.3 Mikroskopische Simulation

Im Gegensatz zur makroskopischen Simulation wird bei der mikroskopischen Simulation das einzelne Individuum betrachtet. Das bedeutet, dass die Wechselwirkungen die zwischen Personen stattfinden, für jede Einheit individuell berechnet werden. Des Weiteren bietet diese Methode die Möglichkeit, jedem einzelnen Individuum ihre ganz spezifischen körperlichen Eigenschaften, wie Gewicht, Alter, Geschlecht, zuzuordnen, die sich maßgeblich auf die persönliche Geschwindigkeit und im Fall einer Evakuierung auf ihre Durchsetzungskraft auswirken. Die *International Microsimulation Association*¹ definiert Mikrosimulation wie folgt:

Microsimulation (originally called microanalytic simulation) is a modelling technique that operates at the level of individual units such as persons, households, vehicles or firms. Within the model each unit is represented by a record containing a unique identifier and a set of associated attributes – e.g. a list of persons with known age, sex, marital and employment status; or a list of vehicles with known origins, destinations

¹<http://www.microsimulation.org/>

and operational characteristics. A set of rules (transition probabilities) are then applied to these units leading to simulated changes in state and behaviour. These rules may be deterministic (probability = 1), such as changes in tax liability resulting from changes in tax regulations, or stochastic (probability ≤ 1), such as chance of dying, marrying, giving birth or moving within a given time period. In either case the result is an estimate of the outcomes of applying these rules, possibly over many time steps, including both total overall aggregate change and, crucially, the distributional nature of any change. Given the emphasis on changes in distribution, microsimulation models are often used to investigate the impacts on social equity of fiscal and demographic changes (and their interactions). Modelling of the distribution of traffic flows over a street network is another increasingly important use of the approach. [29]

Im Folgenden werden nun einige der geläufigsten Mikrosimulationen und deren Techniken kurz erläutert. Der Grad, wie sehr bei den verschiedenen Methoden ins Detail gegangen wird, hängt von der Relevanz für die weitere Arbeit ab.

2.4 Zelluläre Automaten

2.4.1 Theorie

Zelluläre Automaten wurden gegen 1940 vom polnisch-US-amerikanischen Mathematiker Stanislaw Ulam vorgestellt und von seinem damaligen Kollegen John von Neumann erweitert. Es handelt sich dabei um ein räumlich diskretisiertes, dynamisches Modell, dessen Zustand zum Zeitpunkt t_{k+1} ausschließlich vom Zustand des Systems zum Zeitpunkt t_k abhängig ist. Jede Zelle ändert ihren Zellzustand in Abhängigkeit ihres Zustands und jenem ihrer definierten Nachbarschaft.

Man kann zelluläre Automaten durch einige ihrer Eigenschaften kennzeichnen.

1. *Zellanordnung und deren Geometrien R*

Je nach Anwendungsbereich und Dimension des Automaten kann die Anordnung

der einzelnen Zellen variieren. Für Anwendungen im Eindimensionalen ist die Anordnung in einer Reihe möglich, einige Muster für die Zweidimensionale Anordnung mit Quadraten, Drei- und Sechsecken sind in Abbildung 2.2 illustriert.



Abbildung 2.2: *Beispiel Zell Geometrien*

2. Nachbarschaft der Zellen N

Die Nachbarzellen sind, wie bereits erwähnt, maßgeblich am Zustand der Zelle mitverantwortlich und werden, in Abhängigkeit der Geometrie, genau festgelegt. Bekannte Beispiele für die quadratische Anordnung der Zellen sind die Von-Neumann- und die Moore-Nachbarschaft.

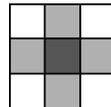


Abbildung 2.3: *Von-Neumann-Nachbarschaft*

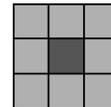


Abbildung 2.4: *Moore-Nachbarschaft*

3. Zustandsmenge Q

Die Zustandsmenge definiert alle möglichen Zustände, die eine Zelle annehmen kann.

4. Überföhrungsfunktion δ

Die Überföhrungsfunktion definiert in Abhängigkeit der Zustände der Nachbarzellen und der Zelle selbst im Zeitpunkt t_k den neuen Zustand zum Zeitpunkt t_{k+1} . $\delta : Q^N \rightarrow Q$

2.4.2 Anwendung als Pedestrian Dynamics Modell

Die zugrundeliegende Struktur für diverse Pedestrian Dynamics Modelle, die auf zellulären Automaten aufbauen, ist ein zweidimensionales Gitter, das, je nach Anwendung, periodische Randbedingungen aufweisen kann. Eine zweidimensionale Zellanordnung mit periodischen Randbedingungen bedeutet, dass die jeweiligen gegenüberliegenden Ränder des Gitters verbunden sind. Solle Flächen stellen sich in der Geometrie als Torus, der in der Form mit dem eines Rettungsringses verglichen werden kann, dar. Die Größe einer Zelle ist dem Platzbedarf einer durchschnittlichen Person angepasst und beträgt etwa 40×40 cm [28], womit eine Entität genau eine Zelle besetzt, die sie bei jedem diskreten Zeitschritt $t \rightarrow t + 1$ in Richtung einer unbesetzten verlassen kann oder in ihrer momentanen Position verharrt. Die Wahl des nächsten Schrittes wird zu einem späteren Zeitpunkt noch genauer erörtert. Die Zustandsmenge des Modells besteht hierbei lediglich aus den beiden Elementen *frei* beziehungsweise *besetzt*. Die Definition eines Zustandes, der ein Hindernis wie eine Wand oder Säule darstellt, kann meist eingespart werden, da in diesem Fall der Einfachheit halber die Zelle entfernt wird und so ebenfalls Unverfügbarkeit signalisiert wird.

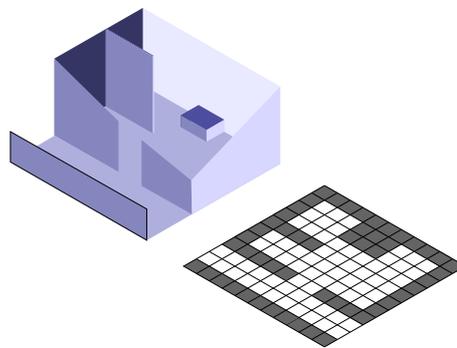


Abbildung 2.5: *Transformation eines Gebäudes in die Ebene*

Abbildung 2.5 illustriert die Transformation eines dreidimensionalen Gebäudes in eine räumlich diskretisierte Ebene. Schwarz markierte Flächen stellen Hindernisse wie Wände oder Ähnliches dar, während weiße Flächen frei begehbar sind und von Personen besetzt werden können.

2.4.3 Floor Fields

Um Personen in die Lage zu versetzen, auch über längere Distanzen miteinander zu interagieren, und des Weiteren anziehende Kräfte, die zu Türen oder ähnlichem, für den einzelnen, Interessanten Gebiete gehören, zu modellieren, werden zwei zusätzliche Felder eingeführt, die sogenannten Static und Dynamic Floor Fields [3, 17, 13]. Diese beiden Hilfsmittel, im Folgenden *SFF* und *DFF* genannt, können ebenfalls als Gitter angesehen werden, wobei das ursprünglich erzeugte als Vorlage angenommen wird. Das bedeutet, dass die Anzahl und Position der Zellen ident sind.

Static Floor Field (SFF): Das SFF ist, wie die Namensgebung bereits andeutet, ein fix bei der Initialisierung des Modells generiertes Feld, das für jede Zelle des Gitters einen zur Distanz zur Tür oder anderen Zieles proportionalen Wert enthält. Es wird also für jedes mögliche Ziel, das eine Person haben kann, ein separates SFF benötigt. Der Wert, der gespeichert wird, kann die Anzahl der Zellen bis zum Ziel oder die Entfernung in Metern sein und je nach Definition der Nachbarschaft variieren.

4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3
3	2	2	2	2	2	2	3
3	2	1	1	1	1	2	3
			0	0			

Abbildung 2.6: *Beispiel Static Floor Field*

Abbildung 2.6 illustriert ein einfaches Beispiel, wie ein SFF aussehen könnte. Die beiden gelben Zellen, die den Wert 0 enthalten stellen die Initialisierungspunkte dar, bei denen die Initialisierung beginnt. Jeder direkt angrenzende Wert wird auf 1 gesetzt und zeigt damit an, dass er einen Schritt von den Startpunkten entfernt liegt, wobei hier von einer Moore-Nachbarschaft ausgegangen wird. Die weiteren Werte erklären sich von selbst. Die Erstellung eines aufwendigeren und effizienteren SFF sowie der Initialisierungsalgorithmus werden in Abschnitt 3.5.2.1 noch genauer ausgeführt. Die

wertabhängige Einfärbung der Zellen kann als diskretes Potentialfeld angesehen werden, das zur besseren Veranschaulichung der unterschiedlichen Distanzen dient.

Dynamic Floor Field (DFF): Im Gegensatz zum SFF, das seine Werte beim Initialisieren erhält und nicht mehr verändert, wird auf das DFF permanent schreibend und lesend zugegriffen. Zweck des DFF ist es, nicht nur die momentane Anwesenheit einer Person in einer bestimmten Zelle zu signalisieren, sondern auch deren kürzliche Präsenz, die nur langsam abklingt und diffundiert. Das bedeutet, dass jede Identität in Bewegung eine kurze Spur im DFF nach sich zieht, die andere in ihren Wegfindungsprozess miteinfließen lassen könne. Diese Spur kann etwa anziehend wirken, um Rudelbildung zu modellieren, oder auch abstoßend, um etwa zu dichte Annäherung zweier Agenten zu unterbinden. Zu diesem Zweck existieren zwei Parameter α und β , die die Diffusion und das Abklingen der Zellwerte kontrollieren und deren Dynamik nun kurz erörtert wird. Bei der Initialisierung des DFF werden die Startwerte der einzelnen Zellen auf 0 gesetzt. $D_{ij}^t = 0, t = 0$ gibt diesen Startzustand an. Die Indizes i, j geben die jeweilige Position der Zelle an, $t = 0$ bezieht sich auf den Zustand zum Zeitpunkt 0. Für den Update-Prozess von $t \rightarrow t + 1$ sind im Wesentlichen zwei Punkte durchzuführen.

$$D_{ij}^{t+1} = D_{ij}^t - \alpha D_{ij}^t + \frac{\alpha}{4}(D_{i+1,j}^t + D_{i-1,j}^t + D_{ij+1}^t + D_{ij-1}^t) \quad (2.1)$$

Gleichung (2.1) beschreibt den Diffusionsprozess der einerseits den Zellwert kontinuierlich vermindert, ihn aber, durch Diffusion der Nachbarzellen, auch nach oben korrigieren kann. Zu erwähnen ist hierbei, dass der additive zweite Term der Gleichung nachbarschaftsabhängig ist und in diesem Beispiel von der Von-Neumann-Nachbarschaft ausgegangen wird.

$$D_{ij}^{t+1} = D_{ij}^t - \delta D_{ij}^t \quad (2.2)$$

Der in Gleichung (2.2) verwendete Parameter δ ist für die stetige Abnahme des Wertes

verantwortlich. Die Reihenfolge, in der die beiden Gleichungen (2.1) und (2.2) durchgeführt werden, ist für das Ergebnis irrelevant.

$$D_{ij}^{t+1} = (1 - \alpha)(1 - \delta)D_{ij}^t + \frac{\alpha(1 - \delta)}{4}(D_{i+1,j}^t + D_{i-1,j}^t + D_{ij+1}^t + D_{ij-1}^t) \quad (2.3)$$

$$D_{ij} = D_{ij} + 1 \quad (2.4)$$

Die Kombination der beiden Gleichungen (2.1) und (2.2) ist durch (2.3) gegeben, mit deren Hilfe der erste Update-Schritt des DFF durchgeführt werden kann. Das zweite, recht simple Update des DFF wird durch die Gleichung (2.4) dargestellt und modelliert den Beginn der Spur, den jede Entität im DFF hinterlässt. Dieser Update-Schritt wird lediglich beim Zellwechsel einer Person ausgeführt.

2.4.4 Ablauf

Übergangsmatrix

Da dieses Modell auf der Von-Neumann-Nachbarschaft aufbaut, hat jede Entität die Wahl aus vier Richtungen, die ihr für den nächsten Schritt zur Verfügung stehen, oder sie entscheidet sich für das Verharren in ihrer momentanen Position. Für die Entscheidungsfindung wird eine 3×3 Übergangsmatrix (Abbildung: 2.7) generiert, die die Wahrscheinlichkeiten für die jeweiligen Richtungen enthält, wobei das Element im Zentrum der Matrix den Verbleib an der entsprechenden Position angibt.

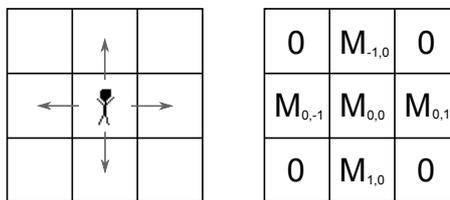


Abbildung 2.7: Wahrscheinlichkeitsmatrix

Die Werte der Übergangsmatrix werden anhand der Gleichung (2.5) ermittelt, wobei hier zu beachten ist, dass nur Zellen infrage kommen, die nicht besetzt sind und somit für einen Wechsel zur Verfügung stehen. Die Werte, die aus den beiden Floor Fields entnommen werden, werden mit D (dynamic) und S (static) bezeichnet und mit den Parametern k_S beziehungsweise k_D gewichtet.

$$p_{ij} = N \cdot \exp(k_D \cdot D_{ij}) \cdot \exp(k_S \cdot S_{ij}) \cdot p_I(i, j) \cdot p_w \quad (2.5)$$

Zusätzlich zu den bereits bekannten Konstanten und Variablen sind in obenstehender Gleichung noch der Normalisierungsfaktor N , der Parameter $p_I(i, f)$, der die Trägheit der jeweiligen Person repräsentiert, sowie p_w , der für einen abstoßenden Effekt von Wänden sorgt, zu erkennen. Der Trägheitsfaktor $p_I(i, f) = \exp(k_i)$ für die Richtung, die im vorhergegangenen Schritt eingeschlagen wurde, und $p_I(i, f) = 1$ für die restlichen Richtungen. k_i bezeichnet hierbei einen Gewichtungsparemeter. Die erwähnte abstoßende Wirkung von Wänden ist durch $p_w = \exp(k_w \cdot \min(D_{max}, d))$ gegeben, wobei auch k_w ein Gewichtungsparemeter ist. d gibt die minimale Distanz zu allen umgebenden Wänden an, die jedoch durch die Konstante D_{max} nach oben hin beschränkt wird.

Kollisionsvermeidung

Dieser Ansatz wird für alle Entitäten parallel und unabhängig durchgeführt, was spätestens bei der eigentlichen Bewegung zu Kollisionen und gegenseitigen Behinderungen führen kann.

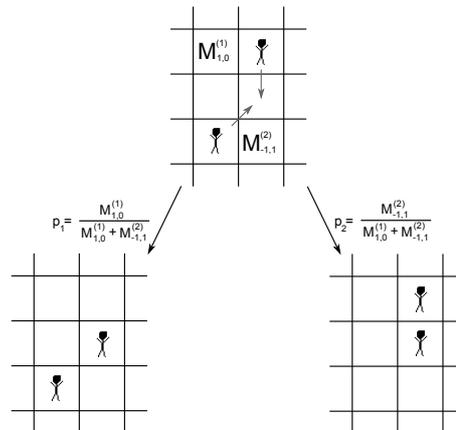


Abbildung 2.8: *Kollisionsvermeidung*

Im Fall, dass zwei Personen dieselbe Zelle besetzen wollen, wird eine davon gemäß der relativen Wahrscheinlichkeit der jeweilig gewählten Richtung ausgewählt. Abbildung 2.8 illustriert ein entsprechendes Szenario mit den resultierenden Wahrscheinlichkeiten.

Update Schritte

1. Update des Dynamic Floor Fields laut Gleichung (2.1) und (2.2)
2. Erstellen der Übergangsmatrizen für jede Entität laut Gleichung (2.5)
3. Jede Entität wählt mithilfe eines Zufallsprozesses und der Übergangsmatrix die Richtung des nächsten Schrittes.
4. Kollisionshandling laut Abbildung 2.8
5. Platzieren der Entitäten an ihren neuen Platz und Aktualisieren des DFF mittels Gleichung (2.4)

Die bis jetzt erörterten Methoden und Algorithmen werden in der obenstehenden Reihenfolge durchgeführt und ergeben so die Regeln, die auf alle Entitäten parallel in dem

zellulären Automaten angewandt werden.

2.5 Agentenbasierte Modellierung

Der Modellierungsansatz bei der agentenbasierten Simulation setzt auf kleine Einheiten, die Agenten, und deren Handlungsmöglichkeiten.[1, 24, 7, 3] Ihren Ursprung hat diese Modellierungsmethodik in diversen Bereichen der künstlichen Intelligenz und den zellulärer Automaten und kann in mancherlei Hinsicht als eine Erweiterung dieser interpretiert werden. Über die Frage, was ein Agent genau ist, herrscht in der Fachwelt nicht immer Einigkeit, es wird jedoch im Anschluss eine Definition bereitgestellt, die sich an dem *Tutorial on agent based modeling and simulation Part 2: How to model with agents* [4] von C. Macal und M. North orientiert.

- Ein Agent ist ein diskretes Individuum mit einer Menge von Eigenschaften, das selbständig Entscheidungen mithilfe von Regeln treffen kann.
- Er befindet sich in einer Umgebung, mit der er und in der er interagiert.
- Als Individuum kann er versuchen, Ziele zu erreichen und seine Entscheidungen abzuwägen.
- Der Agent ist autonom und agiert unabhängig in seiner Umgebung und im Umgang mit anderen Agenten.
- Er ist flexibel, hat die Fähigkeit, zu lernen und sich anzupassen.

Auffallend bei dieser theoretischen Definition eines Agenten ist, dass hier wenig über dessen Implementierung ausgesagt wird. Es wird in keiner Weise erwähnt, ob es sich um eine diskrete oder kontinuierliche Simulation handelt oder ob es eine räumliche Verteilung geben muss. Darum ist es auch nicht verwunderlich, dass die Anwendungsgebiete sehr vielfältig sind und sich von Modellen über Wirtschaftssystemen, Segregation, sozialen Netzwerken bis hin zum Verhalten von Fußgängern erstreckt.

Eine recht häufige Implementierung im Bereich der Pedestrian Dynamics ist die zeit und ort-diskretisierte Modellierung, die sich sehr stark an den zellulären Automaten anlehnt. Die Agenten befinden sich hierbei ebenfalls auf diskreten Gittern, wie sie in Abschnitt 2.4.1 erörtert worden sind, nur findet die Interaktion zur Wegfindung nicht auf einer definierten Nachbarschaft statt, sondern erfolgt durch Kommunikation und Austauschen von Nachrichten zwischen den Agenten selbst. In diesem speziellen Fall kann die agentenbasierte Simulation als eine Erweiterung der zellulären Automaten angesehen werden und wird, da dieses Konzept für die Simulation essentiell ist, später noch genauer besprochen.

Die beiden nachfolgenden Methoden 2.6 und 2.7 unterscheiden sich zwar in ihren Methoden zur Interaktion der Agenten voneinander, erfüllen aber die Definition eines agentenbasierten Systems, wie sie hier vorgestellt wurde.

2.6 Social Force Model

In diesem Abschnitt soll ein kurzer Einblick in das *Social Force Model* vermittelt werden, das maßgeblich von Helbing und Molnar [11, 26] entwickelt wurde. Grundgedanke ist hierbei, dass jeder Agenten i auf kürzeste Weise sein Ziel erreichen möchte und dabei von unterschiedlichsten Kräften abgelenkt wird. Die Richtung, die er zum Erreichen des Ziels einschlagen müsste, gibt dabei ein normierter Vektor, von seiner momentanen Position zu seinem Ziel, an und ist durch Gleichung (2.6) gegeben.

$$\mathbf{e}_i = \frac{\mathbf{x}_i^0 - \mathbf{x}_i(t)}{\|\mathbf{x}_i^0 - \mathbf{x}_i(t)\|} \quad (2.6)$$

- \mathbf{x}_i^0 ... Position des Ziels
- $\mathbf{x}_i(t)$... Position des Agenten zur Zeit t
- \mathbf{e}_i ... die angestrebte Richtung

Mit Gleichung (2.7) wird vorab die endgültig resultierende Kraft für jeden Agenten i in dem Modell bestimmt. Die einzelnen Summanden werden im Anschluss daran noch genauer erläutert.

$$\mathbf{F}_i(t) = m \frac{d\mathbf{v}_i(t)}{dt} = m \underbrace{\frac{v_0 \mathbf{e}_i - \mathbf{v}_i(t) + \xi_i(t)}{\tau}}_* + \underbrace{\sum_{i \neq j} \mathbf{f}_{ij}(\mathbf{x}_i(t), \mathbf{x}_j(t))}_{**} + \underbrace{\mathbf{f}_b(\mathbf{x}_i(t))}_{***} \quad (2.7)$$

$\mathbf{v}_i(t)$... Geschwindigkeit des Agenten zur Zeit t

m ... Masse des Agenten

v_0 ... die angestrebte Geschwindigkeit

$\xi_i(t)$... die Fluktuation der individuellen Geschwindigkeit

\mathbf{f}_{ij} ... die abstoßende Kraft zwischen Agent i und j

\mathbf{f}_b ... die Wechselwirkung mit den Grenzlinien

τ ... Zeitschritt

Bei näherer Betrachtung wird ersichtlich, dass sich die Gleichung (2.7) für die resultierende Kraft aus drei Summanden zusammensetzt.

* *Beschleunigungs-Term*: Ist die Bewegung des Agenten ungestört, würde er sich mit seiner gewünschten Geschwindigkeit v_0 direkt auf sein Ziel zubewegen. Jede Abweichung hiervon führt durch diesen Term dazu, dass es zu einer Kraft in Richtung des Ziels \mathbf{x}_i^0 kommt.

** *Territorialer-Effekt Term*: Jede Person umgibt ein privater Bereich, auf deren Eindringen durch andere Personen, sie mit Unbehagen und Abstoßung reagiert. Um dieses Verhalten zu modellieren, wird die Beeinflussung und Interaktion zweier Agenten durch Gleichung (2.8) beschrieben. Durch Aufsummierung all dieser einzelnen Kräfte wird schlussendlich dieser territoriale Effekt generiert.

$$\mathbf{f}_{ij}(\mathbf{x}_i(t), \mathbf{x}_j(t)) = -\nabla A(d_{ij} - D)^{-B} \quad (2.8)$$

- B ... Konstante
 d_{ij} ... Distanz zwischen Agenten i und j
 D ... Durchmesser der persönlichen Umgebung
 A ... eine monoton fallende Funktion

*** *Begrenzungs-Term*: Nicht nur umgebende Personen beeinflussen die Richtungsfindung, ebenso Hindernisse wie Wände oder Säulen üben Kräfte auf die einzelne Personen aus und zwingen sie in bestimmte Richtungen. Diese Kraft wird durch Gleichung (2.9) ausgedrückt.

$$\mathbf{f}_b(\mathbf{x}_i) = -\nabla A(d_{i\perp} - D/2)^{-B} \quad (2.9)$$

- $d_{i\perp}$... kürzeste Distanz zum nächstgelegenen Hindernis

All die hier kurz angesprochenen Summanden beeinflussen nun die Entscheidung über die nächste Richtung, die der Agent einschlägt. In der Praxis hat sich dieses Modell als taugliches Instrument für die Simulation von Fußgänger-Flüssen etabliert, das unter anderem das Phänomen der Selbstorganisation zeigt. Kritikpunkte sind jedoch, dass Kollisionen bei hohen Dichten bei dieser Methode nicht ausgeschlossen sind.

2.7 Magnetic Force Model

Ähnlich wie das Social Force Model arbeitet das Magnetic Force Model, das von Shigeyuki Okazaki, Satoshi Matsushita und Chikashi Yamamoto [12, 23] entwickelt wurde, mit Kräften, die in Abhängigkeit von Entfernung und Anzahl der umgebenden Agenten und Hindernisse ermittelt werden. Um die abstoßenden Kräfte, die Agenten

untereinander und gegenüber Barrieren haben, zu modellieren, wird ihnen eine positive magnetische Polarität zugeordnet, während das Zielgebiet mit der entgegengesetzten negativen Polarität initialisiert wird. Die Agenten werden dadurch, in Relation zur Intensität der Ladung und der Entfernung, zu ihrem Ziel hingezogen, während sie untereinander abstoßende Kräfte ausüben.

$$\mathbf{F} = \frac{k \cdot q_1 \cdot q_2 \cdot \mathbf{r}}{r^3} \quad (2.10)$$

\mathbf{F}	...	Magnet-Kraft
k	...	Konstante
q_1	...	Ladung des ersten Pols
q_2	...	Ladung des zweiten Pols
\mathbf{r}	...	Richtungsvektor vom Pol eins zu Pol zwei
r	...	Länge des Richtungsvektors

Die als Coulombsches Gesetz bekannte Gleichung (2.10) beschreibt die dabei entstehenden anziehenden oder abstoßenden Kräfte, die die einzelnen Pole aufeinander ausüben. Eine zusätzliche zweite abstoßende Kraft soll des Weiteren Kollisionen der einzelnen Agenten untereinander in Bewegung vermeiden.

$$\mathbf{a} = \mathbf{VA} \cdot \cos(\alpha) \cdot \tan(\beta) \quad (2.11)$$

Gleichung (2.11) beschreibt die Beschleunigung, die Person A in der Abbildung 2.9 durch die kreuzende Person B erfährt. \mathbf{AC} stellt die Verbindungslinie von A zu einem Kreis mit Radius 0.6 m um B herum dar, der den eingenommenen Platz repräsentiert.

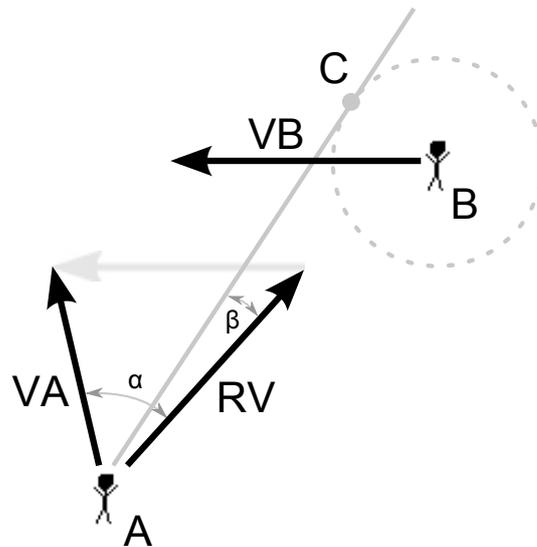


Abbildung 2.9: Beschleunigung im Magnetic Force Model

- a** ... resultierende Beschleunigung
- A** ... Agent A
- B** ... Agent B
- VA** ... Geschwindigkeitsvektor A
- VB** ... Geschwindigkeitsvektor B
- RV** ... relative Geschwindigkeit
- α ... Winkel zwischen **RV** und **VA**
- β ... Winkel zwischen **RV** und **AC**

Die Summe dieser Kräfte wirkt zu jeder Zeit auf den einzelnen Agenten ein und bestimmt so dessen Geschwindigkeit und dessen lokale Richtung. Hindernisse, wie Wände und Säulen, sind in diesem Model als einzelne Punkte gegeben und nur zur Visualisierung als Linien dargestellt. Ist das Ziel des Agenten nicht auf direktem Wege zu erreichen, sind Stützpunkte erforderlich, die als temporäres Ziel dienen und deren magnetische Kraft für die einzelne Agenten ein- und ausgeschaltet wird.

2.8 Queuing Network Model

Das *Queuing Network Model* [16] ist eine relativ einfache Art der Fußgängerfluss-Simulation und unterscheidet sich grundlegend von den vorhergehenden agentenbasierten Methoden. Grundlegende Idee ist die Kopplung mehrerer Queues zu einem Netzwerk, das ein Gebäude oder ein Gebiet in vereinfachter Form nachahmt.

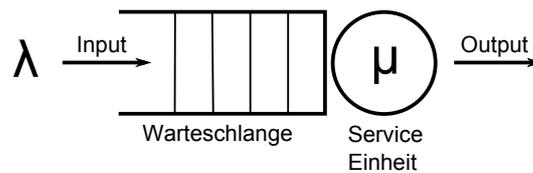


Abbildung 2.10: *Warteschlangen Knoten*

Abbildung 2.10 illustriert einen einzelnen Knoten mit Warteschlange und anschließender Service-Einheit. Eintretende Events oder Personen müssen in der Queue nach dem First-Come-First-Serve Prinzip warten, bis sie von der Service-Einheit abgearbeitet werden. Im Queuing Network Modell wird jeder Raum beziehungsweise jeder Korridor als Knoten angesehen, der, je nach Topologie der realen Umgebung, mit anderen Knoten verlinkt ist. Jede Person kann sich nun, in Abhängigkeit der Warteschlangen und der Geschwindigkeit der Service-Einheit, durch die einzelnen Knoten zu ihrem Ziel bewegen.

Kapitel 3

Das Modell

3.1 Einführung

Bei der in diesem und den folgenden Kapiteln vorgestellten Methode handelt es sich grundsätzlich um ein agentenbasiertes Modell. Um jedoch Nachteile diverser Methoden zu umgehen, wurden gängige Methoden kombiniert und neue hinzugefügt, um ein möglichst effizientes Modell zu entwickeln, dessen Tauglichkeit im Anschluss durch eine Implementierung in JAVA gezeigt wird.

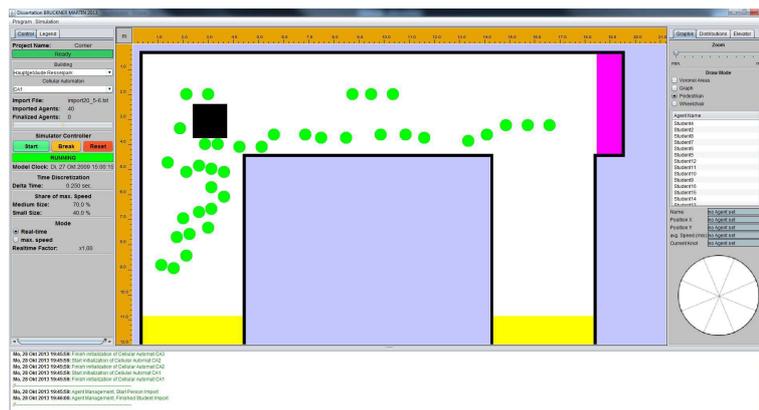


Abbildung 3.1: Screenshot Simulator

Abbildung 3.1 zeigt einen Screenshot und den Ausblick auf das, was in diesem und den folgenden Kapiteln thematisiert wird. Zu sehen ist das GUI (*Graphic User Interface*) der Implementierung im Zuge eines Simulations-Durchlaufes.

Die Hauptelemente eines agentenbasierten Modells sind, wie in Abschnitt 2.5 erwähnt, der Agent und die Umgebung, mit der und in der er mit anderen Agenten interagiert. Um diese Elemente jedoch in ein Modell umsetzen zu können, sind vorab einige Fragen offen, die man sich stellen muss, bevor man mit der Arbeit beginnen kann.

Da es sich bei dem Modell um eine Raum-und Zeit-diskretisierte Umsetzung handelt, wird die Wahl der Auflösungen dieser beiden Größen eine nicht unwesentliche Rolle spielen. Des Weiteren soll es in diesem Modell möglich sein, realitätsnahe Gebäude, die über mehrere Ebenen und Aufzüge verfügen, zu betrachten. Diese Umgebung stellt jedoch wiederum spezielle Anforderungen an das Routing dar, da meist eine Vielzahl von Wegen zu bewerten sein wird, die zwar im Endeffekt alle zum gewünschten Ziel führen, deren jeweiligen Eigenschaften jedoch zu berücksichtigen sein werden.

Es wird im Anschluss im Detail auf den Modellierungsprozess eingegangen, ohne jedoch konkret auf dessen Implementierung Bezug zu nehmen.

3.2 Die Umgebung

Als Umgebung, in der und mit der die Agenten in diesem Modell interagieren, wurde ein diskretes Gitter mit quadratischer Form gewählt. Für die Rasterung des Raumes sind in der Literatur mehrere Werte zu finden. Sie orientieren sich jedoch beinahe ohne Ausnahme an der Größe einer realen Person, sodass in der Simulation ein Agent genau eine Zelle besetzt. Einer der gebräuchlichsten Werte hierfür ist 40×40 cm [3, S. 510]. Um jedoch feinere Strukturen wie etwa Türen, Durchgänge oder generell Engstellen genauer auflösen zu können, ist es von Nöten, diese zu verfeinern, was einerseits die Komplexität der Aufgabe erhöht, andererseits aber auch Möglichkeiten im Bereich der Kollisionsprävention bietet. In diesem Modell wurde auf eine Rasterung von $12,5 \times 12,5$

cm gesetzt, sodass pro Quadratmeter 64 Zellen zum Einsatz kommen. Der Platzbedarf eines Agenten beschränkt sich daher nicht auf eine Zelle, sondern erweitert sich auf 4×4 Zellen, was einer Fläche von $0,25 \text{ m}^2$ entspricht. Da eine Anforderung an die Simulation die Abbildung von reellen Gebäuden mit mehreren Stockwerken war, diese jedoch in den meisten Fällen nicht nur auf lediglich eine Fläche abzubilden sind, wurde das Modell so angelegt, dass sich verschiedene diskrete Flächen an definierten Punkten verbinden lassen. Diese Verbindung ist modelltechnisch so realisiert, dass sich zwei diskrete Flächen an den definierten Verbindungen die jeweiligen Zellen teilen. Ein Agent, der diese speziellen Zellen betritt, existiert somit in beiden „Welten“ für die Dauer, für die er sich in dieser Verbindungszone aufhält. Dieses Konstrukt, das etwa auf Treppen zu finden sein wird, ermöglicht es den Agenten nun, die Flächen und somit die sie repräsentierenden Ebenen zu wechseln und sich somit ihrem Ziel zu nähern.

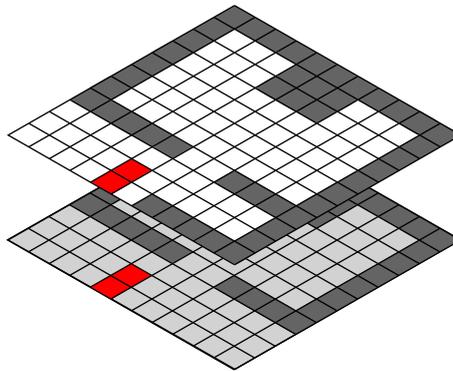


Abbildung 3.2: *Visualisierung verschiedener Ebenen*

Die Darstellung in Abbildung 3.2 soll eine bessere Vorstellung darüber vermitteln wie zwei Etagen, die separat diskretisiert wurden, untereinander verbunden sind. Die beiden roten Flächen in der unteren und oberen Etage stellt dabei die eben erwähnte Verbindung dar, die einen Wechsel der Ebenen ermöglichen soll.

Zusätzlich zu der oben erwähnten Möglichkeit, die Etagen zu wechseln, stehen den Agenten des Weiteren Aufzüge zur Verfügung, die ebenso in das Modell integriert wurden. Diese Erweiterung besteht Modelltechnisch lediglich aus Flächen, die im Grundriss eines Gebäudes die Aufzugsschächte darstellen, und einer Logik, die in Abhängigkeit verschiedenster Parameter wartende Agenten aus der Simulation entfernt und sie nach einer simulierten Zeit in der Ziel-Etage wieder einfügt. Auf die Details der modelltechnischen Umsetzung wird später in diesem Kapitel noch näher eingegangen.

3.3 Der Agent

Auf die Definition eines Agenten wurde bereits im Abschnitt 2.5 im Detail eingegangen. Es wurde erwähnt, dass es sich dabei um ein diskretes Individuum mit einer Menge von Eigenschaften handelt, die wir nun im Folgenden genauer spezifizieren. Die Umrandungen unterteilen die Parameter in solche, die ihren Wert durch stochastische Prozesse erlangen, und denjenigen, die aus anderen Parametern durch Berechnungen hervorgehen.

Stochastische Werte

- Geschlecht
- Alter
- Rollstuhlfahrer

Gefolgerte Werte

- individuelle Geschwindigkeit in der Ebene
- individuelle Geschwindigkeit Stiege rauf
- individuelle Geschwindigkeit Stiege runter

Auf die Zusammenhänge der beiden Parameterblöcke wird in Abschnitt 3.4, der sich mit der Geschwindigkeit von Personenströmen befasst, noch im Detail eingegangen.

Eine Anforderung an das Modell war die Simulation von Personen mit eingeschränkter Mobilität. Um diese Forderung umzusetzen, wurden spezielle Agenten definiert, die sich von Standard-Agenten durch unterschiedliche Parameter und speziellen Algorithmen unterscheiden. Die augenscheinlichste Veränderung ist die Anzahl der Zellen, 6×6

anstelle von 4×4 , die eingenommen werden. Des Weiteren besitzen sie eine geänderte persönliche Geschwindigkeit und spezielle Routingalgorithmen, die sie um Hindernisse, wie Stiegen, herumführen. Die näheren Details der Modellierung werden in den jeweiligen Abschnitten der Themen genauer erörtert.

Um das Verhalten von Menschen möglichst detailgetreu nachzuahmen, wurde das Verhalten implementiert, dass der momentan benötigte Platz, zum Beispiel beim Passieren einer entgegenkommenden Person, kurzzeitig reduziert werden kann. Das in der realen Welt durch Drehen der Schulter stattfindende Vorgehen wurde in das Modell übernommen, sodass es dem Agenten möglich ist, seinen Platzbedarf auf 3×3 Zellen, mit einhergehender Reduktion seiner Geschwindigkeit, zu verkleinern. Die folgenden Abbildungen illustrieren diese Verkleinerung und die Änderung der Farbe, die die Agenten zur besseren Unterscheidung der einzelnen Zustände im grafischen Userinterface haben. Trotz der runden Darstellung benötigen die einzelnen Personen natürlich die kompletten Zellen.

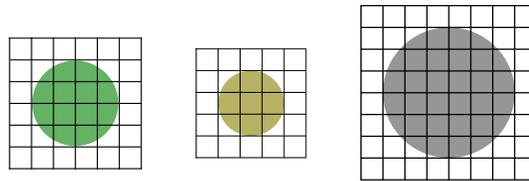


Abbildung 3.3: *Platzbedarf Agenten*

Die in Abbildung 3.3 dargestellten Größen entsprechen von links nach rechts der eines Standard-Agenten, der eines durch ein Hindernis kurzfristig verkleinerten Agenten und der eines Rollstuhlfahrers. Es sei hier nur erwähnt, dass die Größe eines Agenten im Rollstuhl fix mit 6×6 Zellen vorgegeben ist und selbstverständlich nicht reduziert werden kann.

3.4 Gehgeschwindigkeit

Die Geschwindigkeit, mit der sich Agenten fortbewegen, wird maßgeblich durch deren Alter und Geschlecht sowie durch die Umgebung beeinflusst, durch die sie sich bewegen. Während Alter und Geschlecht im Zuge eines Simulationslaufes konstant bleiben, kann der Agent verschiedene Umgebungen betreten und somit seine mögliche Maximalgeschwindigkeit ändern. Aus Effizienzgründen wurde die Anzahl der Umgebungen auf drei Typen beschränkt.

1. Korridor
2. Stiege rauf
3. Stiege runter

Des Weiteren ist die Interaktion zwischen den einzelnen Individuen von Bedeutung, wobei hier unterschieden werden muss zwischen der Personendichte und dem direkten Kontakt, zum Beispiel durch Gegenverkehr. Die Dichte hat direkten Einfluss, ausgedrückt durch eine Funktion, die später noch besprochen wird, auf die Maximalgeschwindigkeit eines Agenten, während direkter Kontakt die mögliche Höchstgeschwindigkeit nicht beeinflusst, sondern das Tempo durch Blockaden und Wartezeiten reduziert. Hierbei sei noch erwähnt, dass Aspekte wie Beschleunigungs- bzw. Bremszeiten zwar Einfluss auf die Simulation haben, sie jedoch vernachlässigbar sind und daher nicht umgesetzt wurden.

3.4.1 Abhängigkeiten von Alter und Geschlecht

Da wie oben bereits erwähnt Alter und Geschlecht einer Person maßgeblichen Einfluss auf ihr vorankommen haben, wollen wir im Folgenden eine Verteilung vorstellen die sich als Standardwerk in diesem Bereich etabliert hat und auch in den *Richtlinien für Mikroskopische Entfluchtungsanalysen (RiMEA)* [2] vorgeschlagen werden. Die Geschwindigkeitsverteilung nach Weidmann beruht auf empirischen Daten und stellt die Geschwindigkeit von Frauen und Männern in Relation zur ihrem Alter dar.

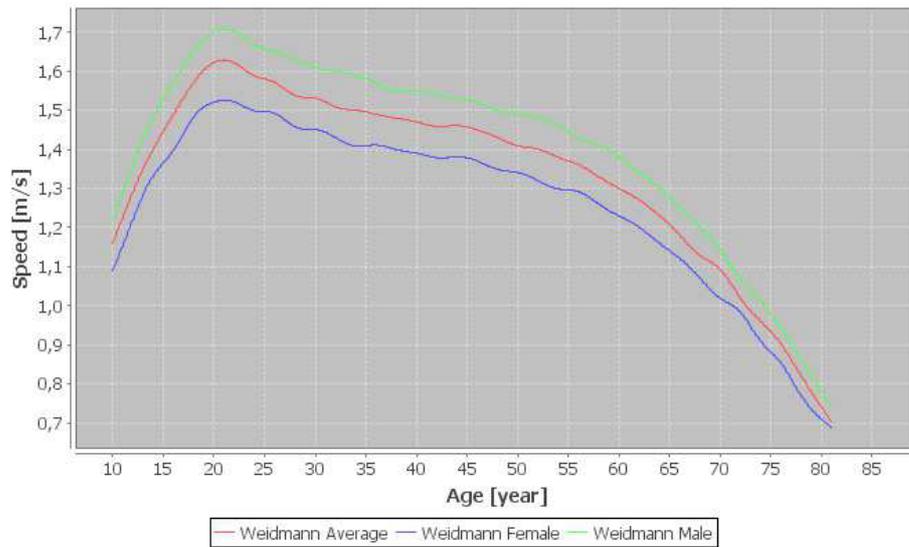


Abbildung 3.4: *Weidmann-Verteilung*

In der Abbildung 3.4 sind nun die Geschwindigkeitsverteilungen für Männer und Frauen dargestellt sowie deren Durchschnitt. Somit ergibt sich folgende Tabelle für einzelne Altersgruppen.

Personengruppe	Gehgeschwindigkeit in der Ebene (m/s)	
	Minimum	Maximum
Unter 30 Jahren	1,15	1,62
30 - 50 Jahren	1,41	1,53
Über 50 Jahren	0,68	1,41

Tabelle 3.1: *Gehgeschwindigkeiten in der Ebene nach Weidmann*

Die somit ermittelte Geschwindigkeit von Männern ist im Durchschnitt um 10,9 % höher als die von Frauen. Somit ergibt sich eine mittlere Gehgeschwindigkeit von 1,41 m/s beziehungsweise 1,27 m/s für Männer und Frauen.

3.4.2 Geschwindigkeit auf Stiegen

Die Geschwindigkeit auf Stiegen unterscheidet sich natürlich maßgeblich von der in der Ebene, sie variiert aber auch je nach Art der Stiegen. Untersuchungen zeigen, dass unterschiedliche Höhen-zu-Breiten Verhältnisse sowie der Gradient der Stufen zu unterschiedlichen Geschwindigkeiten führen. [5] Diese Tatsache sei hier nur erwähnt, es wird in der Modellierung, aus Gründen der Komplexität, stets von einer Standard-Stiege ausgegangen, deren begehbare Geschwindigkeit im Folgenden noch dargelegt wird. Des Weiteren ist zu beachten, dass sich alle Angaben von Distanzen und Geschwindigkeiten hierbei nur auf die horizontale Komponente der Stiegen beziehen.

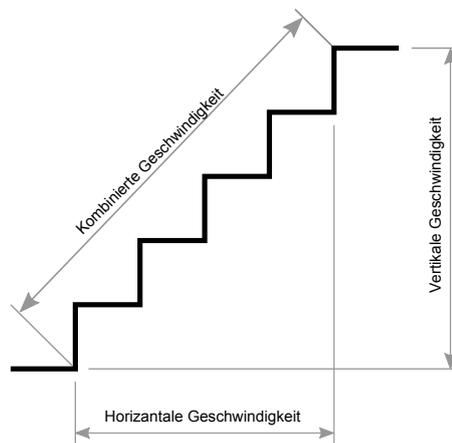


Abbildung 3.5: Geschwindigkeitskomponenten auf Treppen [5]

Dies soll in Abbildung 3.5 weiter verdeutlicht werden. Da der Grundstruktur des Modells ein 2-dimensionales Gitter zugrundeliegt, ist für diesen Zweck ausschließlich die horizontale Komponente von Interesse. Da es auch für die Fortbewegung auf Treppen Empfehlungen für die Geschwindigkeiten in der *RiMEA* gibt, werden diese in der Simulation übernommen.

- (1) Innentreppe, Steigungsverhältnis 17,8 cm / 28,6 cm
- (2) Außentreppe, Steigungsverhältnis 15,2 cm / 30,5 cm

Personengruppe	Mittlere Gehgeschwindigkeit auf Treppen (m/s)			
	Treppe abwärts		Treppe aufwärts	
Stiegentyp	(1)	(2)	(1)	(2)
Unter 30 Jahren	0,76	0,81	0,55	0,58
30 - 50 Jahren	0,65	0,78	0,50	0,58
Über 50 Jahren	0,55	0,59	0,42	0,42

Tabelle 3.2: *Mittlere Gehgeschwindigkeiten auf Treppen [2, S. 7]*

Der in dem Modell verwendete Typ ist bis zu diesem Zeitpunkt ausschließlich die Innenstiege. Mit Hilfe des arithmetischen Mittels der minimalen und maximalen Werte aus Tabelle 1 und der zweiten Spalte aus Tabelle 2 ergibt sich, dass die durchschnittliche Geschwindigkeit für das Hinabgehen einer Stiege der drei jeweiligen Altersgruppen auf etwa 54 %, 46 %, beziehungsweise 52 % der Geschwindigkeit dessen, was sie in der Ebene beträgt, sinkt. Somit erfolgt eine Reduktion über alle Klassen hinweg auf etwa 50.7 %. Eine analoge Rechnung für das Aufwärtsgehen ergibt eine Verminderung auf 38 % des Vorrankommens in der Ebene.

3.4.3 Abhängigkeit von Geschwindigkeit und Dichte

So wie Alter, Geschlecht und Umgebung beeinflusst auch die Fußgängerdichte das Vorrankommen von Personen ungemein. Den Zusammenhang dieser beiden Werte beschreibt die Näherungsrelation von Kladek wie folgt.

$$v_h(\rho) = v_h^0 \left(1 - e^{-\gamma \left(\frac{1}{\rho} - \frac{1}{\rho_{max}}\right)}\right) \quad (3.1)$$

- $v_h(\rho)$ Geschwindigkeit bei Dichte ρ [m/s]
 v_h^0 Geschwindigkeit auf freier Fläche [m/s]
 ρ Momentane Dichte [P/m²]
 ρ_{max} Maximale Dichte [P/m²]
 γ empirische Eichkonstante [P/m²]

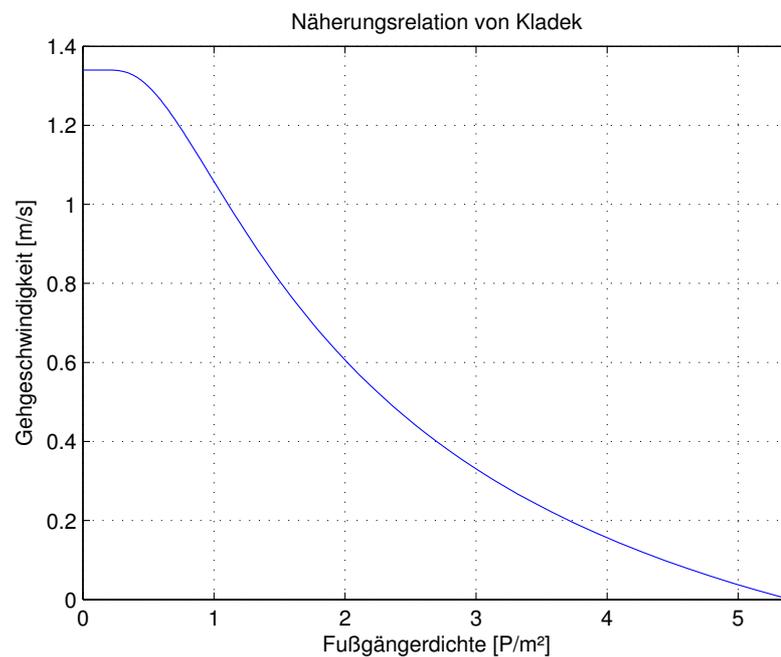


Abbildung 3.6: Näherungsrelation von Kladek bei $v_h^0 = 1,34$ m/s

Mit den Werten $\rho_{max}=5,4$, $\gamma=1,913$ und $v_h^0=1,34$ ergibt sich somit der in Abbildung 3.6 dargestellte Funktionsgraph, der als Grundlage zur Ermittlung der maximalen Geschwindigkeit bei einer vorgegebenen Dichte dient.

3.4.4 Geschwindigkeit von Rollstuhlfahrern

Im Gegensatz zu Fußgängern ist die Geschwindigkeit, mit der sich Rollstuhlfahrer fortbewegen, relativ schwierig vorherzusagen. Zu unterschiedlich und vielfältig sind die Verletzungen, die ein Leben im Rollstuhl notwendig machen, daher ist es auch nicht verwunderlich, dass in der Literatur verschiedenste Durchschnittsgeschwindigkeiten zu finden sind. Sie bewegen sich im Bereich von 0,5 m/s [28, S. 87] über 0,69 m/s [19, S. 380] bis hin zu 1,0 m/s [21, S. 1247]. Da aus diesem Grund keine genaueren Daten vorliegen wurde in dem Modell eine stetige Gleichverteilung für die Geschwindigkeiten von Rollstuhlfahrern angenommen, deren Intervallgrenzen sich an den minimalen und maximalen Werten in der Literatur orientieren.

$$F(x) = \mathcal{U}(0, 54; 1, 0) \quad (3.2)$$

3.4.5 Bestimmung der momentanen Geschwindigkeit

Mit der in diesem Abschnitt besprochenen Methode lassen sich nun die verbleibenden Parameter der Agenten, die im Punkt 4.4.1 beschrieben worden sind, initialisieren. Um nun die momentane Geschwindigkeit der jeweiligen Agenten in jedem Schritt zu ermitteln, ist zu beachten, dass die Größe der einzelnen Agenten variiert, sie aber in jedem Fall grösser als eine Zelle ist. Somit ist es möglich, dass er sich auf Feldern befinden, die unterschiedliche Geschwindigkeiten zulassen, wie etwa beim Übergang von Korridor zur Stiege. Um mit diesem Umstand umzugehen, wird die maximale Geschwindigkeit v_h^0 aus dem arithmetischen Mittel der besetzten Felder berechnet. Nach Bestimmung der Personendichte kann anschließend $v_h(\rho)$ unter Zuhilfenahme der Funktion von Kladek ermittelt werden, die im weiteren Verlaufe der Simulation noch benötigt wird.

3.5 Routing

Der Routing-Prozess beschäftigt sich mit der Navigation der einzelnen Agenten in der virtuellen Welt der diskreten Gebäudeteile und das, wenn möglich, in einer effizienten Art und Weise. Das Ziel hierbei ist es, vorab einen Plan zu erstellen mit dessen Hilfe es möglich ist, ohne große Umwege von Punkt A zu Punkt B zu gelangen. Eine Schwierigkeit ist, dass wie in Abschnitt 3.2 erwähnt wurde es sich nicht nur um eine einzelne Fläche handelt, auf der sich die Agenten bewegen, sondern es eine beliebige Anzahl von untereinander verbundenen Gebiete sind, die ihnen zur Verfügung stehen. Dies stellt nun einige Anforderungen an den Wegfindungsalgorithmus dar, der in der Lage sein muss, den geeignetsten Weg über dieses verzweigte System zu ermitteln. Um dies zu gewährleisten, wurde das Problem auf zwei Teilprobleme reduziert, die separat gelöst werden, wobei jeweils verschiedene Algorithmen zum Einsatz kommen. Die Aufgaben und Herangehensweisen der beiden Aufgabenstellungen, im Folgenden globale und lokale Wegsuche genannt, werden anschließend erläutert.

3.5.1 Globale Wegsuche

In diesem Abschnitt wird die Methodik besprochen, um Wege auch in dem vernetzten System von zellulären Flächen zu finden. Vorab ist zu erwähnen, dass sich globale und lokal Suche abwechseln, da im Laufe der Simulation Daten über den momentanen Zustand der Umgebung eines jeden Agenten gesammelt werden, die zu früheren Zeitpunkten noch nicht bekannt waren. Hierbei startet stets die globale Suche, die als Ergebnis eine Liste mit Zwischenzielen liefert, die im Anschluss an den lokalen Algorithmus zur weiteren Verarbeitung übergeben wird.

Grundlage des globalen Algorithmus ist ein ungerichteter Graph, der die Topologie der in der Simulation in Verwendung befindlichen Umgebung nachbildet und so die Möglichkeit bietet, mit gängigen Methoden verschiedenste Routen und Wege zu finden. Da die Graphentheorie in diesem Abschnitt eine wichtige Rolle spielt, wird nachfolgend eine kurze Zusammenfassung der benötigten Begriffe und Definitionen dargestellt.

3.5.1.1 Graphentheorie

Ein Graph $G = (V, E)$ besteht aus einer nichtleeren Menge von Knoten (*Vertex*) und einer beliebigen Menge von Kanten (*Edge*) mit $V \cap E = \emptyset$, wobei jeder Kante $e \in E$ ein eindeutiges Paar $(a, b) \in V^2$ zugeordnet ist.

Eigenschaften von Graphen

- G ist **ungerichtet** \Leftrightarrow jedes zu $e \in E$ zugeordnete Paar ist ungeordnet.
- G ist **gerichtet** \Leftrightarrow jedes zu $e \in E$ zugeordnete Paar ist geordnet.
- G ist **endlich** \Leftrightarrow die Mächtigkeit beider Mengen ist endlich.
- e, f **parallel** $\Leftrightarrow e, f \in E$ besitzen dasselbe Knotenpaar.
- e ist **Schlinge** $\Leftrightarrow e \in E, e = (a, a)$
- G ist **schlicht** $\Leftrightarrow G$ enthält keine Schlingen und keine parallelen Kanten.

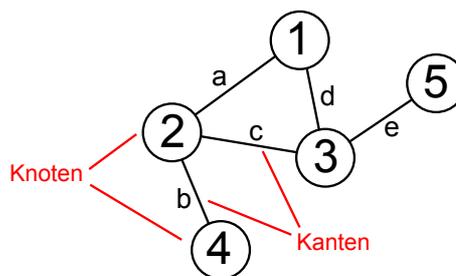


Abbildung 3.7: Beispiel eines Graphen

Definition Sei $G = (V, E)$ ein ungerichteter Graph.

$e_1, e_2, \dots, e_k \in E$ bilden eine **Kantenfolge**, wenn für alle $i = 2, 3, \dots, k - 1$ gilt, dass e_i

mit e_{i-1} und e_{i+1} einen Knoten gemeinsam hat.

Kantenzug \Leftrightarrow Kantenfolge die ausschließlich aus verschiedenen Kanten besteht.

Geschlossener Kantenzug \Leftrightarrow Kantenzug, bei dem Startknoten und Endknoten ident sind.

Offener Kantenzug \Leftrightarrow Kantenzug mit unterschiedlichen Startknoten und Endknoten.

Weg \Leftrightarrow ein offener Kantenzug, in dem kein Knoten mehrfach vorhanden ist.

Definition Sei $G = (V, E)$ ein Graph ohne parallele Kanten.

Eine Kantengewichtsfunktion ist eine Funktion, die jeder Kante eine reelle Zahl zuordnet.

$$d : E \rightarrow \mathbb{R} \quad (3.3)$$

Das Kantengewicht einer Kante wird mit $d(e)$ oder $d(a, b)$ bezeichnet.

Ein Graph mit einer Kantengewichtsfunktion ist ein **gewichteter Graph**.

3.5.1.2 Modell Graph

In diesem Modell wird ein ungerichteter gewichteter Graph zur globalen Wegsuche eingesetzt. Die Knoten sind hierbei definierte Objekte mit einer genau festgelegten Position in den zellulären Flächen, die Kanten die Wege, die zwischen ihnen liegen. Die Gewichtungsfunktion ist eine Funktion, die jeder Kante die Zeit zuordnet, die benötigt wird, um vom Start zum Endknoten der jeweiligen Kante zu gelangen. Der Graph ist des Weiteren dynamisch ausgelegt, das bedeutet, dass die Gehzeit, die für jede Kante anfällt, sich stets der Dichte des Verkehrs anpassen. Mehr darüber in Abschnitt 3.5.1.3. Anhand der Definition der Gewichtungsfunktion ist ersichtlich, dass das Kantengewicht variiert, aber stets ≥ 0 ist, was für den Suchalgorithmus von essenzieller Bedeutung ist.

3.5.1.3 Gewichtungsfunktion

Wie in Punkt 3.5.1.2 erwähnt wurde, weist die Gewichtungsfunktion jeder Kante einen spezifischen Wert zu, der im Fall des Modells für die Zeit steht die für die Kante oder präziser ausgedrückt für den Weg, den sie repräsentiert, benötigt wird. Da dieser Wert von diversen Faktoren abhängt, soll im Folgenden ein kurzer Einblick über die Funktionsweise der Gewichtungsfunktion gegeben werden. Den Hauptfaktor hierbei stellt der Verkehr in der Nähe der zu begehenden Kante dar. Als Definition für den Begriff *Nähe* wird der Raum zu Beginn mittels einer Voronoi-Zerlegung in einzelne Segmente unterteilt.

Voronoi-Diagramm

Ein Voronoi-Diagramm wird durch eine gegebene Menge von Punkten, die das Zentrum der einzelnen Segmente darstellt, bestimmt. Jeder weitere Punkt im Raum wird demjenigen Segment zugeordnet, zu dessen Zentrum es, in Bezug zur euklidischen Metrik, die geringste Distanz hat. Aus denjenigen Punkten, die zu mehr als einem Zentrum die geringste Distanz haben, entsteht das Voronoi-Diagramm.

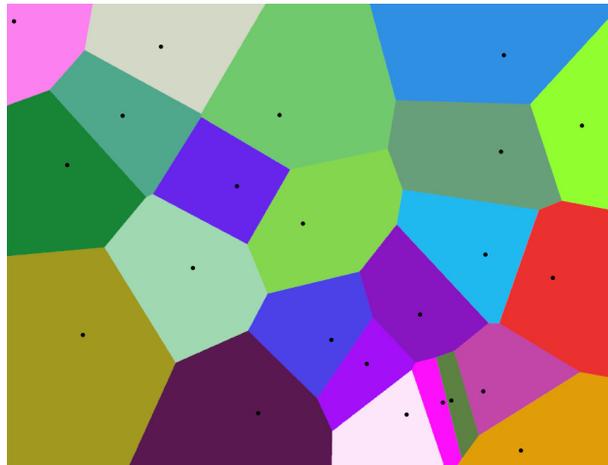


Abbildung 3.8: *Beispiel eines Voronoi-Diagramms*

Abbildung 3.8 illustriert die Zerlegung eines zweidimensionalen Raumes mithilfe des Voronoi-Algorithmus. Zu erkennen sind die unterschiedlich eingefärbten Segmente mit den jeweiligen, schwarz markierten, Zentren im Inneren. Die Grenzen zwischen den Regionen bilden hierbei das Voronoi-Diagramm.

Ermittlung des Kantengewichtes

Nach der Einführung einer Definition für den Begriff *Nähe* kann die zelluläre Fläche mithilfe des oben dargelegten Voronoi-Diagramms in einzelne Regionen zerlegt werden. Dieser Vorgang wird in der Implementierung der Routingalgorithmen noch spezifiziert.

1. Das Gewicht aller Kanten wird auf den Wert gesetzt, der der Zeit entspricht, die der zu routende Agent benötigen würde, wenn er den repräsentierenden Weg der Kante ohne Behinderung mit seiner persönlichen Geschwindigkeit gehen würde. Hier wird die Annahme des Agenten modelliert, dass Wege, die nicht von ihm einsehbar sind, wahrscheinlich ohne größere Behinderung passierbar sind.
2. Es werden alle Agenten, die für den zu routenden einsehbar sind, mithilfe der Voronoi-Zerlegung des Raumes ihrer jeweiligen Kante zugeordnet.
3. Anhand der in den zugeordneten Agenten gespeicherten durchschnittlichen Geschwindigkeit kann für das jeweilige Voronoi-Segment und somit für die jeweilige Kante eine Schätzung für das Vorankommen abgegeben werden und als Erwartungswert der Zeit in der Kante gespeichert werden.

Nachdem der nun personalisierte Graph mit den neuen Kantengewichten versehen ist, kann mit der Suche nach dem schnellsten Weg begonnen werden.

3.5.1.4 Dijkstra Algorithmus

Der im vorigen Abschnitt 3.5.1.2 erstellte Graph kann jetzt als Hilfe zum Auffinden von Wegen herangezogen werden. Hierfür wird der Dijkstra-Algorithmus verwendet, da die Voraussetzung des nicht negativen Kantengewichtes laut Definition erfüllt ist. Der Dijkstra-Algorithmus, benannt nach seinem Erfinder *Edsger W. Dijkstra*, ist ein Suchalgorithmus, der in einem gewichteten Graphen, deren Kantengewichte positiv sein müssen, denjenigen Weg zu gegebenen Start und Zielknoten ermittelt, sodass die Summe der Kantengewichte, die dabei passiert werden, minimal ist. Zur Durchführung benötigt dabei jeder Knoten die Eigenschaften *Gewicht*, die das momentane Gewicht des Knoten speichert, *Vorgänger*, die den Vorgänger des Knoten beinhaltet, sowie einen *Status*, der permanent oder temporär sein kann. Der Status zeigt an, ob der Knoten bereits aktiv bei der Suche beteiligt war, was wiederum bedeutet, dass die kürzeste Distanz zu ihm schon ermittelt wurde und es keinen kürzeren Weg zu ihm gibt. Dieses Ereignis wird im Status mit permanent markiert. Der Ablauf der Suche wird im Folgenden Schritt für Schritt erörtert.

1. Initialisierung aller Knoten des Graphen mit $Vorgänger = null$, $Distanz = \infty$, $Status = temporär$
2. Setzen der Werte des Startknotens, $Distanz = 0$ und $Status = permanent$
3. Startknoten wird als aktiver Knoten ausgewählt
4. Berechne die temporären Distanzen aller Nachbarknoten des aktiven Knotens, durch Addition von dessen Distanz mit den Kantengewichten
5. Wenn die berechnete Distanz für einen Knoten kleiner als die aktuelle ist, aktualisiere die Distanz und setze den momentan aktiven Knoten als Vorgänger
6. Wähle einen Knoten mit minimaler temporärer Distanz als aktiv und markiere seine Distanz als permanent
7. Wiederhole 4-7, bis es keinen Knoten mit permanenter Distanz gibt, dessen Nach-

barn noch temporäre Distanzen haben.

Somit berechnet der Algorithmus, ausgehend vom Startknoten, alle Weg mit geringstem Gewicht zu den restlichen Knoten. Ist nur der Weg zu einem bestimmten Knoten von Interesse, kann die Suche abgebrochen werden, sobald der Zielknoten als aktiv markiert wurde. Nachfolgendes Beispiel soll den Prozess weiter verdeutlichen.

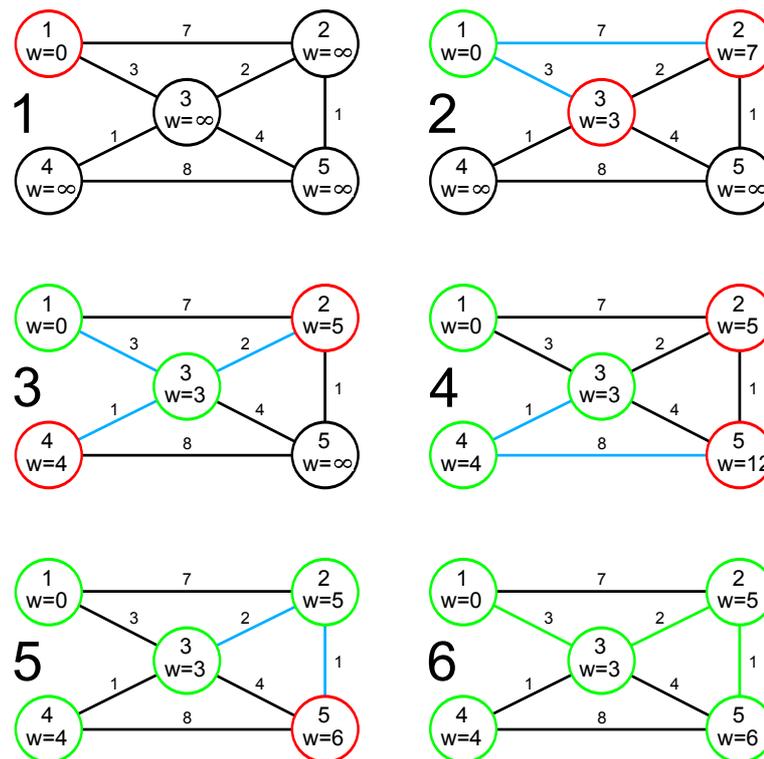


Abbildung 3.9: *Beispiel Dijkstra Suche*

Das in Abbildung 3.9 dargestellte Beispiel stellt die Suche des Pfades mit dem geringsten Kantengewicht von Knoten 1 zu 5 dar. Rot markierte Knoten haben den Status temporär, wurden aber bereits einmal aktualisiert, während grüne Knoten permanent sind, es existiert daher schon der leichteste Weg zu ihnen. Die verbleibenden Knoten, schwarz markiert, sind noch in ihrem Startzustand.

3.5.1.5 Erstellen der Zwischenziel List

Mit Hilfe der in den vorigen Punkten erörterten Methoden des personalisierten Graphen und des Dijkstra-Algorithmus ist es jetzt möglich, eine Liste von Knoten zu erstellen, deren den Knoten zugrundeliegenden Elemente dem Agenten Zwischenziele liefern, die ihm bei sequentieller Abarbeitung an sein gewünschtes Ziel bringen. Diese Liste dient nun dem, im nächsten Abschnitt ausführlich erklärten, Algorithmus der lokalen Wegsuche als Input.

3.5.2 Lokale Wegsuche

Die lokale Wegsuche beschäftigt sich ausschließlich mit der Orientierung innerhalb einer diskreten Fläche, wobei sich Start- und Endpunkte von denen der globalen Suche unterscheiden können. Liegen beide Punkte etwa in zwei unterschiedlichen Flächen, wird die erste lokale Suche den Agenten vom eigentlichen Startpunkt zu einem Punkt führen, der einen Wechsel in das Feld des zweiten Punktes durchführt. Dort angekommen wird der lokale Startpunkt das Objekt sein, durch das der Agent eingefügt wurde, und der Endpunkt kann etwa das globale Ziel oder ein weiterer Übergang sein.

Für die Durchführung dieser Suche wird in vielen Simulatoren häufig auf ein *Static Floor Field* gesetzt, das in Abschnitt 2.4.3 bereits kurz vorgestellt wurde. Da auch im Laufe der Entstehung des Simulators, der dieser Dissertation zugrunde liegt, ausschließlich diese Methode zum Einsatz kam, wird im nächsten Abschnitt näher darauf eingegangen.

3.5.2.1 Static Floor Field

Wie bereits im Abschnitt 2.4.3 erwähnt wurde, handelt es sich bei einem *Static Floor Field* (SFF) um ein, bei der Initialisierung generiertes, Feld mit fixen Werten, die im Laufe der Simulation nicht mehr verändert werden. Jede Zelle enthält dabei einen Wert, der in Relation zur Entfernung zu einem definierten Zielpunkt steht. Für die Initiali-

sierung dieses Feldes gibt es dabei unterschiedlichste Metriken, deren Eigenschaft das Simulationsergebnis erheblich beeinflussen können. Ein Beispiel hierfür sind die Distanzen, die sich in natürlicher Weise aus der Von-Neumann und Moore Nachbarschaft ergeben, die, wie in Abbildung 3.10 und 3.11 gut zu erkennen ist, jedoch nicht isotrop sind.

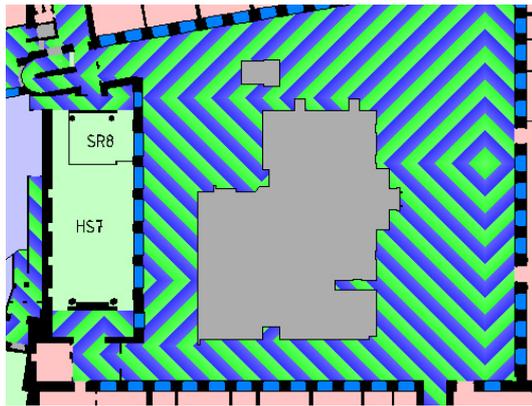


Abbildung 3.10: *Von-Neumann Distanz*

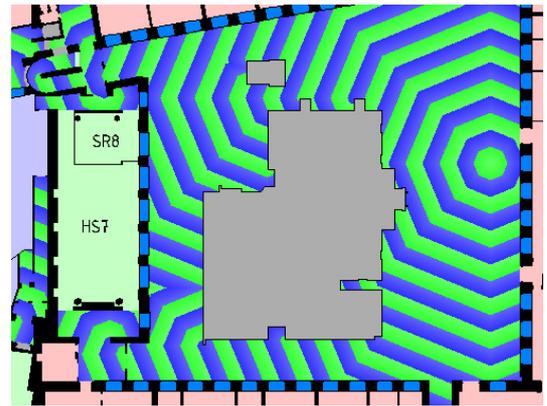
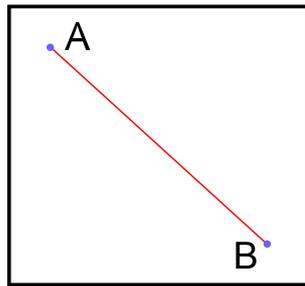
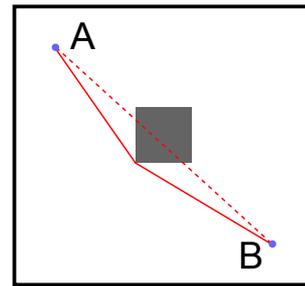


Abbildung 3.11: *Moore Distanz*

Diese anisotrope Eigenschaft der beiden Metriken hat zur Folge, dass es an den Verlängerungen der direkten Nachbarschaft, die bei Von-Neumann 90° und Moore 45° zueinander versetzt sind, zu künstlich geschaffenen Häufungen der Agenten und zur Ausprägung von Ameisenstraßen kommt. Um diesen Effekt zu vermeiden, wurde ein Algorithmus entwickelt, der die euklidische Distanz, L^2 Norm, berechnet, die jede Zelle zu dem Zielpunkt hat. Bei der Berechnung dieser Distanz, die in einem konvexen Raum recht simpel ist, muss in einer Umgebung, die der Realität entspricht und daher Ecken und Hindernisse aufweist, schon etwas genauer darauf geachtet werden, dass die Verbindungsgerade auch tatsächlich existiert und durch kein Hindernis unterbrochen wird. Abbildung 3.12 zeigt ein Beispiel eines konvexen Raumes, in dem zwei beliebige Punkte durch eine Linie verbunden werden können, wobei diese durch kein Hindernis unterbrochen wird.

Abbildung 3.12: *Konvexer Raum*Abbildung 3.13: *Nicht konvexer Raum*

Anders stellt sich die Situation in Abbildung 3.13 dar, in der die kürzeste Distanz zwischen Punkt A und B um das in der Mitte platzierte Hindernis herum führt. Das bedeutet, dass für die Berechnung der euklidischen Distanz auch überprüft werden muss, ob die Verbindungslinie zweier Zellen existiert. Der für diesen Zweck entwickelte Algorithmus, der im Folgenden erörtert wird, stellt dies sicher und befüllt sukzessive, beginnend beim Zielpunkt, das Static Floor Field.

L^2 Initialisierungs-Algorithmus:

Initialisierung der Zellen: Die Distanz aller Zellen wird auf den maximal möglichen Wert gesetzt, der für die Variable zur Verfügung steht.

Algorithm 1: Initialisierung Static Floor Field

```

1 for  $x = 0$  to  $x < sf.f.width$  do
2   for  $y = 0$  to  $y < sf.f.height$  do
3      $sf.f[x][y] = \infty$ 
4   end
5 end
```

Suchen und Initialisieren der Startpunkte: Es ist nicht erforderlich, dass das Ziel des Static Floor Field nur aus einer einzigen Zelle besteht. Im Fall einer Tür oder eines Aufzugs besteht das Ziel aus einer größeren zusammenhängenden Fläche. Ist dies der Fall, müssen mehrere Zielpunkte ermittelt werden, die als Einstiegspunkte für den Algorithmus dienen. Hierfür werden alle Zellen der zusammenhängenden Fläche

durchlaufen und diejenigen ausgewählt, die mindestens eine Nachbarzelle haben, die als Korridor oder Stiege ausgezeichnet sind. Durch diesen Schritt wird die Anzahl der Zellen minimiert, die als Startpunkt in Frage kommen, was zu einer Steigerung der Geschwindigkeit für die nächsten Schritte beiträgt. Die Distanzen der Startzellen wird auf 0 gesetzt.

```
6 minimumPoints = calcStartpoints(ActionElement)
7 for i = 0 to i < minimumPoints.length do
8   | aktivePoint = minimumPoints.get(i)
9   | sff[aktivePoint.x][aktivePoint.y] = 0
10 end
```

Berechnung der euklidischen Distanzen: Die Variable, die in Zeile 11 angelegt und mit 0 initialisiert wird speichert die Distanz, die momentan aktiv ist und die sämtliche Punkte in der Datenstruktur *minimumPoints* besitzen. Die eigentliche Hauptarbeit in diesem Algorithmus beginnt mit der *while*-Schleife ab Zeile 12. Nach dem Entnehmen des ersten Punktes aus der Struktur *minimumPoints* wird der Sichtbereich dieses Punktes berechnet. Dies geschieht mit Hilfe eines Algorithmus, der ausgehend vom aktiven Punkt eine Bresenham-Linie, ähnlich einem Radarstrahl, kreisen lässt und so den Bereich evaluiert, der vom aktiven Punkt aus einsehbar ist. Innerhalb dieser sichtbaren Fläche wird nun in der *for*-Schleife in Zeile 16 die euklidische Distanz berechnet und bei Bedarf aktualisiert. Sind keine *minimumPoints* mehr vorhanden, wird das momentan vorhandene Static Floor Field analysiert und es werden geeignete neue *minimumPoints* ausgewählt, die folgende Kriterien erfüllen.

- Die entsprechende Zelle ist für Agenten begehbar.
- Die Zelle besitzt einen Nachbarn, der für Agenten nicht begehbar ist, oder befindet sich an Rand des Feldes.
- Die momentane Distanz der Zelle ist die kleinste innerhalb des Static Floor Field, die größer ist als der Wert der Variablen *currentMinimum*.

Im Anschluss daran wird in Zeile 11 das erste Objekt der Struktur *minimumPoints* entnommen, falls mindestens eine Zelle gefunden wurde, die oben stehende Kriterien

erfüllt, und deren momentane Distanz als *currentMinimum* gespeichert. Bleibt die Datenstruktur leer, terminiert der Algorithmus und die Initialisierung des Static Floor Field ist beendet.

```
11 currentMinimum = 0
12 while minimumPoints ≠ empty do
13   aktivePoint = minimumPoints.pop()
14   referenceDistance = ssf[aktivePoint.x][aktivePoint.y]
15   visiblePoints = calcVisibleArea(aktivePoint)
16   forall the visiblePoints do
17     currentPoint = visiblePoints.pop()
18     distance = calcEuclideanDistance(aktivePoint, currentPoint)
19     distance = distance + referenceDistance
20     if distance < ssf[currentPoint.x][currentPoint.y] then
21       | ssf[currentPoint.x][currentPoint.y] = distance
22     end
23   end
24   if minimumPoints = empty then
25     | minimumPoints = calcMinimum(ssf, currentMinimum)
26     if minimumPoints ≠ empty then
27       | aktivePoint = minimumPoints.peek()
28       | currentMinimum = ssf[aktivePoint.x][aktivePoint.y]
29     end
30   end
31 end
```

Das Resultat ist nun ein Static Floor Field, das die euklidischen Distanzen zwischen jeder einzelnen Zelle und den Zielpunkt(en) enthält, und das auch mit Hindernissen umgehen kann, was in Abbildung 3.14 anschaulich illustriert wird.

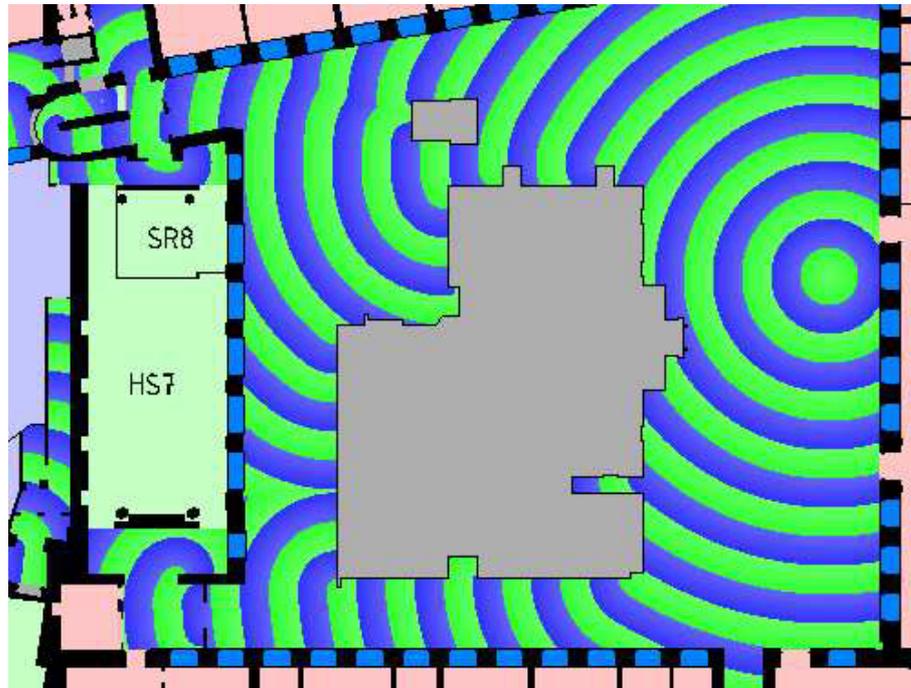


Abbildung 3.14: *Potentiallinien L^2 -Norm*

Orientierung anhand des Static Floor Field:

Anhand des Aufbaus des SFF stellt sich das Auffinden des kürzesten Weges als relativ simpel dar. Würde ein Agent nur eine Zelle besetzen, würde es genügen, die Distanzwerte der acht angrenzenden Nachbarn zu vergleichen und die Richtung mit der geringsten Distanz für den nächsten Schritt heranzuziehen. Da dies aber nicht der Fall ist, ein Standard-Agent benötigt 4×4 Zellen, muss ein Zwischenschritt eingeschoben werden, bei dem das arithmetische Mittel der statischen Werte berechnet wird und anschließend dieser Vergleich vonstattengeht.

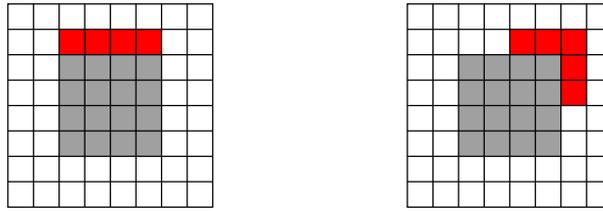


Abbildung 3.15: *Zellen des arithmetischen Mittels*

Das Prinzip für die Berechnung des arithmetischen Mittels wird in Abbildung 3.15 dargestellt. Die grauen Zellen in der Mitte der beiden Bilder stellen einen Agenten dar, während die mit rot markierten Bereiche jene Zellen sind, die für den arithmetischen Durchschnitt herangezogen werden. Der Einfachheit halber werden nur die beiden Richtungen Nord und Nordost dargestellt, da die Berechnung für die restlichen Seiten analog stattfindet.

Nachteil des Static Floor Field:

Ein Nachteil des SFF ist, dass für jedes separat wählbare Ziel ein individuelles generiert werden muss. Dies kann bei einer Simulation, in der ein größeres Gebäude modelliert wurde, den Speicherbedarf schnell auf einige Gigabyte ansteigen lassen, was sich nachteilig auf die Performance und somit auf den reellen Nutzen des Simulators auswirkt. Des Weiteren ist es durch ein SFF nur möglich, den kürzesten Weg zum jeweiligen Zielpunkt zu finden, was sich im Falle eines Staus oder dichten Verkehrs, bei dem nur ein kurzer Umweg genommen werden müsste, um ihn zu umgehen, als sehr unpraktisch herausstellt.

Vorteil des Static Floor Field:

Die Suche nach dem kürzesten Weg gestaltet sich, wie oben gezeigt, sehr einfach. Des Weiteren ist gewährleistet, dass er in jedem Punkt des SFF zu ermitteln ist.

3.5.2.2 Wegpunkte

In diesem Abschnitt wird eine alternative Methode zum Static Floor Field dargestellt und auch deren Vor- und Nachteile kurz dargestellt. Ziel ist es, dabei aufzuzeigen, dass eine Kombination beider Algorithmen eine Verbesserung gegenüber den einzelnen arbeitenden Methoden ist. Basis des alternativen Verfahrens ist die Orientierung an Wegpunkten. In den generierten zellulären Flächen werden einerseits Wegpunkte verteilt, andererseits Wegsegmente definiert, die im Grunde genommen lediglich eine geordnete Menge aus ihnen sind. Die Wegpunkte eines Segmentes werden hierbei so gewählt, dass sichergestellt ist, dass stets eine Sichtverbindung zweier benachbarter Punkte besteht.

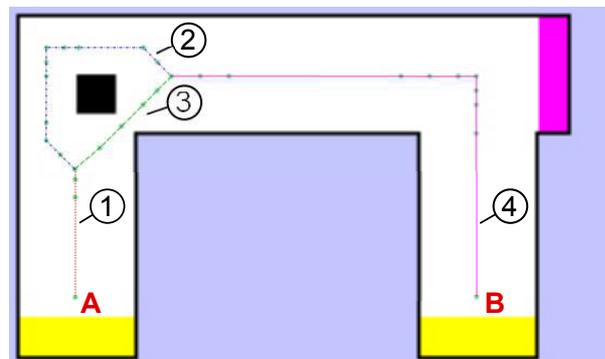


Abbildung 3.16: Darstellung von Wegsegmenten

Abbildung 3.16 soll das Prinzip der Orientierung an Wegpunkten graphisch darstellen. Zu sehen sind die beiden Start- und Stop-Punkte *A* beziehungsweise *B* sowie die Wegsegmente 1–4. Um nun von *A* nach *B* zu gelangen, sind nun die beiden Kombinationen 1, 2, 4 oder 1, 3, 4 aus Wegsegmenten möglich.

Die Wegpunkte und Segmente werden bei der Initialisierung des Projektes aus der Project.csv Datei entnommen und in die Simulation eingebunden.

Orientierung anhand von Wegpunkten:

Auf die Zusammensetzung der Wegsegmente zu einem durchgängigen Pfad wird in der globalen Wegsuche 4.5.1 genauer Bezug genommen. Hier in diesem Abschnitt wird die richtige Anordnung als gegeben vorausgesetzt. Die Orientierung anhand der Wegpunkte gestaltet sich in ihrer Grundform als recht einfach. Der Agent ermittelt anhand seiner Position und der Position des nächsten Wegpunktes den Winkel, den er einschlagen muss, um ihn zu erreichen. Anschließend kontrolliert er, welche seiner acht möglichen Richtungen diesem Wert am nächsten kommt, und erhält somit seinen neuen Kurs. Dieser Ablauf wird so lang durchgeführt, bis der Wegpunkt erreicht wird und ein neues Ziel anvisiert werden kann. Mit dieser Methode durchläuft der Agent den kompletten Weg von Start- bis zum Endpunkt.

Da dieses genaue Folgen des Pfades in der Simulation nicht erstrebenswert erschien, wurde ein zusätzlicher Schritt eingeführt, der es den Agenten ermöglichen soll, Wegpunkte zu überspringen und somit eine Abkürzung zu nehmen. Hierzu wurde vor der Ermittlung des Winkels eine Überprüfung des übernächsten Wegpunktes auf direkte Verbindung eingeschoben. Ist diese Verbindung gegeben, könnte dieser gleich direkt angesteuert werden, ohne Umweg über den momentanen Punkt zu nehmen. Dies erfordert aber noch einer weiteren Überprüfung, um den Fall, der in Abbildung 3.17 dargestellt wird, zu vermeiden.

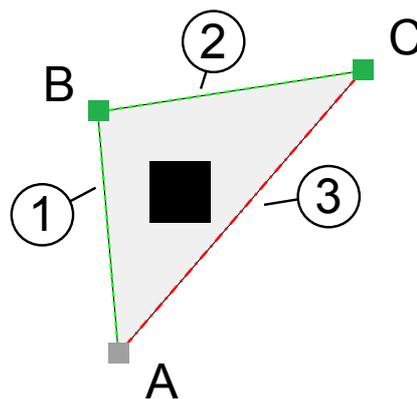


Abbildung 3.17: Fehler beim Überspringen von Wegpunkten

Der Agent (A) hat als Vorgabe den Weg über Punkt (B) und (C). Im Zuge der Überprüfung des übernächsten Wegpunktes stellt er fest, dass er freie Sicht darauf hat (β) und will daher, wie es der Algorithmus vorschreiben würde, (B) ignorieren und sofort übergehen zu (C). Dieses Verhalten würde ihn aber nicht, wie es die globale Wegsuche vorsieht, links am Hindernis vorbeiführen, sondern rechts, was jedoch ein Abweichen vom Weg darstellt. Um dieses fehlerhafte Überspringen zu vermeiden, muss eine zusätzliche Überprüfung des Dreieckes (A, B, C) durchgeführt werden, in dessen Innerem kein Hindernis vorhanden sein darf.

Kann der Wegpunkt nach Überprüfung der beiden oben angeführten Kriterien übersprungen werden, wird dies iterativ fortgesetzt, bis ein Punkt gefunden wird, der als neues Ziel zur Winkelberechnung dient.

Nachteil der Wegpunkte:

Wegpunkte und Segmente müssen, zumindest bis zum momentanen Stand, noch händisch erstellt werden. Abhilfe könnte hierbei ein modifizierter Algorithmus aus der Computergraphik schaffen, der das Skelett eines Bildes berechnet und somit mit einigen weiteren Methoden die Segmente und Wegpunkte herleiten könnte.

Agenten könnten die direkte Sicht auf ihr momentanes Ziel verlieren, zum Beispiel durch das Abgedrängtwerden von anderen Agenten. Ist das der Fall, wäre es ihnen unmöglich, den Winkel zu bestimmen und somit ihre nächste Richtung zu wählen.

Vorteil der Wegpunkte:

Der Algorithmus benötigt keine aufwendige Initialisierung zu Beginn der Simulation. Da der Algorithmus lediglich mit den Wegsegmenten und deren Wegpunkten arbeitet, fällt der Speicherbedarf sehr gering aus.

3.5.2.3 Kombinierte Suche

In den vorhergehenden beiden Abschnitten 3.5.2.1 und 3.5.2.2 wurden zwei Methoden vorgestellt, wie es Agenten möglich ist, sich in zellulären Flächen zurechtzufinden. In diesem Abschnitt wird nun die Kombination der beiden Algorithmen vorgestellt, wie sie auch in diesem Simulator verwendet wird. Ziel war es, die Vorteile beider Methoden zu vereinigen, ohne jedoch deren Schwächen zu übernehmen.

Als Grundlage hierbei wurde die Suche mit Hilfe der Wegpunkte gewählt, da sie mit geringem Speicherbedarf auskommt und ohne aufwendige Initialisierung zu Beginn der Simulation. Der Nachteil, der ihr anhaftet, dass, wenn keine direkte Sicht auf das momentane Ziel besteht, der Agent orientierungslos ist, wird durch ein lokales Static Floor Field kompensiert. Das bedeutet, dass ein Agent jederzeit ein sehr kleines SFF generieren kann, das je nach Abstand zum momentanen Ziel nur weniger 100 Zellen bedarf. Die Initialisierung ist dadurch sehr schnell und speichereffizient. Hierbei kommt der Vorteil des SFF zu tragen, das jederzeit den kürzesten Weg zu seinem Initialisierungspunkt findet. Die Orientierung erfolgt dabei solange mit dieser Methode, bis der Agent wieder direkte Sicht auf sein Ziel hat.

Mit dieser Kombination der beiden Methoden ist es nun Agenten möglich, schnell und effizient ihren Weg durch die einzelnen zellulären Flächen zu finden, ohne dass ihnen dabei die Nachteile der einzelnen Algorithmen anhaften.

3.6 Aufzüge

Seitdem der US-Amerikaner *Elisha Graves Otis* 1853 mit seiner Erfindung des absturzsicheren Aufzugs, dieser Technik zum Durchbruch verhalf, ist er aus der modernen Architektur nicht mehr wegzudenken. So wäre der Großteil der Hochhäuser, die in den letzten 100 Jahre entstanden ohne ihn kaum in ihrer heutigen Form möglich gewesen, da sie bis zu dessen Einführung auf fünf bis sechs Stockwerke beschränkt waren. Dieser Umstand soll natürlich auch hier nicht vernachlässigt werden, und so beschäftigt sich dieser Abschnitt mit den spezifischen Eigenschaften von Aufzügen in Simulationen.

Folgende Punkte sollen hierbei genauer erörtert werden.

- Aufzugssteuerung
- Wartezeiten vor dem Aufzug

3.6.1 Aufzugssteuerung

Die Steuerung der Kabinen stellt in gewisser Weise das Gehirn einer jeden Aufzugsanlage dar. Sie allein bestimmt, wohin sie wann fahren und in welchen Stockwerken sie halten. Um dieser Anforderung gerecht zu werden und das Verlangen nach sicherer und effizienter Beförderung zu stillen, ist sie meist mit aufwendigen Algorithmen und verschiedenen Beförderungsmustern versehen. Diese Muster passen sich in modernen Aufzügen im Laufe des Tages an das Nutzungsverhalten der Passagiere an, um eine möglichst hohe Fördereffizienz der Kabinen zu erreichen. Die geläufigsten Typen sind hierbei

- up-peak traffic
- down-peak traffic
- inter-floor traffic

Die Funktionsweise dieser Muster geht hierbei meist schon aus dem Namen hervor. So ist ein „up-peak“-Verhalten eines Aufzuges darauf ausgerichtet, möglichst effizient Passagiere, von der Lobby oder dem Erdgeschoss ausgehend, in die einzelnen Etagen zu befördern, wie es in Bürogebäuden für gewöhnlich am Morgen stattfindet, während es in Wohnblöcken zu dieser Zeit wohl eher zu einem down-peak kommen wird. Trotz dieser Muster bleibt wohl eine der größten Herausforderungen an jede Aufzugssteuerungen die effiziente Verteilung und Zuordnung der Eingaben, die getätigt werden, besonders wenn es sich um gruppierte Systeme handelt, die auf die unzähligen Kombination der Passagierwünsche eingehen müssen. So stellt sich schnell die Frage, wie auf die Anforderung aus einem Stockwerk reagiert werden soll. Soll eine leere Kabine aus einer entfernteren Etage losgeschickt werden oder eine Kabine, die in Kürze

die Etage passiert, kurz gestoppt werden. Diese und noch mehr Entscheidungen muss die Anlage treffen, um einen wichtigen Aspekt zu gewährleisten, die Minimierung der durchschnittlichen Wartezeit (AWT). Über die Lösung dieses Problems wurden schon unzählige Abhandlungen mit unterschiedlichsten Herangehensweisen veröffentlicht. Sei es mit Hilfe der Fuzzylogik[20], genetischen Algorithmen[30] oder neuronaler Netzwerke, um nur Einige zu nennen. Da die Implementierungen dieser meist recht aufwendigen, Methoden, die mitunter auch mit Trainingsdatensätze trainiert werden müssten, nicht Teil dieser Simulation sein soll, wurde auf eine recht simple Umsetzung dieser Steuerung zurückgegriffen.

3.6.2 Modellierung der Steuerung

Bei der Implementierung der Aufzüge in die Simulation wurde auf das Konzept des Endlichen Zustandsautomaten [27] (EA) zurückgegriffen. Das bedeutet, dass sich jede der Kabinen zu jeder Zeit in einem von sieben genau definierten Zuständen befindet, deren Zusammenhänge in Übergangsmatrizen und Übergangsdiagrammen eindeutig beschrieben wird. Jeder der Zustände führt dabei genau eine, ihm zugeteilte Aktion aus und generiert nach Beendigung seiner Aufgabe, in Abhängigkeit von Zustand und verschiedener Eingaben, eine entsprechende Ausgabe. Da es sich in der Literatur durchgesetzt hat, werden im Folgenden für die eingehenden Signale aus den Stockwerken beziehungsweise der Kabine die englischen Begriffe *floor call* und *car call* verwendet.

3.6.2.1 Mathematisches Modell

Einen endlichen Zustandsautomaten kann man in einer mathematischen Definition als 5-Tupel $(S, s_0, \Sigma, \delta, F)$ ansehen.

- $S \cdots$ eine nicht leere endlich Menge von Zuständen
- $s_0 \cdots$ der Anfangszustand $s_0 \in S$

- $\Sigma \dots$ das Eingabealphabet, eine nicht leere endliche Menge
- $\delta \dots$ Zustandsübergangsfunktion $\delta : S \times \Sigma \rightarrow S$
- $F \dots$ eine, möglicherweise leere, Menge von Endzuständen $F \subseteq S$

Für die Implementierung in unserem Modell ergeben sich daher folgende Mengen und Übergangstabellen beziehungsweise Übergangsdigramme.

$$S = \{\text{wait, move up, move down, open door, close door, agents get in, agents get out}\}$$

$$s_0 = \{\text{wait}\}$$

$$\Sigma = \{\text{destination arrived, opened, closed \& up, closed \& down, closed \& wait, get out finished, get in finished, floor call up, floor call down, floor call this}\}$$

$$F = \{\}$$

Nachfolgend wird die Zustandsübergangsfunktion dargestellt in einer Übergangstabelle und dem entsprechendem Übergangsdigramm. Die Tabelle ist dabei so aufgebaut, dass sich in der obersten Zeile die momentanen Zustände befinden, in der linkensten Spalte alle Elemente des Eingabealphabetes und in den entsprechenden Zellen der Kreuzungspunkte das Ergebnis von $S \times \Sigma \rightarrow S$, also der aus dem alten Zustand und dem Element des Eingabealphabetes resultierende neue Zustand. Ist die Zelle leer, gibt es keinen neuen Zustand, da das Eingabeelement für den momentanen Zustand keine gültige Eingabe darstellt. Der Automat bleibt in dem momentanen Zustand, was aus Gründen der Übersichtlichkeit in der Tabelle nicht explizit ausgeführt wurde.

$\Sigma \backslash S$	wait	move up	move down	open door	close door	agents get in	agents get out
destination arrived		open door	open door				
opened				agents get out			
closed & up					move up		
closed & down					move down		
closed & wait					wait		
get out finished							agents get in
get in finished						close door	
floor call up	move up						
floor call down	move down						
floor call this	open door						

Tabelle 3.3: Zustandsübergangstabelle

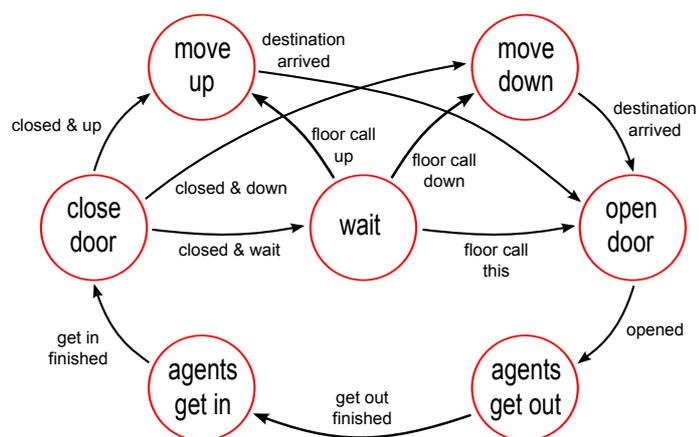


Abbildung 3.18: Zustandsübergangsdigramm

3.6.3 Wartezeiten vor dem Aufzug

Ein relativ wichtiger Aspekt in der Simulation liegt in der Schätzung der Zeit, die man vom Erreichen des Aufzuges bis zu dessen Verlassen einkalkulieren muss. Sie fließt in die Entscheidung über den Verbleib und Warten auf den Lift oder der Suche nach Alternativen wie das Treppenhaus maßgeblich mit ein. Unglücklicherweise hängt diese Zeit von sehr vielen Faktoren ab, auf die wir keinen Einfluss haben und die wir in den meisten Fällen auch nicht kennen. So ist die Personenzahl innerhalb eines Aufzuges von maßgeblicher Bedeutung für die Wartezeit in den einzelnen Etagen, da durch sie eine unbekannte Anzahl von Zwischenstopps ausgelöst werden. Dieser und noch mehr Parameter müssen daher durch statistische Werte ersetzt werden, um zu einem möglichst befriedigenden Ergebnis zu gelangen. Die geschätzte Zeit bis zum Ziel (ETD) setzt sich im Wesentlichen aus acht einzelnen Zeitintervallen zusammen, bei denen einige konstant, andere wiederum flexibel sind.

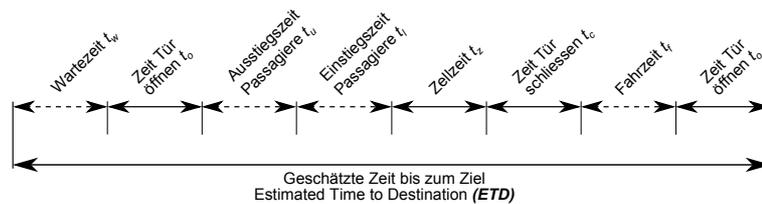


Abbildung 3.19: Wartezeiten für Aufzüge

Abbildung 3.19 illustriert die einzelnen Abschnitte, die in ihrer Summe die ETD ergeben, wobei die durchgezogenen Elemente fixe Zeiten markieren, während unterbrochene Intervalle flexibel sind und deren Dauer noch genauer analysiert wird.

3.6.3.1 Abschätzung der Transportzeit via Aufzug

In diesem Punkt wird auf die Abschätzung der Transportzeit eingegangen, die maßgeblich für die Entscheidung der Agenten ist, ob sie die Treppe oder den Lift nehmen. Um das zu erreichen, wird versucht, vernünftige Erwartungswerte für die vier flexiblen

Zeitintervalle zu ermitteln, die diese Transportzeit entscheidend beeinflussen. Um dies möglichst realitätsnah zu gestalten, werden nur Input Parameter für die Bestimmung herangezogen, die auch realen Personen zur Verfügung stehen, die vor einem Aufzug stehen und intuitiv abschätzen, wie lange sie für die Fahrt benötigen werden. Als Beispiel hierfür seien die Anzahl der momentanen Passagiere und die anstehenden car calls erwähnt, die zwar dem Simulator zur Verfügung stehen würden, aber in den nachfolgenden Berechnungen nur als Erwartungswerte und Annahmen mit einfließen. Die einzigen Informationen, die als gegeben angenommen werden, sind die momentane Position der Kabine und die Fahrtrichtung. Im Folgenden wird eine Methode aufgezeigt, mit deren Hilfe eine sinnvolle Unterteilung aller möglichen Positionen und Richtungen des Aufzugs in Relation zu einem floor call erstellt werden kann, die im weiteren Verlauf hilfreich sein wird.

Drei-Abschnitte-Konzept

Das drei-Abschnitte-Konzept [22, S. 5ff.] beschreibt die Beziehung zwischen dem floor-call und der Bewegungsrichtung bzw. der Position der Aufzugskabine, wobei hier, wie der Name schon verrät, in drei disjunkte Klassen unterteilt wird.

- P1* Bezeichnet jenen Abschnitt, in welche alle floor calls ohne Änderung der Bewegungsrichtung abgearbeitet werden können.
- P2* Bezeichnet jenen Abschnitt, in welche alle floor calls mit einer Änderung der Bewegungsrichtung abgearbeitet werden können.
- P3* Bezeichnet jenen Abschnitt, in welche alle floor calls mit zwei Änderungen der Bewegungsrichtung abgearbeitet werden können.

Abbildung 3.20 stellt das Konzept graphisch dar. Der Einfachheit halber wurde nur die Situation dargestellt, in der sich der Aufzug aufwärts bewegt. Die Abwärtsrichtung funktioniert analog.

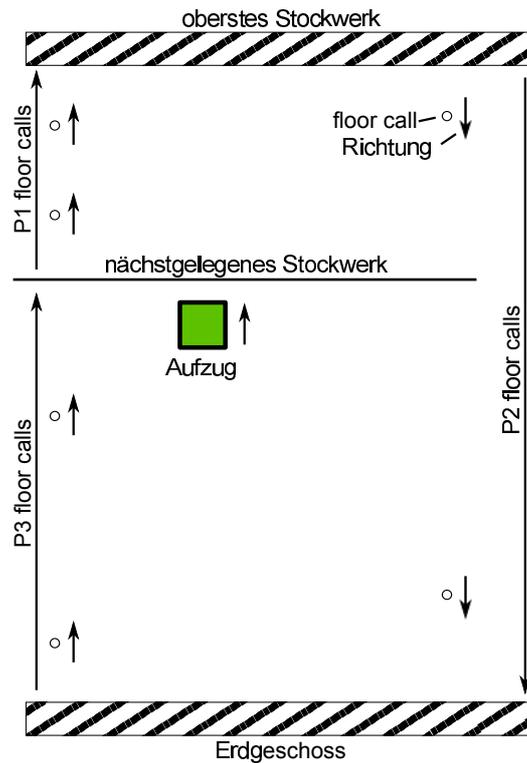


Abbildung 3.20: *Drei-Abschnitte-Konzept*

Für die nun folgenden Berechnungen und Herleitungen werden vorab einige Definitionen eingeführt.

- f_k Anzahl der verbleibenden Stockwerke in der momentanen Fahrtrichtung, wenn sich die Kabine in der k -ten Etage befindet.
- P_k Die Wahrscheinlichkeit, dass keiner der Passagiere in den folgenden Stockwerken aussteigen will.
- d_{fc} Die Richtung des floor calls.
- f_{fc} Das Stockwerk des floor calls.
- f_c Das momentane Stockwerk der Kabine.
- k_E Das erwartete Endstockwerk.
- s_k Die Menge der noch zu erwartenden Stopps.

t_s	Erwartete Zeit für einen Stopp.
t_e	Zeit, die der Fahrstuhl für eine Etage benötigt.
n_c^{exit}	Anzahl der Personen, die pro Stopp aussteigen.
t_u^{pp}	Zeit, die eine Person zum Aussteigen benötigt.
t_i^{pp}	Zeit, die eine Person zum Einsteigen benötigt.
n_{fc}^{pass}	Anzahl der wartenden Personen im Stockwerk f_{fc} .

Wartezeit t_w

Für die Bestimmung der Wartezeit sind einige Fallunterscheidungen zu beachten. Im Idealfall ist die Kabine leer und wartet in einem beliebigen Stockwerk auf den nächsten Auftrag. In diesem Fall wäre die Wartezeit lediglich die Fahrzeit, die der Aufzug benötigt, um die Etage zu wechseln.

$$t_w = |f_c - f_{fc}| \cdot t_e \quad (3.4)$$

Ist das nicht der Fall wird es schon etwas komplizierter, eine sinnvolle Annahme über die zu erwartende Zeit bis zum Eintreffen des Fahrstuhles zu treffen. Nachfolgend wird eine Möglichkeit aufgezeigt, wie solch eine Abschätzung getroffen werden kann.

Anzahl der Passagiere:

Steht die Fahrstuhlkabine nicht wartend in einer Etage, ist davon auszugehen, dass sich Passagiere in ihr befinden, deren Fahrziele der Aufzug gerade im Begriff ist abzu- arbeiten. Da es keine Methode gibt, Aussagen über die genaue Anzahl der Fahrgäste zu treffen, wird im Folgenden davon ausgegangen, dass der Fahrstuhl zu 20 % ausgelastet ist.

$$n_c^{pass} = [0, 2 \cdot CC] \quad (3.5)$$

n_c^{pass} Anzahl der Passagiere im Fahrstuhl.
 CC Kapazität der Kabine (*car capacity*).

Erwartungswert der noch zu fahrenden Etagen: [22, S. 8ff.]

Gleichung 3.6 beschreibt die Wahrscheinlichkeit, dass in den nachfolgenden Stockwerken kein Passagier die Kabine verlassen will, der Aufzug also nicht stoppt. Hierbei und im Folgenden wird stets davon ausgegangen, dass der Fahrstuhl für keinen anderen floor call halten muss. Als Verteilungsfunktion wurde im zweiten Fall ($n_c^{pass} > 1$) eine Exponentialverteilung angenommen.

$$P_k = \begin{cases} 1 - 1/f_k & \text{für } n_c^{pass} = 1 \\ e^{-n_c^{pass}/f_k}, & \text{für } n_c^{pass} > 1 \end{cases} \quad (3.6)$$

Da die Wahrscheinlichkeiten für das „nicht Halten“ nun bekannt sind, kann der Erwartungswert bestimmt werden, wie viele Stockwerke der Aufzug noch zurücklegen wird. Gleichung 3.7 beschreibt diesen Erwartungswert für die beiden Fälle $f_k = 1$, also es existiert nur mehr ein Stockwerk in die jeweilige Richtung, und $f_k > 1$, bei dem noch mehrere Stockwerke zur Verfügung stehen.

$$E(\text{Stockwerke}) = \begin{cases} f_k & \text{für } f_k = 1 \\ f_k - \sum_{l=2}^{f_k} \prod_{j=f_k-l+1}^{f_k} P_k & \text{für } f_k > 1 \end{cases} \quad (3.7)$$

$$k_E = \begin{cases} k + \lceil E(\text{Stockwerke}) \rceil & \text{für } d = \uparrow \\ k - \lceil E(\text{Stockwerke}) \rceil & \text{für } d = \downarrow \end{cases} \quad (3.8)$$

Nach der Berechnung des Erwartungswertes stellt sich nun noch die Frage über die Anzahl der Stopps, die der Fahrstuhl bis zum Erreichen dieses Erwartungswertes einlegen wird. Nachfolgende Gleichung 3.9 beschreibt diesen Zusammenhang der zu erwartenden Zwischenstopps, die Anzahl der zu erwartenden noch zurückzulegenden Stockwerke

und schlussendlich die Wahrscheinlichkeit, in einem Stockwerk nicht anzuhalten.

$$s_k = \lceil E(\text{Stockwerke})(1 - P_k) \rceil \quad (3.9)$$

Als nächster Schritt zur Ermittlung der Wartezeit muss die Fallunterscheidung getroffen werden in welchem Abschnitt des im 3.6.3.1 vorgestellten Konzeptes sich die Kabine in Relation zur Etage des floor calls befindet.

Fall P1.a

Der Aufzug bewegt sich bereits in Richtung des floor calls, der Erwartungswert der zurückzulegenden Stockwerke ist jedoch geringer als die Distanz zwischen momentaner Position des Aufzuges und des Stockwerkes des floor calls. Es treten also voraussichtlich s_k Stopps auf, bevor angenommen werden kann, dass der Fahrstuhl leer ist und den neuen Auftrag annehmen kann.

$$t_w = s_k \cdot t_s + |k_E - f_{fc}| \cdot t_e \quad (3.10)$$

Fall P1.b

Der Aufzug bewegt sich bereits in Richtung des floor calls, der Erwartungswert der zurückzulegenden Stockwerke ist größer als die Distanz zwischen momentaner Position des Aufzuges und des Stockwerkes des floor calls. Es treten also voraussichtlich keine s_k Stopps ein.

$$t_w = t_s \cdot \lceil |f_c - f_{fc}| \cdot (1 - P_k) \rceil \quad (3.11)$$

Fall P2

Der Aufzug bewegt sich vom floor call weg. Das Szenario ist ident mit P1.a, die Kabine absolviert die zu erwartenden s_k Stopps, bevor sie im Stockwerk k_E den neuen floor

call entgegen nimmt.

$$t_w = s_k \cdot t_s + |k_E - f_{fc}| \cdot t_e \quad (3.12)$$

Fall P3

Dieser ist ident mit P2, der Aufzug bewegt sich ebenfalls vom floor call weg und absolviert k_E Stopps, bevor er den neuen floor call entgegennimmt.

$$t_w = s_k \cdot t_s + |k_E - f_{fc}| \cdot t_e \quad (3.13)$$

Ausstiegszeit t_u

Für die Ausstiegszeit t_u ist der oben erwähnte Abschnitt, P1.a – P3, in der sich der Fahrstuhl befinden kann, von entscheidender Bedeutung. Wie bei der Berechnung der Wartezeit gezeigt wurde, ist anzunehmen, dass die Kabine in den Abschnitten P1.a, P2, P3 leer in der Etage f_{fc} ankommt. Im Fall P1.b wird für die Anzahl der Passagiere, die den Fahrstuhl verlassen, angenommen, dass die Zieletagen der Passagiere, die sich in Etage f_c im Fahrstuhl befinden, gleichverteilt auf die zu erwartenden s_k Stopps sind. Es verlassen demzufolge immer gleich viele Passagiere bei jedem Stopp die Kabine.

$$n_c^{exit} = \frac{0,2 \cdot CC}{s_k} \quad (3.14)$$

$$t_u = \begin{cases} 0 & \text{für } P1.a, P2, P3 \\ n_c^{exit} \cdot t_u^{pp} & \text{für } P1.b \end{cases} \quad (3.15)$$

Einstiegszeit t_i

Die Einstiegszeit setzt sich aus dem Produkt der in der Etage f_{fc} wartenden Passagiere n_{fc}^{pass} und der Zeit t_i^{pp} , die für das Einsteigen pro Person veranschlagt ist, zusammen.

$$t_i = n_{fc}^{pass} \cdot t_i^{pp} \quad (3.16)$$

Fahrzeit t_f

Die in Graphik 3.19 dargestellte Fahrzeit t_f setzt sich aus der Fahrzeit und der Zeit zusammen, die für die Stopps benötigt wird. Die vorab getroffene Annahme, dass keine weiteren floor calls anstehen, sei auch hier noch gültig. Aus dieser Annahme folgt des Weiteren, dass die hervorgerufenen Stopps nur durch die zugestiegenen Passagiere hervorgerufen werden. Für die Berechnung der zu erwartenden Stopps können wieder die Gleichungen 3.6 und 3.9 herangezogen werden.

Wie bei der Ermittlung der Wartezeit t_w steht zu Beginn die Anzahl der Passagiere im Vordergrund. Hierzu wird wieder die erwähnte Fallunterscheidung benötigt. Befindet sich der Fahrstuhl in einem der Abschnitte P1.a, P2 oder P3, ist davon auszugehen, dass die Kabine leer ist, wenn sie in der Etage des floor calls f_{fc} hält.

$$n_c^{pass} = \text{Anzahl der wartenden Passagiere} \quad (3.17)$$

Im Abschnitt P1.b unterbricht die Kabine ihre Fahrt zur erwartenden Etage k_E , um die wartenden Passagiere aufzunehmen. Die Anzahl der Passagiere ergibt sich somit aus der Summer der wartenden Personen plus derjenigen, die sich noch in der Kabine befinden.

$$n_c^{rest} = [0,2 \cdot CC - \frac{0,2 \cdot CC}{s_k} |f_c - f_{fc}| (1 - P_k)] \quad (3.18)$$

n_c^{rest} gibt die Anzahl der Passagiere, die sich erwartungsgemäß noch in der Aufzugskabine befinden, wenn der Aufzug in Etage f_{fc} einen Zwischenstopp für den anstehenden floor call einlegt. Die Annahme für die Berechnung dieses Wertes ist, dass die Zieletagen der Passagiere, die sich in Etage f_c im Fahrstuhl befinden, gleichverteilt auf die zu erwartenden s_k Stopps sind. Es verlassen demzufolge immer gleich viele Passagiere bei jedem Stopp die Kabine.

$$n_c^{pass} = \text{Anzahl der wartenden Passagiere} + n_c^{rest} \quad (3.19)$$

Im Anschluss daran kann mit Hilfe der Gleichungen 3.6 bis 3.9 die Anzahl der zu erwartenden Zwischenstopps und somit die zu erwartende Fahrzeit berechnet werden.

Gesamtzeit ETD

Die zu erwartende Zeit bis zum Aussteigen ergibt sich somit aus der Summe der einzelnen, nun bekannten Teilzeiten. Es kann somit eine Abschätzung darüber getroffen werden, inwieweit der Fahrstuhl zeitliche Vorteile gegenüber den Stiegen bietet.

Kapitel 4

Implementierung

4.1 Einführung

In diesem Kapitel wird auf die modelltechnische Umsetzung eingegangen, die, wie bereits im Vorfeld erwähnt, in JAVA ausgeführt wurde. Es werden dabei die bereits in Kapitel 3 erwähnten modellierten Objekte noch einmal aufgegriffen und deren Implementierung beschrieben, die mitunter einige Tücken aufweist. Des Weiteren wird auf die Entstehung von Fehlern und Ungenauigkeiten eingegangen, die bei dieser Art von diskreten Simulation unumgänglich sind, sowie auf deren Minimierung. Die Arbeit fußt dabei zum Teil auf einer meiner Vorrangegangenen Arbeiten. [1]

4.2 Simulationsprojekt

Um die geometrischen Strukturen der Gebäude und deren Inhalte wie Aufzüge und Räume dem Simulator zugänglich zu machen, sind diese Information in verschiedenen Dateien aufgeteilt, welche zu einem Projekt zusammengefasst sind. In diesem Abschnitt wird nun auf die Aufgaben und Inhalte der einzelnen Datei eingegangen und deren Inhalte und Aufbau genauer beschrieben. Nach dem Öffnen des Simulators kann ein Projekt mittels eines Filedialoges ausgewählt und geladen werden.

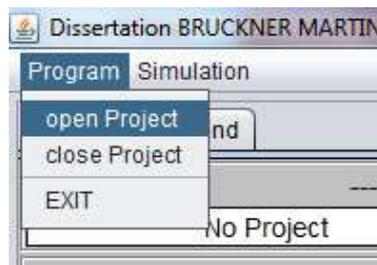


Abbildung 4.1: Screenshot Projekt öffnen

Ein Simulationsprojekt besteht grundsätzlich aus einer beliebigen Anzahl von Dateien, die sich aus drei unterschiedlichen Dateiformaten zusammensetzen:

- der *Project.csv* Datei
- einer oder mehrerer **.png* Dateien
- einer **.txt* Datei

4.2.1 Project.csv

Die Datei *Project.csv* ist das eigentliche Gehirn des Simulationsprojektes. Sie enthält alle Informationen, die der Simulator zum Erstellen und Erkennen der Zusammenhänge benötigt. Der Aufbau dieser Datei ist so ausgelegt, dass jede Zeile genau eine Informationseinheit enthält, die mit einem Schlüsselwort beginnt, gefolgt von Parametern, deren Reihenfolge genau definiert ist. Im Folgenden werden die einzelnen Schlüsselwörter mit ihren zugehörigen Parametern in der Form

- „*Schlüsselwort*“
 - *Parameter 1*
 - *Parameter 2*
 - \vdots

dargestellt. Nach Angabe des Schlüsselwortes wird eine Aufzählung der zugehörigen Parameter und ihrer Bedeutung aufgelistet.

- „*PROJECT*“
 - Name des Projektes.
 - Beliebiger Kommentar.
 - Ein Status, der angibt, ob das Projekt Fehler oder Warnungen enthält.

- „*BUILDING*“
 - Eindeutige ID des Gebäudes.
 - Der zugehörige Gebäudecode.
 - Die Adresse des Gebäudes.

- „*CA*“
 - Eindeutige ID der zellulären Fläche.
 - Die Etage, in der sich die Fläche befindet.
 - Die Gebäude-ID des entsprechenden Gebäudes.
 - Ein Status, der angibt, ob die Fläche Fehler oder Warnungen enthält.

- „*DOOR*“
 - Eindeutige ID der Tür.
 - Die CA-ID der Fläche, in der sich die Tür befindet.
 - Der *RGBA*-Farbcode, in der die Tür als Fläche in der **.png* Datei dargestellt wird.
 - Flag, das die Nutzung als Eingang signalisiert.
 - Flag, das die Nutzung als Ausgang signalisiert.

- „*CHANGE_AREA*“
 - Eindeutige ID des Wechselfeldes.
 - Die CA-ID der Fläche, in der sich das Wechselfeld befindet.
 - Eindeutige ID des verbundenen Wechselfeldes.
 - Der *RGBA*-Farbcode, in der das Wechselfeld in der **.png* Datei dargestellt wird.

- „*ROOM*“
 - Eindeutige ID des Raumes.
 - Die ID des Gebäudes, in der sich der Raum befindet.
 - Die ID der Tür 1, die diesem Raum zugeordnet ist.

 - ⋮
 - Die ID der Tür *x*, die diesem Raum zugeordnet ist.

- „*ELEVATOR_AREA*“
 - Eindeutige ID der Aufzugsfläche.
 - Die CA-ID der Fläche, in der sich die Aufzugsfläche befindet.
 - Der *RGBA*-Farbcode, in der die Aufzugsfläche in der **.png* Datei dargestellt wird.

- „*ELEVATOR_AREA_GROUP*“
 - Eindeutige ID der Gruppe von Aufzugsflächen.
 - Die Etage, in der sich die Gruppe befindet.

- „*ELEVATOR*“
 - Eindeutige ID des Aufzugs.
 - Die ID des Gebäudes, in der sich der Aufzug befindet.
 - Die Zeit, die zum öffnen/schließen der Türen in Sekunden benötigt wird.
 - Die Fahrzeit, die pro Etage in Sekunden benötigt wird.
 - Die maximale Anzahl von Personen.
 - Die ID der Aufzugsfläche 1, die diesem Aufzug zugeordnet ist.
 - ⋮
 - Die ID der Aufzugsfläche x , die diesem Aufzug zugeordnet ist.

- „*ELEVATOR_GROUP*“
 - Eindeutige ID der Gruppe von Aufzügen.
 - Die ID des Aufzugs 1, der dieser Gruppe zugeordnet ist.
 - ⋮
 - Die ID des Aufzugs x , der dieser Gruppe zugeordnet ist.

- „*WAYPOINT*“
 - Eindeutige ID des Wegpunktes.
 - Die ID der Fläche, in der sich der Wegpunkt befindet.
 - Die x-Koordinate des Wegpunktes.
 - Die y-Koordinate des Wegpunktes.

- „*WAYSEGMENT*“
 - Eindeutige ID des Wegsegmentes.
 - Die ID der Fläche, in der sich das Wegsegment befindet.
 - Die ID des Wegpunkts 1, der diesem Wegsegment zugeordnet ist.
 - ⋮
 - Die ID des Wegpunkts x , der diesem Wegsegment zugeordnet ist.

Auf die Aufgaben und Funktionsweisen der eben vorgestellten Objekte wird im Laufe dieses Kapitels noch eingegangen.

Nach Parsen der *Project.csv* Datei und Anlegen beziehungsweise teilweisen Initialisierung der einzelnen Objekte können nun die entsprechenden Bilddateien geladen werden, um die Struktur des Gebäudes in dem Simulator zu komplettieren.

4.2.2 *.png

Diese Dateien enthalten die geometrischen Informationen, die der Simulator benötigt, um das Gebäude virtuell zu erstellen. Sie enthalten den Grundriss des Objektes sowie dessen Stiegen und Korridore. Die Auflösung der Pixelgrafik ist dabei so gewählt, dass ein Pixel in der *.png Datei genau einer Zelle in der diskretisierten Ebene entspricht, also $0,125 \times 0,125$ m. Die Farben, die dabei Verwendung finden, setzen sich zum einen aus fix definierten Grundfarben, Tabelle 4.1, zusammen, zum anderen aus den Farben der jeweiligen Elemente, die in der oben erwähnten *Project.csv* Datei, für etwa *DOORS* oder *CHANGE_AREAs*, spezifiziert worden sind.

Bezeichnung	Farbwert (R)	Farbwert (G)	Farbwert (B)	Begehbar
CORRIDOR	255	255	255	JA
UPSTAIRS	255	232	232	JA
DOWNSTAIRS	232	232	255	JA
WALL	0	0	0	NEIN
OUTDOOR	196	196	255	NEIN
NO PASS	173	173	173	NEIN

Tabelle 4.1: Grundfarben der Bilddateien

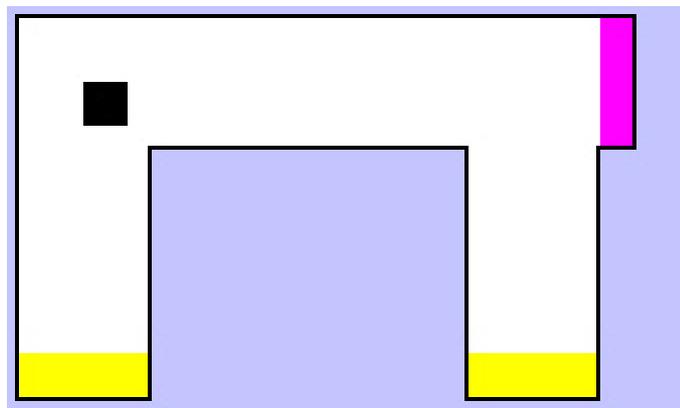


Abbildung 4.2: Beispiel *.png Datei

Abbildung 4.2 illustriert das Beispiel einer *.png Datei, die einen U-förmigen Korridor darstellt, in dessen linker oberen Ecke sich ein Hindernis in Form einer Säule befindet. Des Weiteren sind an den beiden Enden Türen, gelb markiert, vorhanden sowie in der rechten oberen Ecke ein *CHANGE_AREA* für den Übergang zu einer verbundenen zweiten Fläche.

Alphakanal

Die Aufgabe des Alphakanals, der im Normalfall die Transparenz eines Farbwertes angibt, muss hier noch genauer dargelegt werden. Die Grundfarben in Tabelle 4.1 enthalten Werte für *UPSTAIRS* und *DOWNSTAIRS*, die optisch darstellen, ob die jeweilige

Alphakanal	Aufwärtsrichtung
255	N ↑
254	NE ↗
253	E →
252	SE ↘
251	S ↓
250	SW ↙
249	W ←
248	NW ↖

Tabelle 4.2: *Bedeutung des Alphakanals*

Stiege ins obere oder untere Stockwerk führt. Dies ist für den einzelnen Agenten jedoch zu wenig Information, denn ob er nun hinauf-oder hinuntergeht hängt von der Richtung ab, mit der er sich über die Stiege bewegt. Es muss also des Weiteren codiert werden, welche Ausrichtung die Treppe hat. Zu diesem Zweck wurde der Alphakanal verwendet und acht unterschiedliche Werte definiert, die die Orientierung der Treppe enthalten. Die Werte werden in Tabelle 4.2 angegeben, finden jedoch nur Beachtung, wenn der zugehörige Farbcode dem einer Stiege entspricht.

4.2.3 *.txt

Zur Komplettierung des Simulationsprojekt existiert noch eine Textdatei, die alle in der Simulation vorkommenden Agenten spezifiziert. Hierzu beinhaltet die Datei zeilenweise für jeden Agent folgende Daten.

- ID
- Start-Raum
- Ziel-Raum
- Startzeitpunkt

Abbildung 4.3 zeigt den Screenshot einer Beispieldatei, die vier Agenten definiert.

Agent1	ROOM_1	ROOM_2	27.10.2013-15:00:00
Agent2	ROOM_1	ROOM_3	27.10.2013-15:00:00
Agent3	ROOM_2	ROOM_1	27.10.2013-15:01:30
Agent4	ROOM_2	ROOM_1	27.10.2013-15:01:30

Abbildung 4.3: Screenshot Agenten Definition

Da es von Nutzen ist, verschiedenste Szenarien zu simulieren, ist mit Hilfe eines Filedialoges, wie er in Abbildung 4.4 zu sehen ist, möglich, unterschiedliche Agentenspezifikationen zu laden.

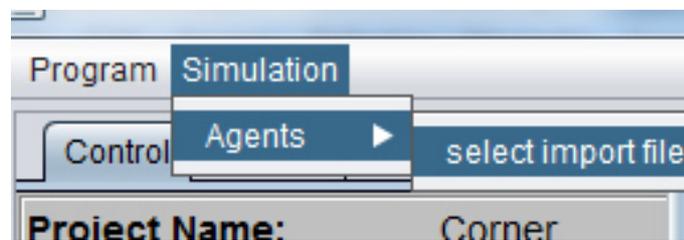


Abbildung 4.4: Screenshot Agenten öffnen

Mit den hier vorgestellten Dateien ist es nun möglich, ein Simulationsprojekt zu erstellen, das alle, für die Simulation notwendigen Daten zu einer Einheit zusammenfasst, beginnend bei der Topologie der Umgebung bis hin zu den zu simulierenden Probanden.

4.3 Umgebung

Im Punkt 3.2 der Modelltechnischen Umsetzung des Rasters wurde bereits auf die Größen der Zelle eingegangen. Wie erwähnt handelt es sich dabei stets um einen Kompromiss zwischen der Genauigkeit bei der Auflösung von Gebäudeteilen und dem verbundenen Speicherbedarf bei der Implementierung der mit der Ordnung $\mathcal{O}(n^2)$ steigt.

Das bedeutet eine Vervielfachung der Zellen bei Verdoppelung der Auflösung. Im Folgenden wird auf nähere Details der Initialisierung und Implementierung der im Kapitel 3 vorgestellten Objekte eingegangen.

4.3.1 Initialisierung

Die Initialisierung beginnt mit dem Laden der Projektdateien, deren Aufgabe und Aufbau in Abschnitt 4.2 ausführlich erörtert wurde. Im Zuge dessen werden die im folgenden aufgezählten Schritte durchgeführt, die schlussendlich zu der gewünschten Umgebung führen, in der sich die Agenten im Laufe der Simulation bewegen und miteinander interagieren können.

1. Die Datei *Project.csv* wird zu Beginn zeilenweise ausgelesen und abgearbeitet. Die Grundgerüste der enthaltenen Elemente, die in Abschnitt 4.2.1 beschrieben worden sind, werden angelegt und in diversen Datenstrukturen gespeichert.
2. Erstellen diverser Verknüpfungen, die die logischen Strukturen der angelegten Elementen darstellen. Beispiele solcher Verknüpfungen sind:

$$\begin{aligned} & \text{DOOR} \rightarrow \text{ROOM} \rightarrow \text{CA} \rightarrow \text{BUILDING} \\ & \text{EVATOR_AREA} \rightarrow \text{ELEVATOR_AREA_GROUP} \rightarrow \text{ELEVATOR} \\ & \qquad \qquad \qquad \vdots \end{aligned}$$

3. Die Dateien **.png* werden geladen und analysiert, hierbei wird das Bild pixelweise durchlaufen und jeder Farbwert muss entsprechend zuordenbar sein. Hierbei gibt es die folgenden beiden Möglichkeiten
 - Der RGBA-Wert ist eine der Grundfarben, die in Tabelle 4.1 definiert wurde. Hierzu zählen Mauern, Gänge und so weiter.
 - Die Farbe ist einem, in der *Project.csv* definierten Element zugewiesen worden.

Durch diese Analyse werden die Positionen der erzeugten Elemente lokalisiert sowie Informationen über die Art der Zelle gespeichert.

4. Überprüfen aller *CHANGE_AREA* Elemente auf ihre Größe und Ausrichtung. Die geometrische Übereinstimmung der Paare ist für die Simulation zwingend erforderlich.
5. Erstellen der Router-Struktur. Siehe dazu Abschnitt 4.5.
6. Erstellen der Voronio-Felder. Siehe dazu Abschnitt 3.5.1.3.

4.3.2 Elemente

Die möglichen Elemente, mit der der Agent in seiner Umgebung in Konflikt geraten kann, sind in der *Project.csv* Datei definiert und wurden im Abschnitt 4.2.1 schon kurz angeschnitten. Nachfolgend soll ein näherer Blick auf die Funktionsweisen der einzelnen Elemente gemacht werden, wobei auf Komponenten wie Mauern und Stiegen nicht eingegangen wird.

DOOR

Dieses Element modelliert die Ein- und Ausstiegspunkte in der Simulation. Ein Agent kann nur in die Simulation eingefügt werden, indem er sie durch ein Element *DOOR* betritt. Abbildung 4.5 illustriert den Ausschnitt einer Simulation, in der zwei *DOOR* Strukturen, mit Pfeilen markiert, zu sehen sind.

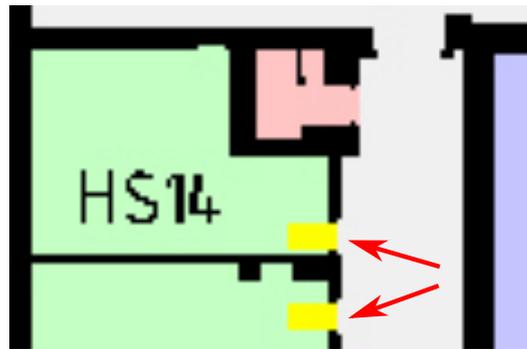


Abbildung 4.5: Screenshot *DOOR*-Element

ROOM

Ein *ROOM* stellt in dem Modell eine Sammlung von *DOOR* und Agenten dar. Wie bereits erwähnt wurde ist, der Zugang zur Simulation einem Agenten nur mit Hilfe eines *DOOR* möglich. Das Einfügen erfolgt aber stets über einen übergeordneten Container, wie in diesem Fall der *ROOM*, der alle einzufügenden Agenten vorher aufnimmt und sie anschließend, je nach Platz und Entfernung zum Ziel, in das geeignetste *DOOR* Element einfügt.

EXIT

Ein weiterer Container zur Aufnahme von *DOOR* Objekten und Agenten ist ein *EXIT*, welches, im Unterschied zu einem *ROOM*, je Gebäude nur einmal vorhanden ist und alle Ausgänge beinhaltet.

CHANGE_AREA

Dieser Bestandteil der Simulation hat bereits in Kapitels 3 Erwähnung gefunden. Er befähigt Agenten, diskretisierte Flächen zu wechseln, und hilft dadurch die Implementierung von Gebäudeplänen in die Simulation zu vereinfachen, da verschiedenste

Gebäudeteile voneinander getrennt diskretisiert werden können. Aus der Anwendung dieser Elemente ist ersichtlich, dass sie immer paarweise vorhanden sein müssen. Des Weiteren ist erforderlich, dass jedes Paar exakt die gleiche Größe und räumliche Ausrichtung hat, da ansonsten der Simulator nicht in der Lage ist, die beiden Flächen an dieser Stelle überlappend zusammenzufügen.

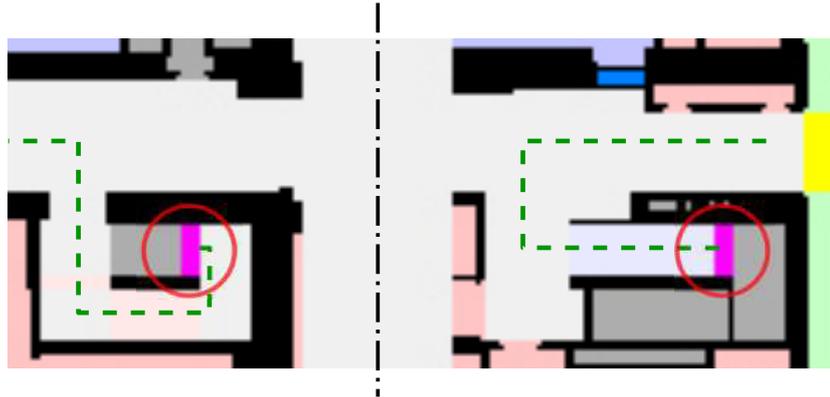


Abbildung 4.6: Screenshot *CHANGE_AREA*-Element

Die Funktionsweise soll in Abbildung 4.6 verdeutlicht werden. Zu sehen sind zwei Etagen, die durch ein *CHANGE_AREA* Paar miteinander verbunden sind. Die grüne gestrichelte Linie soll dabei den Weg darstellen, den ein Agent nehmen muss, um eine Ebene nach oben zu gelangen.

ELEVATOR_AREA

Ein *ELEVATOR_AREA* stellt in der Simulation die Fläche dar, die ein Aufzug in einem Gebäude in jeder Etage einnimmt. Es dient dazu, um Agenten, die einen Lift benutzen, temporär aus der Simulation zu entfernen und sie nach virtueller Fahrzeit in einem anderen *ELEVATOR_AREA* wieder einzufügen. Abbildung 4.7 illustriert zwei dieser speziellen Flächen, die durch blaue Flächen dargestellt werden. Über den genauen Ablauf der Aufzugssimulation wurde bereits in Abschnitt 3.6 berichtet.

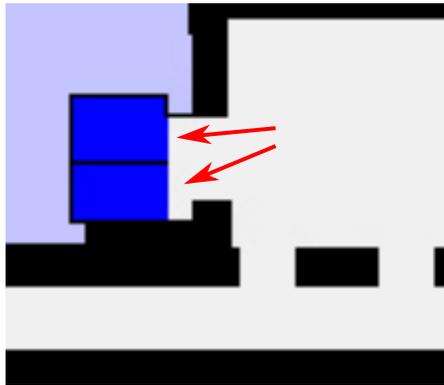


Abbildung 4.7: Screenshot *ELEVATOR_AREA*-Element

ELEVATOR_GROUP

In stark frequentierten Bereichen ist es üblich, dass mehrere Aufzüge nebeneinander errichtet und gruppiert werden, um die Wartezeit der Fahrgäste zu minimieren. Hierbei ist die Steuerung so ausgelegt, dass es nicht dem Fahrgast obliegt, welcher Fahrstuhl für die Beförderung ausgewählt wird, sondern die Steuerung selbst diese Auswahl übernimmt. Um dies umzusetzen, existieren auch, wie in Abbildung 4.7 zu sehen ist, nebeneinanderliegende *ELEVATOR_AREA*, die zu einer *ELEVATOR_AREA_GROUP* zusammengefasst sind und im Verbund agieren.

ELEVATOR

Mit der nach Stockwerken geordneten Zusammenfassung von *ELEVATOR_AREA* zu einer Gruppe wird in dem Simulator ein Fahrstuhl modelliert. Mit Hilfe dieser Gruppierung, einigen parametrisierbaren Zeiten und der maximalen Kapazität der Aufzugskabine ist es möglich, Agenten zwischen den einzelnen Ebenen gezielt zu transportieren.

ELEVATOR_GROUP

Ähnlich wie *ELEVATOR_AREA_GROUP* müssen auch die entsprechenden *ELEVATOR* zu einem Verbund gekoppelt werden, um die gewünschte Gruppierung der Aufzüge zu erhalten.

4.4 Agent

Auf die Eigenschaften von Agenten und deren Parameter wurde bereits im Kapitel der Modellspezifikation eingegangen. Im Folgenden wird deren Initialisierung und die Verteilungsfunktionen erörtert, die ihnen zugrundeliegen.

4.4.1 Initialisierung

Die Initialisierung eines Agenten erfolgt einerseits durch stochastische Prozesse, andererseits durch Abhängigkeiten diverser Parameter, auf die in Abschnitt 3.4 bereits näher eingegangen wurde. Die drei Eigenschaften, die hierbei durch einen Zufallsprozess ermittelt werden, sind

- Geschlecht
- Rollstuhlfahrer
- Alter

Für die Verteilung der beiden ersten Werte wurde eine Gleichverteilung verwendet, mit deren Hilfe eine Zufallsvariable erzeugt wurde, die in Kombination mit der Einstellung am Panel für die Wahrscheinlichkeiten, Abbildung 4.8, die Initialisierungswerte für das jeweilige Geschlecht des Agenten sowie die Abhängigkeit von einem Rollstuhl erzeugen.

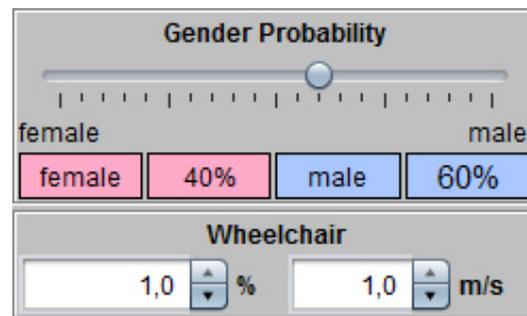


Abbildung 4.8: Screenshot Wahrscheinlichkeits-Panel

Der Initialisierungswert des Alters wird mit Hilfe einer Normalverteilung 4.1 bestimmt, wobei die generierten Lebensjahre nach oben und unten beschränkt sind, um mögliche extreme Ausreißer abzufangen. Die somit möglichen Werte sind auf das Intervall $[20, 40]$ begrenzt.

$$\text{Altersverteilung} = \mathcal{N}(30, 10) \quad (4.1)$$

Die nachfolgende Abbildung 4.9 zeigt das Panel der Altersverteilung und die Einstellungsmöglichkeit für Varianz und Erwartungswert sowie die beiden Werte für das minimale und maximale Alter.

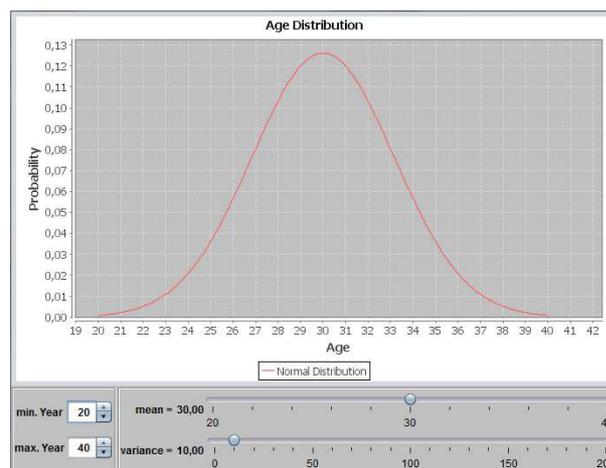


Abbildung 4.9: Screenshot Wahrscheinlichkeitsverteilung des Alters

Nachdem die stochastischen Werte für den Agenten gesetzt worden sind, können die drei Parameterwerte, die die individuelle Geschwindigkeit bestimmen, ermittelt werden. Die Details über die Zusammenhänge und die Methoden, die hierbei Verwendung finden, wurden in Abschnitt 3.4 behandelt.

4.5 Routing

Auf die Funktionsweise der beiden Routingprozesse wurde in der technischen Modellbeschreibung bereits eingegangen. Im folgenden Abschnitt wird die Implementierung der globalen Wegsuche, mit ihrem speziellen Routing-Graphen, vermittelt.

4.5.1 Initialisierung Modell Graph

Die Initialisierung des Graphen besteht grundsätzlich aus zwei Schritten, die nacheinander durchgeführt werden, um ihn für den Einsatz des Dijkstra-Algorithmus vorzubereiten.

1. Erstellen eines Knotens für jedes einzelne Element.
2. Verknüpfen der Knoten mittels Kanten, sodass sie die Topologie des Gebäudes widerspiegeln.

Erstellen der Knoten

Hierbei wird für jedes in der Simulation vorkommende Elemente aus nachfolgender Liste ein separater Knoten generiert und gespeichert.

- *DOOR*
- *CHANGE_AREA*

- *ELEVATOR_AREA*
- *WAYPOINT*

Erstellen der Kanten

Die Erstellung der Kanten beginnt mit der Generierung der Verbindungen der paarweise auftretenden *CHANGE_AREAs*. Sie werden mit dem Kantengewicht 0 generiert, was bedeutet, dass die Zeitdauer eines Agent, für den Wechsel 0 Sekunden beträgt. Dieser Wert wird in weiterer Folge auch nicht mehr aktualisiert.

Als Nächstes folgt die Suche nach der Anbindungen der Knoten, die für die Elemente *DOOR*, *CHANGE_AREA* und *ELEVATOR_AREA* generiert worden sind an das *Backbone* der Wegsuche, das seinerseits von den Elementen des Typs *WAYPOINT* gebildet wird. Diese verzweigte Struktur durchzieht sämtliche Korridore und wird daher von den einzelnen Agenten bei der Navigation genutzt. Mehr dazu siehe Abschnitt 3.5.2.2. Zur Anbindung der einzelnen Elemente wird hierzu nun der *WAYPOINT* mit der kürzesten Distanz in derselben Fläche gesucht und mit dem jeweiligen Element mittels Kante verbunden.

Anzumerken ist hierbei, dass in jedem Knoten jeweils zwei Sammlungen von Kanten existieren. Einer für jeweils Agenten mit und ohne Mobilitätseinschränkung. Der Unterschied zwischen den Kanten ist, dass für Agenten mit eingeschränkter Mobilität nur Kanten existieren, deren Wege keine Stiegen enthalten. Die Anbindung der oben erwähnten Elemente wird also zweimal durchgeführt, wobei das Meiden von Stiegen jeweils einmal berücksichtigt wird. Mathematisch bedeutet das, dass die beiden Graphen $G_1 = (V_1, E_1)$ für Agenten ohne Handicap und $G_2 = (V_2, E_2)$ für Agenten mit Handicap die Eigenschaften besitzen $V_1 = V_2$ und $E_1 \supseteq E_2$.

Im Anschluss daran, wird das *Backbone* generiert. Hierzu werden alle Elemente des Typs *WAYPOINT* in der Reihenfolge, die in der *Project.csv* Datei angegeben wurde, durchlaufen. Dabei wird die Distanz zwischen den benachbarten Punkten ermittelt sowie das eventuelle Vorhandensein von Stiegen. Die Zeit, die zum Bewältigen des

Weges benötigt wird, wird erst später, wie oben erwähnt, dynamisch ermittelt. Die Werte werden der Kante zugeführt und in den Knoten gespeichert, wobei hier wiederum Rücksicht genommen wird, ob die Kante auch für den Graphen G_2 geeignet ist.

Zum Abschluss werden die *ELEVATOR_AREAs* pro Fahrstuhl mittels Kanten etagenweise verbunden. Das Kantengewicht wird dabei erst später beim dynamischen Routingvorgang laut Vorschrift in Abschnitt 3.6.3 gesetzt.

4.6 Aufzüge

4.6.1 Graphische Darstellung

Alle Aufzüge manifestieren sich in der Simulation ausschließlich durch ihre sogenannten *ELEVATOR_AREA* in den einzelnen Stockwerken. Dies sind die Flächen, aus denen die Agenten aus der Simulation entfernt werden um sich in den virtuellen Aufzug zu begeben oder, wie beim Verlassen des Aufzuges, wieder in die Simulation eingefügt zu werden. Die Beziehung, in der die einzelnen Flächen zueinander stehen, wird im Konfigurationsfile *Project.csv* genau spezifiziert. So wird jeder Fläche ein Aufzug zugeordnet sowie ihre Position, die sie in der Stockwerkshierarchie einnimmt.

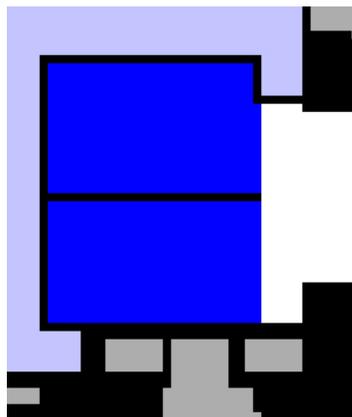


Abbildung 4.10: Darstellung *ELEVATOR_AREA*

Der in Abbildung 4.10 dargestellte Screenshot illustriert zwei *ELEVATOR_AREAs*, blau dargestellt, die zu unterschiedlichen Aufzügen gehören, sich jedoch in derselben Aufzugsgruppe befinden. Nähere Details lassen sich im Laufe der Simulation durch einen Mausklick auf die Fläche abrufen. Daten wie Anzahl der Passagiere, Fahrtrichtung, Stockwerk und so weiter werden dabei in einem Informationsfeld dargestellt.



ELEVATOR PANEL	
Elevator Group:	ELEVATOR_GROUP_1
Elevator:	ELEVATOR_1
Passengers 3/10	
30%	
Door:	OPENED
Direction:	DOWN
Upper Floor:	NO ELEVATOR AREA
Current Floor:	2
Lower Floor:	1
Time Door:	2.0s
Time Floor:	3.0s
current Job:	GET IN PASSENGERS
Students UP:	0
Students DOWN:	1

Abbildung 4.11: Informationsfeld für Aufzüge

Abbildung 4.11 stellt das erwähnte Informationsfeld der *ELEVATOR_AREA* dar und beinhaltet, beginnend von oben, folgende Daten:

- Identität der Aufzugsgruppe (*ELEVATOR_GROUP_1*)
- Identität des Aufzugs (*ELEVATOR_1*)
- Momentane Auslastung des Aufzugs (*30 %*)
- Status der Türen (*OPENED*)
- Momentane Fahrtrichtung (*DOWN*)
- Oberes Stockwerk (*NO ELEVATOR AREA*) \Rightarrow Der Fahrstuhl befindet sich bereits im obersten Stockwerk.

- Momentanes Stockwerk (2)
- Unteres Stockwerk (1)
- Zeit zum Öffnen/Schliessen der Türen (2 s)
- Zeit pro Stockwerk (3 s)
- Momentaner Status der Steuerung (*GET IN PASSENGERS*)
- Wartende Agenten Richtung aufwärts (0)
- Wartende Agenten Richtung abwärts (1)

4.6.2 Funktionsweise der implementierten Aufzugssteuerung

Da der Implementierungsaufwand, wie bereits eingangs erwähnt, für gruppierte Anlagen recht hoch sein kann, wurde, um den Aufwand für diesen relativ kleinen Bereich der Arbeit in Grenzen zu halten, auf eine aufwendige Modellierung verzichtet und eine entsprechende Methode vorgesehen, die sicherlich nicht in der Lage ist, die Wartezeiten zu minimieren, aber für unsere Zwecke mehr als ausreichend ist.

Für die Umsetzung dieser Steuerung wurden Algorithmen entwickelt, die nur auf wenigen Regeln aufbauen, sodass sie leicht verständlich und nachvollziehbar sind und dennoch effizient genug, um die Aufgabe problemlos und in adäquater Form zu bewältigen. Des Weiteren ist durch den Aufbau über Regeln gewährleistet, dass sich die Logik, die die Liftkabinen steuert, einfach erweitern lässt, um die Effizienz gegebenenfalls zu erhöhen. Diese fundamentalen Regeln werden im Folgenden nun kurz erläutert.

Abarbeitung von floor calls

- Alle eingehenden floor calls werden in einer Queue gesammelt.
- Die Queue wird sequenziell, nach dem First-In-First-Out-Prinzip, abgearbeitet.
- Existiert ein floor calls in der Queue, wird er an den nächstgelegenen leeren Aufzug übergeben.

- Steht kein Aufzug zur Verfügung, stoppt der nächste, der die Etage des floor calls passiert.
- Der Aufzug ändert nur seine Richtung, wenn er leer ist und sich im Wartezustand befindet.

Mit diesen fünf recht simplen Regeln lassen sich die Kabinen eines Aufzugverbundes zu den einzelnen anstehenden floor calls dirigieren und es ist des Weiteren gewährleistet, dass alle calls über kurz oder lang abgearbeitet werden.

Abarbeitung von car calls

Im Gegensatz zu den floor calls ist das Handling der car calls recht simple. Alle anstehenden und eingehenden calls werde aufsteigend beziehungsweise absteigend, in Abhängigkeit der Fahrrichtung, sortiert und nacheinander abgearbeitet. Ist keine car call mehr vorhanden, meldet sich die Kabine als frei bei der Aufzugssteuerung und kann für etwaige anstehende oder eingehende floor calls genutzt werden.

4.6.3 Programmierparadigma

Die Aufzugssteuerung ist als Event basiertes Element im Modell implementiert worden. Die grundlegenden Bausteine hierbei sind eine Event-Queue, die alle die Aufzüge betreffenden Events selbst, die in ihr gespeichert werden, der Zeit nach aufsteigend sortiert, und die Events. Ein Beispiel für ein Event wäre, dass ein Aufzug ein Stockwerk erreicht hat. Die Abarbeitung erfolgt nun in der Form, dass anhand der momentanen Modellzeit alle Events evaluiert werden, die zur Zeit anstehen und bearbeitet werden müssen. Für obiges Beispiel würde das etwa bedeuten, die Anzeige, die den Status des Aufzugs visualisiert, zu aktualisieren und, falls das Zielstockwerk noch nicht erreicht worden ist, neuerlich einen Event zu generieren, der beim Erreichen der nächsten Etage aufgerufen wird.

4.7 Diskretisierungsfehler

Da es sich bei dem agentenbasierten System um ein zeit-und ortdiskretes Modell handelt, ergeben sich hierbei unweigerlich Ungenauigkeiten, die hier näher beleuchtet werden und deren Minimierung zur Qualität und Aussagekraft der Simulation beiträgt. Diese Fehler entstehen dadurch, dass sich die Agenten auf einer Gitterstruktur bewegen und sich somit nur an festen, durch die räumliche Auflösung der Struktur vordefinierten, vorgegebenen Positionen befinden können. Ihre Bewegungsfreiheit wird dadurch stark eingeschränkt und es kommt zu Abweichungen, die je nach Richtung und eingestelltem Zeitintervall Δt variiert. Grundsätzlich entstehen bei der Implementierung und Umsetzung des Modells zwei Fehler.

4.7.1 Fehler durch Zeitintervall Δt

Bei der Implementierung des Modells werden für jeden Agenten die Trajektorien-Schritte für die Dauer eines Zeitintervalls vorab berechnet. Es wird hierbei genau die Anzahl n von Zellwechseln ermittelt, sodass der Betrag aus der Differenz der Summen der Zeiten, die für die Wechsel benötigt werden, und der Intervalldauer minimal wird.

t_i	Benötigte Zeit für den i -ten Zellwechsel im aktuellen Zeitintervall
n	Anzahl der Zellwechsel im aktuellen Zeitintervall

$$n \in N : \left| \left(\sum_{i=1}^n t_i \right) - \Delta t \right| = \text{minimal} \quad (4.2)$$

Der Fehler entsteht nun dadurch, dass n so gewählt wird, dass die Differenz minimal wird, sie aber in den seltensten Fällen 0 ergibt. So nutzt der Agent seine ihm zur Verfügung stehende Zeit nicht vollständig aus oder er überzieht sie, sodass er mehr Zeit „konsumiert“, als ihm eigentlich zur Verfügung steht. Dies drückt sich in der Zeit

$$t_f = \left(\sum_{i=1}^n t_i \right) - \Delta t \quad (4.3)$$

aus, die je nach Vorzeichen

- $t_f > 0$: Zeitschuld
- $t_f < 0$: Zeitguthaben

bedeutet. Die dadurch entstehende Ungenauigkeit soll anhand des nachfolgenden Beispiels verdeutlicht werden. Die Werte für die Geschwindigkeit und das Zeitintervall entsprechen denen, die in der Simulation Verwendung finden.

4.7.1.1 Fehlerberechnung anhand eines Beispiels

v	Geschwindigkeit des Agenten = 1,34 m/s
Δt	Zu berechnendes Zeitintervall = 0,25 s

Daraus folgt eine zurückgelegte Distanz bei freier Strecke pro Zeitintervall von

$$s_i = 1,34m/s \cdot 0,25s = 0,335m$$

Mögliche diskrete Distanzen sind $2/8 = 0,250$ m oder $3/8 = 0,375$ m. Wir entscheiden uns für den näherliegenden größeren Wert, um den entstandenen Fehler zu minimieren. Hiermit ergibt sich ein absoluter Fehler von

$$f_{da} = 0,375m - 0,335m = 0,04m$$

was einem relativen Fehler von

$$f_{d_r} = (0,04m)/(0,335m) \cdot 100\% \approx 12\%$$

entspricht. Wie anhand dieses Beispiels nachzuvollziehen ist, beeinflusst der so entstehende Fehler nicht unerheblich das Simulationsergebnis, und was noch erschwerend hinzukommt, ist, dass er sehr stark abhängig vom gewählten Zeitintervall Δt ist. Da diese Abweichung in unserer Simulation die ermittelten Werte zu sehr verfälschen, wurde eine Kompensation dieses Fehlers notwendig.

4.7.1.2 Kompensation des Δt Fehlers

Der Ablauf jedes Zellenwechsels besteht in groben Zügen aus der Ermittlung der folgenden Punkten.

- Der Richtung des nächsten Zellwechsels.
- Die sich aus der Richtung des Wechsels ergebende Geschwindigkeit.
- Die Zeit, die für den Wechsel in die nächste Zelle benötigt wird.

Dieser Ablauf wird so oft wiederholt, bis der oben erwähnte Fehler t_f minimal wird. Ist das der Fall, wird er abgespeichert, um beim nächsten Durchlauf zu Δt addiert, was zur Folge hat, dass dem Agenten nun mehr oder weniger Zeit in diesem Zeitintervall zur Verfügung steht und so der Fehler, der durch die Diskretisierung entsteht, kompensiert wird.

4.7.2 Fehler durch Rasterung des Raumes

Dem hier zu erörternden Problem liegt die Tatsache zugrunde, dass den Agenten nicht alle Richtungen zur Verfügung stehen, die es in einem kontinuierlichen System gibt. Sie können nur aus einer beschränkten Auswahl wählen, die ihnen durch die Nachbarschaftsdefinition des agentenbasierten Modells zur Verfügung gestellt werden. Die bei der Implementierung verwendete Moore-Nachbarschaft stellt hierbei dem Agenten acht

verschiedene Richtungen zur Verfügung, aus der er wählen kann. Diese eingeschränkte Richtungswahl hat nun zur Folge, dass die Agenten, wenn sich ihr Ziel nicht direkt an den acht direkten Routen befindet, durch das Zick-Zack gehen und Umwege in Kauf nehmen müssen.

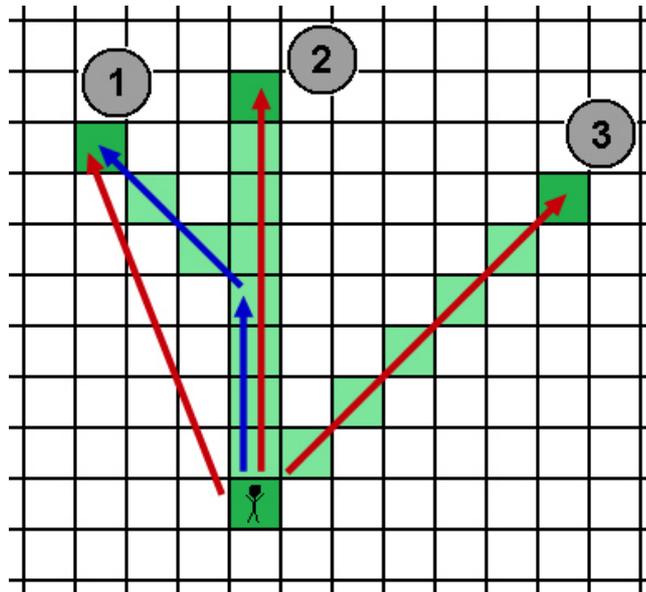


Abbildung 4.12: Ursache des Rasterfehlers

Abbildung 4.12 visualisiert den Diskretisierungsfehler aufgrund der beschränkten Auswahl der Richtungen. Wie zu erkennen ist liegen die Punkte 2 beziehungsweise 3 in jeweils einer der Verlängerungen der acht möglichen Richtungen, die dem Agenten zur Verfügung stehen, was ihm ermöglicht, auf direkten Weg zu seinem Ziel zu gelangen. So ist der Punkt 2 beispielsweise bei einer Auflösung von $0,125$ m pro Zelle 2 m vom Agenten entfernt. Dieser muss, um ihn zu erreichen, acht Züge Richtung Norden vollziehen, was einer Strecke von $8 \times 0,125$ m also 2 m entspricht. Ebenso verhält es sich beim Punkt 3.

Anders stellt sich die Lage bei Punkt 1 dar, der vom Agenten $\approx 0,9519$ m entfernt ist, er jedoch als kürzesten Weg vier gerade und drei diagonale Wechsel vollziehen muss, was einer Entfernung von $\approx 1,030$ m entspricht, was einem relativen Fehler von $\approx 8,2$ % entspricht.

Die in Abbildung 4.12 dargestellten Schritte, die der Agent nimmt, um Punkt 1 zu erreichen, sollen hier nur als Referenz für die Wege dienen, die aus vier Geraden und drei diagonalen Zellwechsel kombiniert werden können. Abbildung 4.13 zeigt einen alternativen Weg, der sich der direkten Verbindung annähert, was aber an der Tatsache, dass die kürzeste zurückzulegende Distanz 1,030 m beträgt, nichts ändert.

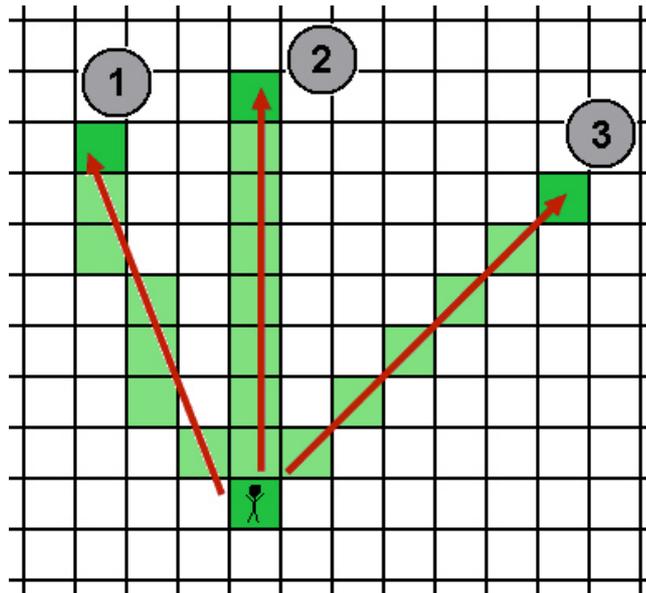


Abbildung 4.13: *Alternative Route*

4.7.2.1 Fehlerberechnung

Um zu verdeutlichen, dass der im vorherigen Abschnitt erläuterte Fehler stark winkelabhängig ist, wollen wir uns zunächst mit der Darstellung eines Kreises auf einer diskreten Fläche befassen.

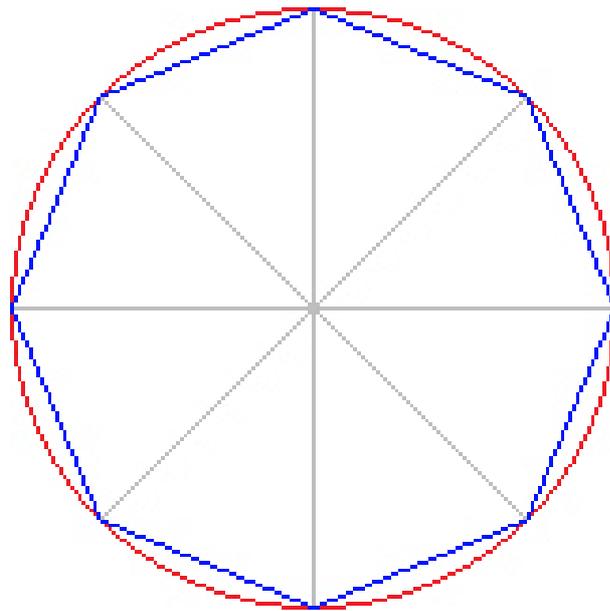


Abbildung 4.14: *Diskrete Kreise*

Der in Abbildung 4.14 dargestellte äußere Ring mit Radius R stellt die bestmögliche Annäherung in einem diskreten Raum an einen perfekten Kreis dar. Zur Rasterung oder Diskretisierung wird hierzu in Computergrafik-Programmen meist auf einen Bresenham-Algorithmus zurückgegriffen wie er in unzähligen Einführungsbüchern in diese Thematik umfangreich erörtert wird. Dem gegenüber stellt das innere Achteck in der Abbildung einen „Kreis“, ebenfalls mit Radius R , in dem von uns eingesetzten agentenbasierten System dar. Gut zu erkennen ist hierbei, dass die Radien beider Kreise an den vier um 45° verdrehten Achsen übereinstimmen. Entlang dieser Richtungen ist es den Agenten, aufgrund der in unserem System gewählten Moore-Nachbarschaft, möglich, den kürzest möglichen Weg zu nehmen. In allen anderen Richtungen hingegen kommt es durch das im vorherigen Abschnitt erwähnte erzwungene Zick-Zack-Gehen zu einem Umweg, sodass nach einer vom Agenten zurückgelegten Distanz von R Metern erst die innere blaue Linie erreicht wird.

Um die Größenordnung dieses Fehlers zu ermitteln und einen Lösungsansatz aufzuzeigen wurde mit Hilfe der in Abbildung 4.14 dargestellten Kreise die relativen Fehler, von $0 \dots 45^\circ$, einer Stichprobe ermittelt. Dazu wurden gerasterte Linien vom Zentrum zu den Pixeln des äußeren Kreises im entsprechenden Intervall gezogen und die Gesamtlänge mit derjenigen bis zum Schnittpunkt des inneren Kreises in Relation gesetzt.

Die dabei gesammelten Daten wurden mit einem Fehlerpolynom p_f zweiten Grades approximiert und in Abbildung 4.15 dargestellt. Der Definitionsbereich von $\pi/4$ oder eben 45° ist hierbei völlig ausreichend, da sich der relative Fehler periodisch wiederholt.

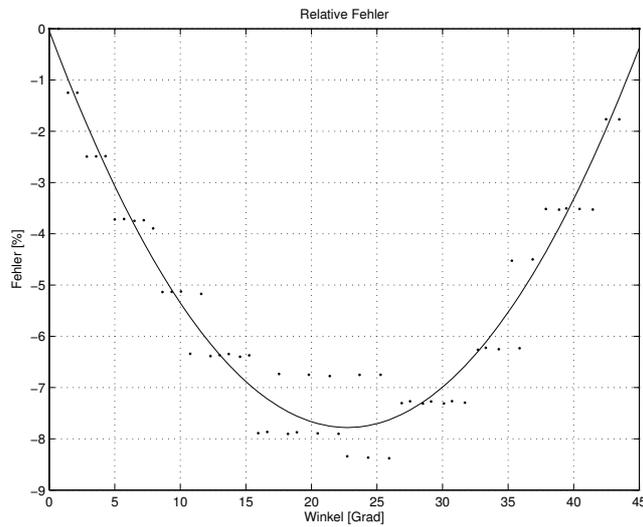


Abbildung 4.15: *Relativer Fehler*

Mit dem in Abbildung 4.15 approximierenden Fehlerpolynom p_f

$$p_f = 0,015x^2 - 0,6806x - 0,0405 \quad (4.4)$$

kann jetzt der relative Fehler genauer betrachtet werden und eine Methode für dessen Kompensation aufgezeigt werden.

4.7.2.2 Kompensation des Fehlers durch Rasterung des Raumes

Grundgedanke bei dieser Methode ist die Annahme, dass die Richtungen, in der sich die Agenten bewegen möchten, gleichverteilt von $0 \dots 360^\circ$ sind. Der durchschnittliche Fehler, der hierbei entsteht, wird wie folgt berechnet.

$$\bar{p}_f = \frac{1}{b-a} \int_a^b p_f dx \quad (4.5)$$

Durch Einsetzen der beiden Intervallgrenzen von 0° beziehungsweise 45° in Gleichung (4.5) ergibt sich nun ein durchschnittlicher relativer Fehler von

$$\bar{p}_f = \frac{1}{45} \int_0^{45} (0,015x^2 - 0,6806x - 0,0405) dx = -5,229\% \quad (4.6)$$

Das bedeutet, dass jeder Agent durchschnittlich nur 94,771 % des Weges zurücklegt, den er in einer kontinuierlichen Welt in der vorgegebenen Zeit zurücklegen könnte. Um dies nun zu kompensieren, ist es das Naheliegendste, einen Multiplikationsfaktor einzuführen, mit Hilfe dessen dieser Fehler so gering als möglich gehalten wird.

$$k_f = \frac{100\%}{94,771\%} \approx 1,0551 \quad (4.7)$$

$$\Delta t_k = \Delta t \cdot k_f \quad (4.8)$$

Der so ermittelte Faktor k_f wird schlussendlich mit dem Intervall Δt multipliziert, um den Agenten eine etwa 5,5 % längere Zeitspanne zu geben, in der er Aktionen und Interaktionen ausführen kann.

Kapitel 5

Simulationsablauf

5.1 Update-Prozess

In diesem Kapitel wird auf den Ablauf der Simulation eingegangen. Während in Kapitel 3 auf den grundlegenden Aufbau des Modells sowie in Kapitel 4 auf die Implementierung eingegangen wurde, soll hier ein Verständnis dafür vermittelt werden, wie und warum sich Agenten in der Simulation verhalten.

Die eigentliche Arbeit des Simulators wird in einer Update-Schleife durchgeführt, die die Bewegung aller Agenten für die nächsten $\Delta T = 250$ ms ermittelt. Diese Methode wird, je nach Einstellung, siehe Abbildung 5.1, exakt alle ΔT aufgerufen, sodass die Simulation in Echtzeit zu verfolgen ist, oder in einer Endlosschleife, in der ein Aufruf dem nächsten folgt, sodass der Prozess mit maximaler Geschwindigkeit abläuft. Innerhalb der Methode werden nachfolgende Punkte Schritt für Schritt ausgeführt.

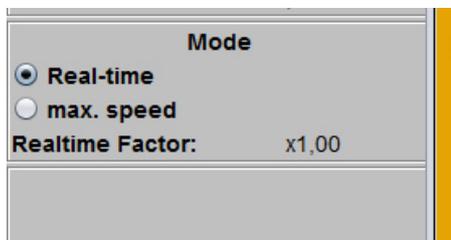


Abbildung 5.1: Screenshot Simulationsmodus

- Abarbeiten der Aufzugs-Events.
- Einfügen neuer Agenten.
- Erstellen der Trajektorien der einzelnen Agenten.
- Bewegen der Agenten.
- Visualisieren der ausgewählten Umgebung.

5.1.1 Aufzüge

Die Aufzugssteuerung ist, wie in Abschnitt 4.6.3 beschrieben, als eventsbasierte Steuerung implementiert worden. In diesem Punkt wird die Event-Queue, die alle, die Aufzüge betreffenden Events beinhaltet aktualisiert. Das bedeutet, dass alle im momentanen Zeitpunkt anstehenden Events erledigt und aus der Queue entfernt werden. Des Weiteren werden bei dieser Abarbeitung die neu generierten Events in die Queue eingebracht und an der richtigen zeitlichen Position abgelegt.

5.1.2 Einfügen neuer Agenten

Analog zur eventbasierten Steuerung der Aufzüge wurde auch bei der Organisation der Agenten diese Methode angewandt. In Abschnitt 4.2.3 wurde erwähnt, dass jeder Agent seine individuelle Startzeit und Position hat. Mit dieser ihnen zugedachten Zeit versehen verharren die Agenten in einer Queue und warten darauf, in die Simulation eingefügt zu werden. Dieser Prozess beginnt mit der Entnahme der einzufügenden Agenten aus der Queue und dem anschließenden Routen, siehe dazu Abschnitt 4.5, und endet mit der Eingliederung in den definierten Startraum. Hierbei ist zu erwähnen, dass dies nur bedeutet, dass sich der Agent in einem virtuellen Raum befindet und nun bereit ist, sich ehest möglich auf die eigentliche zelluläre Fläche zu begeben.

5.1.3 Erstellen der Trajektorien

Einer der Kernprozesse dieser Simulation ist das Erstellen der Trajektorien, also derjenigen Pfade, die die Agenten für ihre Fortbewegung durch die einzelnen Räume und Korridore nehmen. Dieser Vorgang besteht aus mehreren Schritten, die nacheinander ausgeführt werden und somit eine Abfolge von Zellwechsel hervorbringt, die die Agenten ihren Zielen näher bringen würden. Diese Abfolge ist jedoch nicht mehr als eine Art Überlegung, die jedes Individuum für sich selbst anstellt, ob sie jedoch in die Tat umgesetzt werden können, stellt sich erst bei der eigentlichen Bewegung der Agenten heraus. Nachfolgend eine Auflistung von Schritten, die zum Erstellen der Trajektorien notwendig sind.

- Einfügen der wartenden Agenten
- Neuer Routing-Vorgang
- Bestimmung der idealen Richtung
- Scannen der Umgebung

5.1.3.1 Einfügen Wartender Agenten

Für die, wie oben erwähnt, in den Räumen wartenden Agenten wird nach Platz zum Einfügen auf der zellulären Fläche gesucht. Hierzu wird jede dem Raum zugeordnete Tür nach freien Flächen durchsucht, die dem Platzbedarf der einzufügenden Agenten entspricht, und falls gefunden mit ihnen belegt. Ist das Platzangebot nicht ausreichend genug um alle Agenten unterzubringen, verbleiben die restlichen im Raum und werden beim nächsten Durchlauf erneut abgearbeitet.

5.1.3.2 Neuer Routing-Vorgang

Ein neuer Routing-Vorgang ist immer dann notwendig, wenn sich der Agent an einem Kreuzungspunkt befindet, sich also zumindest zwischen zwei Wegen entscheiden kann.

Ist das der Fall, wird, trotz des bereits zu Beginn ermittelten Weges, ein erneutes Routen vorgenommen. Dies hat den Grund, da erst zu diesem Zeitpunkt eine analytische Bewertung der Situation erfolgen kann. Der Agent ermittelt hierbei, wie in Abschnitt 3.5.1.3 beschrieben, das Verkehrsaufkommen in seiner Umgebung und aktualisiert dabei seinen persönlichen Routing-Graphen, der daraufhin die bereits vorhandene Route bestätigt oder eine neue, schnelleren findet.

5.1.3.3 Bestimmung der besten Richtung

Nach Beendigung des eventuellen Routing-Vorganges ist es nun Zeit, die bestmögliche Richtung des nächsten Schrittes zu ermitteln. Die globale und lokale Wegsuche, die jeweils in den Abschnitten 3.5.1 und 3.5.2 erläutert wurde, gibt hierbei die ideale Richtung vor, die genommen werden sollte, um so schnell wie möglich an das gewünschte Ziel zu gelangen. Dieser Vorschlag wird jedoch nur als Grundlage für den nächsten zu machenden Schritt herangezogen, da lokale Störungen, wie sie durch die Interaktion mit andere Agenten entstehen, meist eine Korrektur dieser Richtung nötig machen.

5.1.3.4 Umgebung Scannen

Nach der Bestimmung der optimalen Richtung beginnt als nächster Schritt das Analysieren der Umgebung. Hierzu analysiert der Agent seine Umgebung im Radius von 8 Metern und ordnet jede, sich in dieser Distanz befindlichen Agenten die Richtung, in der er sich in Relation zum Agenten befindet, zu. Mit Hilfe dieser Zuordnung lässt sich anschließend ein Dichtewert pro Richtung ermitteln. Des Weiteren wird eine Sammlung von Lanes erstellt, die den Bereich vor dem Agenten abdeckt und alle gefundenen Individuen sowie deren Distanzen und Richtungen enthält.

Abbildung 5.2 illustriert das Prinzip der Lanes, hier 12 an der Zahl, die sich vor dem Agenten erstrecken.

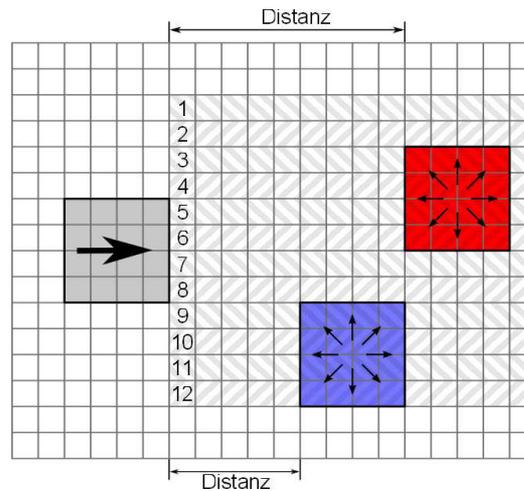


Abbildung 5.2: Agenten-Klassifizierung

Mit diesen Informationen ausgestattet kann nun die Korrektur der Richtung durchgeführt werden.

5.1.3.5 Interaktion mit Agenten

Für die Interaktion zwischen den einzelnen Agenten wurde nicht wie üblich ein einzelner Algorithmus verwendet, es wurde eine Kombination von drei Methoden implementiert, die nach Art der Interaktion zum Einsatz kommen.

1. freies Gehen (keine Interaktion)
2. Verkehr in dieselbe Richtung
3. Gegenverkehr

Zur Bestimmung des richtigen Algorithmus werden die Agenten aller Lanes in der idealen Richtung auf ihre Gehrichtung hin untersucht. Das Ergebnis davon bestimmt nun, mit welchen Methoden der Agent auf seine Umgebung reagiert und somit den möglichst besten Pfade ermittelt. Nachfolgend soll für jeden einzelnen Algorithmus eine kurze Idee darüber vermittelt werden, wie sie mit der Interaktion der einzelnen Agenten umgehen.

freies Gehen

Hierbei ist vorab zu erwähnen, dass freies Gehen nicht bedeutet, dass es keine fremden Individuen gibt, die den Agenten umgeben, sondern lediglich, dass keine Beeinflussung zu erwarten ist, da die Abstände oder Geschwindigkeitsunterschiede zu groß sind. Da es keine störenden Einflüsse bezüglich umgebender Agenten gibt, wird hier lediglich überprüft, ob ein benachbarter Agent Platz benötigt, um einer Kollision auszuweichen oder einen Überholvorgang zu starten. Ist das der Fall, wird verifiziert, ob ein Wechsel der Richtung möglich ist, ohne sich selbst auf Kollisionskurs mit umgebenden Individuen oder Hindernissen zu begeben.

Verkehr in dieselbe Richtung

Werden ein oder mehrere Agenten als Hindernisse erkannt, wird der Algorithmus versuchen einen Überholvorgang zu starten. Um dies erfolgreich umzusetzen, muss ein paralleler Weg zur momentanen Richtung gefunden werden, der sich am besten eignet, das oder die Hindernisse zu passieren. Hierzu werden alle Wege links und rechts des Agenten auf ihre Tauglichkeit überprüft, wobei Tauglichkeit hier definiert ist als kein erkennbarer Gegenverkehr und kein Verkehr in dieselbe Richtung auf einer Distanz von 3 Metern.

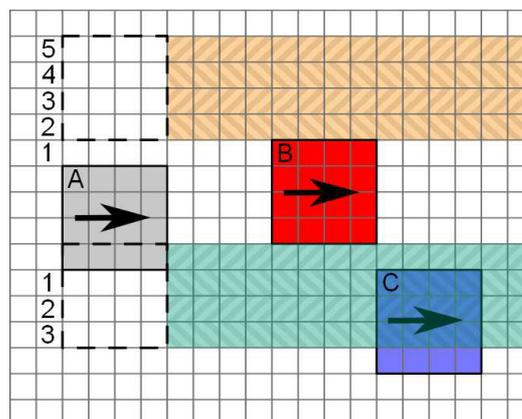


Abbildung 5.3: *Überholvorgang*

Abbildung 5.3 soll die Vorgangsweise graphisch darstellen. Agent A hat Agent B als Hindernis erkannt und versucht, den geeignetsten parallelen Weg zu finden, der ihn daran vorbeiführt. Er erkennt, dass ein Wechsel 3 Zellen nach rechts ihn Agent B passieren ließe, er aber dadurch durch Agent C blockiert würde. Erst weitere 5 Schritte nach rechts, würden eine sinnvolle Lösung darstellen und ihn an beiden Agenten vorbeiführen. Ein Wechsel nach links hingegen würde nach lediglich 5 Schritten dasselbe Ergebnis bringen, jedoch mit geringerem Aufwand. Der Algorithmus würde in diesem Fall eine Korrektur des Agenten A um 5 Schritte Richtung NE vornehmen und somit die beiden Hindernisse umgehen.

Gegenverkehr

Der Fall des Gegenverkehrs ist mit einfachen *if-else* Regeln nur schwer in den Griff zu bekommen. Zu vielfältig können sich hierbei die Situationen von Gegenverkehr und Verkehr in dieselbe Richtung darstellen. Es wurde daher auf ein Algorithmus gesetzt, der mit Hilfe einer Bewertungsfunktion allen zur idealen Richtung parallel 1-Zell-breiten Streifen nach folgender Formel ein Gewicht zuweist. [7, S. 15]

$$w_k = \sum_{j=0}^{np_k} \frac{d_j}{l_k \cdot v^i} \quad (5.1)$$

- w_k ... Gewicht des Streifens k
- np_k ... Anzahl der Agenten, die in Streifen k zu finden sind
- d_j ... Richtung des j Agenten, wobei $d_j = -1$ für Agenten in selbige Richtung und $d_j = 1$ für Agenten im Gegenverkehr
- l_k ... Gewichtungsfaktor für den Abstand zwischen Streifen und Agenten. Die Werte werden in Abbildung 5.4 dargestellt.
- v^i ... Anzahl der Zellen in Streifen k

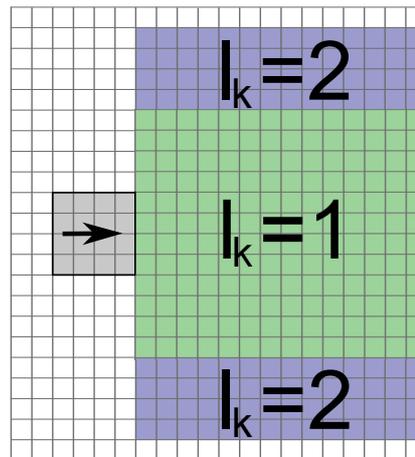


Abbildung 5.4: Gewichtung der Streifen

Die so ermittelten Gewichte sind nun ein Indikator für jene Streifen, die von Agenten gemieden werden sollten, und solchen, die ein möglichst unproblematisches Vorkommen gewährleisten. Diese Bewertungsfunktion ist in der Simulation auch für den Effekt des Ausbildens von Linien, einer Art Gänsemarsch, verantwortlich, die auch in verschiedener Literatur erwähnt wird.[25, 9] Mit dieser intuitiven Ausbildung von Linien gelingt es Gruppen von Menschen, einander zu passieren, ohne einen großen Umweg in Kauf nehmen zu müssen.

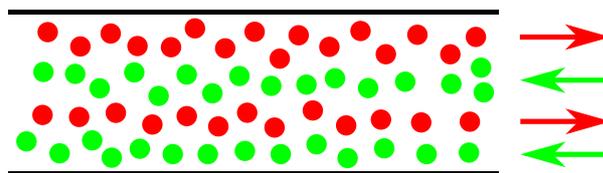


Abbildung 5.5: Bildung von Lanes

Abbildung 5.5 soll dieses Phänomen nochmals graphisch verdeutlichen.

5.1.3.6 Bewegen der Agenten

Nachdem für jeden Agenten die Trajektorien erstellt worden sind, kann nun damit begonnen werden, diese Bewegungen in die Tat umzusetzen. Hier kommt die Methode *No Crossing Path* zum Einsatz, die es den Agenten erlaubt, seine Schritte so lange auszuführen, bis er auf einen anderen Agenten stößt, der sich ihm in den Weg gestellt hat, oder auf eine Spur, also Path, trifft, den ein anderer Agent hinterlassen hat.

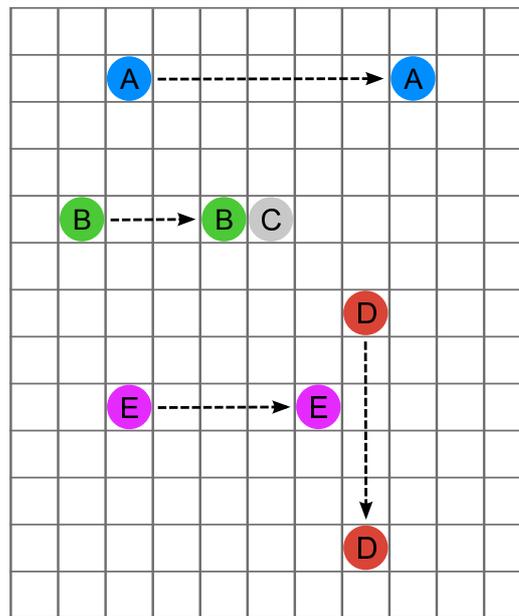


Abbildung 5.6: Bewegung *No-Crossing-Path*

In Abbildung 5.6 soll das Prinzip graphisch verdeutlicht werden. Agent *A* hat in diesem Beispiel keine Hindernisse vor sich und kann seine Trajektorie, die für ihn berechnet worden ist, komplett umsetzen. Agent *B* hat mit der Umsetzung seiner Schritte schon mehr Probleme. Er stößt nach lediglich drei Schritten auf Agent *C*, der bei der Berechnung der Trajektorien noch nicht an dieser Position gestanden hat und jetzt zum Abbruch der schrittweisen Bewegung führt. Agent *E* hat selbiges Problem, er trifft nach vier Schritten auf die Spur, die Agent *D* bei seiner Bewegung hinterlassen hat, und muss daher ebenfalls sein Vorrankommen stoppen.

Wie im letzten Beispiel ersichtlich wird, ist es bei dieser Methode unerlässlich, die Reihenfolge, in der sich die Agenten bewegen, zufällig zu wählen, da zu Beginn dieses Prozesses eine unberührte Fläche vorhanden ist, auf der die Agenten an der Position sind, die sie bei der Berechnung der Trajektorien inne hatten, und es noch keine Spuren sich bewegendender Agenten gibt.

Den Abschluss dieses Bewegungsprozesses bildet das Löschen der verursachten Spuren, um die Fläche für den nächsten Update-Schritt vorzubereiten.

5.1.3.7 Visualisierung

Der letzte Punkt im Update-Prozess bildet das Visualisieren der einzelnen Agenten, um die momentane Situation sichtbar zu machen.

5.2 Szenario

In diesem Abschnitt der Dissertation wird nun ein Szenario simuliert, das ein Phänomen im Verhalten von Fußgängern aufzeigen soll, welches im ersten Moment etwas sonderlich scheint. Es handelt sich dabei um das Platzieren eines vermeintlichen Hindernisses, wie etwa einer Säule, kurz vor einem Ausgang oder einer Engstellen. Dieses Hindernis, das im ersten Augenblick als Behinderung erscheint, die den Zugang zur Tür erschwert und somit für eine längere Zeit zur Leerung des Raumes sorgt, hat speziell im Fall einer Evakuierung den gegenteiligen Effekt. Sie fängt den Druck, den die hinteren Personen ausüben, ab und kanalisiert gleichzeitig vorab die anströmenden Menschen und erhöht somit den Durchfluss durch den Ausgang um bis zu 50 %.[10, S. 24]

5.2.1 Simulationsumgebung

Als Simulationsumgebung wurde ein Korridor mit einer Länge von 32 m sowie einer Breite von 5 m. In einem Abstand von 17 m vom Eingang entfernt befindet sich eine Mauer, in deren Mitte sich ein 0,75 m breiter Durchgang befindet. Als Hindernis wurde zentral 2 m vor diesem Durchgang eine Säule mit einem Durchmesser von 0,5 m platziert.

Zum Nachweis des in der Einleitung beschriebenen Phänomens wurden zwei Szenarien mit obiger Umgebung durchgeführt. Szenario 1 soll als Referenzergebnis dienen. Zu diesem Zweck wurde die Säule entfernt, wodurch die Personen ungehinderten und direkten Zugang zu der Engstelle erhalten. Für Szenario 2 wurde die Säule wieder hinzugefügt und stellt damit genau die Umgebung dar, wie sie in Abbildung 5.7 zu sehen ist.

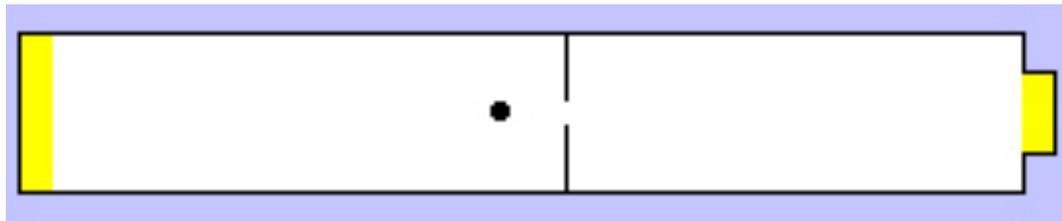


Abbildung 5.7: *Simuliertes Szenario 2 mit Säule*

5.2.2 Durchführung

Zur Durchführung der beiden Szenarien werden jeweils 40 Agenten generiert, die zur selben Zeit in der Simulation gestartet wurden und deren Ziel es ist, sich von der linken Eingangsseite zur rechten Seite des Korridors zu begeben und die Simulation durch den Ausgang zu verlassen. Die Zeitnehmung wird für jeden Agenten separat gestartet, sobald er in die Simulationsfläche eingefügt wird. Des Weiteren wird die Zeit gemessen von dem Moment an, sobald der erste Agent gestartet wird bis zu dem Zeitpunkt, zu dem der letzte Agent durch den Ausgang die Simulation verlässt.

Wert	Szenario 1 ohne Säule	Szenario 2 mit Säule
Minimum	17,50	19,00
Maximum	55,25	47,50
Durchschnitt	39,34	32,67
Varianz	11,12	8,20
Gesamt	56	49

Tabelle 5.1: *Auswertung Szenario 1 und 2 (Werte in Sekunden)*

5.2.3 Ergebnis

Um ein aussagekräftiges Ergebnis zu erhalten, wurden beide Szenarien jeweils 10 Mal durchgeführt und anschließend der Durchschnittswert der gemessenen Zeiten ermittelt. Die Ergebnisse der beiden Szenarien sind in Tabelle 5.2 ersichtlich. Für die durchschnittlichen Gesamtzeiten der beiden Szenarien wurden für Szenario 1 56 Sekunden und für Szenario 2 49 Sekunden gemessen. Das in der Einleitung beschriebene Phänomen zeigte sich also auch in diesem Experiment.

Tabelle 5.1 beinhaltet die Auswertung aus der in Tabelle 5.2 aufgelisteten Durchschnittswerte der einzelnen Agenten. Gut zu erkennen ist hierbei, dass die schnellste Zeit für die Strecke im Szenario 1, also ohne Hindernis, gemessen worden ist, jedoch alle anderen Zeiten höher sind.

Agent ID	Zeit ohne Säule	Zeit mit Säule	Agent ID	Zeit ohne Säule	Zeit mit Säule
1	26,75 s	31,50 s	21	19,25 s	20,50 s
2	53,25 s	33,00 s	22	44,25 s	27,75 s
3	45,5 s	47,50 s	23	37,00 s	21,75 s
4	54,75 s	41,00 s	24	26,75 s	26,00 s
5	53,00 s	42,75 s	25	40,25 s	19,75 s
6	38,00 s	33,25 s	26	36,00 s	23,25 s
7	29,75 s	26,00 s	27	51,25 s	30,00 s
8	32,00 s	42,25 s	28	52,50 s	31,25 s
9	48,00 s	34,75 s	29	27,25 s	38,75 s
10	44,50 s	23,25 s	30	49,25 s	44,00 s
11	55,25 s	32,75 s	31	25,75 s	22,50 s
12	36,00 s	44,75 s	32	45,25 s	24,75 s
13	52,00 s	29,50 s	33	17,50 s	21,25 s
14	33,50 s	22,50 s	34	43,75 s	37,25 s
15	46,50 s	35,50 s	35	25,00 s	19,00 s
16	23,50 s	37,00 s	36	50,25 s	45,50 s
17	46,00 s	45,25 s	37	29,50 s	29,00 s
18	53,50 s	36,00 s	38	44,75 s	40,00 s
19	41,00 s	38,75 s	39	18,25 s	39,25 s
20	45,75 s	37,50 s	40	31,50 s	30,50 s

Tabelle 5.2: *Ergebnisse Szenario 1 und 2*

Kapitel 6

Zusammenfassung und Erkenntnisse

6.1 Zusammenfassung

In dieser Arbeit sind zu Beginn mehrere Verfahren aufgezeigt worden, die zur Simulation von Fußgängerströmen Verwendung finden. Im Anschluss daran wurde in Kapitel 3 ein Modell, das auf der Methode der agentenbasierten Systeme basiert, vorgestellt, das verschiedene bestehende Algorithmen mit neuen Ansätzen verbindet, um somit eine Verbesserung gegenüber den einzelnen Methoden zu bieten. Die Kernpunkte dieses Modells sind die Implementierung von Personen mit eingeschränkter Mobilität sowie das dynamische Routen der Agenten, das ihnen mehr Flexibilität in Bezug auf die Wahl des schnellsten Weges bietet. Die Kapitel 4 und 5 befassen sich mit der aufwändigen Implementierung des entwickelten Modells in JAVA, das zu Testzwecken und zur Durchführung von Szenarien und Experimenten dient. Die Interaktion der Agenten untereinander wurde dabei mit Hilfe verschiedener Methoden umgesetzt, um sich den Gegebenheiten besser anpassen zu können. Dabei stellen sich auch empirische Beobachtungen wie das Ausbilden von Gassen beim Zusammentreffen größerer Gruppen ein. Den Abschluss bildet die Durchführung eines Szenarios, bei dem die Wirkung von sinnvoll platzierten Hindernissen zur Steigerung des Durchflusses untersucht wird.

6.2 Erkenntnisse

Eine der Haupteigenschaften dieser Arbeit ist, dass die alleinige Verwendung eines *Static Floor Field* zur Orientierung einige Nachteile mit sich bringt. So steigt etwa der Speicherbedarf bei aufwendigeren Simulationsumgebungen sehr schnell in einen Bereich, der trotz der heutigen Computertechnik zu einem Problem werden kann. Des Weiteren hat es sich als Problem herausgestellt, dass das Konzept des *Static Floor Field* grundsätzlich immer den kürzesten Weg liefert, der aber in vielen Fällen nicht der schnellste sein muss. Abhilfe würde hierbei das Einführen von Zwischenzielen bringen, doch erhöht diese Methode den Speicherbedarf zusätzlich, was die Umsetzung in der Praxis erschweren bis unmöglich machen würde. Um diese Probleme zu vermeiden, ist eine kombinierte Herangehensweise, wie sie in Kapitel 3 aufgezeigt wurde, zu bevorzugen.

Des Weiteren haben sich im Laufe des Modellierungs- beziehungsweise Implementierungsprozesses Fehlerquellen herauskristallisiert, deren Ursache und Minimierung in Kapitel 4 aufgearbeitet wurden. Diese Fehler, die durch die Diskretisierung der Zeit und des Raumes entstehen, können bei nicht Beachten zu einer erheblichen Verfälschung des Ergebnisses führen. Bei Korrektur dieser Ungenauigkeiten ist dieses Modell oder allgemeinere agentenbasierte Systeme durchaus in der Lage, Fußgängerflusssimulationen in adäquater Weise durchzuführen.

Die Erhöhung der Auflösung der Umgebung von $0,5 \text{ m} \times 0,5 \text{ m}$ wie sie in vielen wissenschaftlichen Arbeiten zu finden war, auf $0,125 \text{ m} \times 0,125 \text{ m}$ steigert zwar den Aufwand bei der Implementierung erheblich, bietet aber, dadurch dass ein Agent mehr als eine Zelle besetzt, Vorteile bei der Interaktion zwischen den Agenten. So lassen sich Bewegungsabläufe "feiner" ausführen und mit der Befähigung, den erforderlichen Platzbedarf kurzzeitig zu reduzieren, lassen sich auch Lücken nutzen, die in anderen Modellen bereits zur Blockade und daraus zum folgenden Stillstand führen.

Die Modellierung der Aufzüge ist für die Simulation von Personen, die unter einer physischen Behinderung leiden unumgänglich. Für die Aufzugssteuerung, vor allem von gruppierten Fahrstühlen, ist die in Kapitel 3 vorgestellte Umsetzung ausreichend,

um die an sie gestellten Anforderungen zu erfüllen. Das Thema der Steuerung und die Optimierung der Strategien, wie einzelne Kabinen verteilt werden, an sich würden schon Stoff für eine eigene Dissertation bieten und den Rahmen dieser Arbeit sprengen.

Schlussendlich kann das Resümee gezogen werden, dass das Modell die Dynamik von Fußgängerströmen in Gebäuden in ausreichender Qualität abbilden kann. Besonders hervorzuheben ist hierbei die Implementierung der speziellen Bedürfnisse von Menschen mit physischer Beeinträchtigung, durch deren Simulation schon im Vorfeld Behinderungen im Umfeld erkannt und vermieden werden könnte.

Abbildungsverzeichnis

2.1	Makroskopische Simulation mittels Lattice-Boltzmann Methode [15] . . .	9
2.2	Beispiel Zell Geometrien	11
2.3	Von-Neumann- Nachbarschaft	11
2.4	Moore- Nachbarschaft	11
2.5	Transformation eines Gebäudes in die Ebene	12
2.6	Beispiel Static Floor Field	13
2.7	Wahrscheinlichkeitsmatrix	15
2.8	Kollisionsvermeidung	17
2.9	Beschleunigung im Magnetic Force Model	23
2.10	Warteschlangen Knoten	24
3.1	Screenshot Simulator	25
3.2	Visualisierung verschiedener Ebenen	27
3.3	Platzbedarf Agenten	29
3.4	Weidmann-Verteilung	31
3.5	Geschwindigkeitskomponenten auf Treppen [5]	32
3.6	Näherungsrelation von Kladek bei $v_h^0 = 1,34$ m/s	34
3.7	Beispiel eines Graphen	37
3.8	Beispiel eines Voronoi-Diagramms	39
3.9	Beispiel Dijkstra Suche	42
3.10	Von-Neumann Distanz	44
3.11	Moore Distanz	44
3.12	Konvexer Raum	45

3.13	Nicht konvexer Raum	45
3.14	Potentiallinien L^2 -Norm	48
3.15	Zellen des arithmetischen Mittels	49
3.16	Darstellung von Wegsegmenten	50
3.17	Fehler beim Überspringen von Wegpunkten	51
3.18	Zustandsübergangsdiagramm	57
3.19	Wartezeiten für Aufzüge	58
3.20	Drei-Abschnitte-Konzept	60
4.1	Screenshot Projekt öffnen	68
4.2	Beispiel *.png Datei	73
4.3	Screenshot Agenten Definition	75
4.4	Screenshot Agenten öffnen	75
4.5	Screenshot DOOR-Element	78
4.6	Screenshot CHANGE_AREA-Element	79
4.7	Screenshot ELEVATOR_AREA-Element	80
4.8	Screenshot Wahrscheinlichkeits-Panel	82
4.9	Screenshot Wahrscheinlichkeitsverteilung des Alters	82
4.10	Darstellung <i>ELEVATOR_AREA</i>	85
4.11	Informationsfeld für Aufzüge	86
4.12	Ursache des Raster Fehlers	92
4.13	Alternative Route	93
4.14	Diskrete Kreise	94
4.15	Relativer Fehler	95
5.1	Screenshot Simulationsmodus	97
5.2	Agenten-Klassifizierung	101
5.3	Überholvorgang	102
5.4	Gewichtung der Streifen	104
5.5	Bildung von Lanes	104
5.6	Bewegung <i>No-Crossing-Path</i>	105
5.7	Simuliertes Szenario 2 mit Säule	107

Tabellenverzeichnis

2.1	Klassifizierung nach Gershenfeld [6]	7
3.1	Gehgeschwindigkeiten in der Ebene nach Weidmann	31
3.2	Mittlere Gehgeschwindigkeiten auf Treppen [2, S. 7]	33
3.3	Zustandsübergangstabelle	57
4.1	Grundfarben der Bilddateien	73
4.2	Bedeutung des Alphakanals	74
5.1	Auswertung Szenario 1 und 2 (Werte in Sekunden)	108
5.2	Ergebnisse Szenario 1 und 2	109

Literaturverzeichnis

- [1] Bruckner. *Modellierung der Fußgängerdynamik im universitären Betrieb mit Hilfe von Zellulären Automaten in der Programmiersprache JAVA*. 2009.
- [2] U Brunner, H Kirchberger, C Lebeda, M Oswald, R Könnecke, M Kraft, and T Kretz. Rimea-richtlinie für mikroskopische entfluchtungsanalysen 2.2. 1, 2009.
- [3] C Burstedde, K Klauck, A Schadschneider, and J Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 295(3–4):507 – 525, 2001.
- [4] Michael J. North Charles M. Macal. Tutorial on agent-based modeling and simulation part 2: How to model with agents. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 73 – 83, 2006.
- [5] Taku Fujiyama. Pedestrian speeds on stairs. 2004.
- [6] Neil A Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, Cambridge, 1999.
- [7] JorgeD. González, M.Luisa Sandoval, and Joaquín Delgado. Social field model to simulate bidirectional pedestrian flow using cellular automata. In Valery V. Kozlov, Alexander P. Buslaev, Alexander S. Bugaev, Marina V. Yashina, Andreas Schadschneider, and Michael Schreckenberg, editors, *Traffic and Granular Flow '11*, pages 197–206. Springer Berlin Heidelberg, 2013.

-
- [8] Dirk Helbing. A fluid dynamic model for the movement of pedestrians. *Complex Systems* 6, pages 391–415, 1992.
- [9] Dirk Helbing, Lubos Buzna, Anders Johansson, and Torsten Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science*, 39(1):1–24, 2005.
- [10] Dirk Helbing, Illes J Farkas, Peter Molnar, and Tamás Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian and evacuation dynamics*, 21:21–58, 2002.
- [11] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51:4282–4286, May 1995.
- [12] Hajime Inamura Kardi Teknomo, Yasushi Takeyama. Review on microscopic pedestrian simulation model. In *Proceedings Japan Society of Civil Engineering Conference*, 2000.
- [13] Ekaterina Kirik, Tat'yana Yurgel'yan, and Dmitriy Krouglov. An intelligent floor field cellular automation model for pedestrian dynamics. In *Proceedings of the 2007 Summer Computer Simulation Conference, SCSC '07*, pages 21:1–21:6, San Diego, CA, USA, 2007. Society for Computer Simulation International.
- [14] Tobias Kretz. *Pedestrian Traffic-Simulation and Experiments*. PhD thesis, Universität Duisburg-Essen, Fakultät für Physik» Theoretische Physik, 2007.
- [15] Daniel Leitner. *Simulation of Arterial Blood flow with the Lattice Boltzmann Method*. PhD thesis, Technische Universität Wien, 2007.
- [16] G.G. Lovas. Modeling and simulation of pedestrian traffic flow. *Transportation Research Part B: Methodological*, 28(6):429 – 443, 1994.
- [17] Katsuhiro Nishinari, Ansgar Kirchner, Alireza Namazi, and Andreas Schadschneider. Extended floor field ca model for evacuation dynamics. *IEICE Transactions*, 87-D(3):726–732, 2004.

-
- [18] Michael Orth. Virtuelle evakuierung. *PROTECTOR*, 09 2012.
- [19] Nuria Pelechano and Ali Malkawi. Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches. *Automation in Construction*, 17(4):377 – 385, 2008.
- [20] Gudwin R. A fuzzy elevator group controller with linear context adaptation. In *Fuzzy Systems Proceedings, 1998*, volume Vol. 1, pages 481 – 486, 1998.
- [21] Jeffery W. Rankin, W. Mark Richter, and Richard R. Neptune. Individual muscle contributions to push and recovery subtasks during wheelchair propulsion. *Journal of Biomechanics*, 44(7):1246 – 1252, 2011.
- [22] A. Rong, H. Hakonen, and R. Lahdelma. *Estimated Time of Arrival (ETA) Based Elevator Group Control Algorithm with More Accurate Estimation*. TUCS technical report. Turku Centre for Computer Science, 2003.
- [23] Satoshi Matsushita Shigeyuki Okazaki. A study of simulation model for pedestrian movement with evacuation and queuing. In *Proceeding of the International Conference on Engineering for Crowd Safety*, pages 271–280, 1993.
- [24] Weiguo Song, Yanfei Yu, Weicheng Fan, and Heping Zhang. A cellular automata evacuation model considering friction and repulsion. *Science in China Ser. E Engineering & Materials Science*, 48(4):403–413, 2005.
- [25] Xiong Tao, Zhang Peng, Wong S. C., Shu Chi-Wang, and Zhang Meng-Ping. A macroscopic approach to the lane formation phenomenon in pedestrian counterflow. *Chinese Physics Letters*, 28(10):108901, 2011.
- [26] Kardi Teknomo. *Microscopic Pedestrian Flow Characteristics: Development of an Image Processing Data Collection and Simulation Model*. PhD thesis, Tohoku University, 2002.
- [27] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme. *Modeling Software with*

-
- Finite State Machines: A Practical Approach*. Number Bd. 0 in Modeling Software with Finite State Machines: A Practical Approach. Taylor & Francis, 2006.
- [28] U. Weidmann. *Transporttechnik der Fussgänger: Transporttechnische Eigenschaften des Fussgängerverkehrs (Literaturauswertung)*. Schriftenreihe des IVT. ETH, IVT, 1993.
- [29] Paul Williamson. The role of the international journal of microsimulation. *INTERNATIONAL JOURNAL OF MICROSIMULATION*, 1:1, 2007.
- [30] Lu Yu. Multi-car elevator system using genetic network programming. In *SICE Annual Conference*, pages 127 – 131, 2008.

Martin Bruckner

Curriculum Vitae

Persönliche Daten

Name Martin Bruckner
Geboren am 01. März 1976
Nationalität Österreich

Ausbildung

- seit 10/2009 **Doktoratsstudium der technischen Wissenschaften**
Technische Universität Wien
Agentenbasierte Simulation von Personenströmen mit unterschiedlichen Charakteristiken
- 10/2002-10/2009 **Diplomstudium der technischen Mathematik**
Technische Universität Wien
Mathematik in den Computerwissenschaften
- 2001-2002 **Studienberechtigungsprüfung**
Volkshochschule Floridsdorf, Wien
- 06/1994 **Fachschule Elektrotechnik**
HTL Hollabrunn

Berufserfahrung

- seit 04/2012 **dwh GmbH**, Projektmitarbeiter
- 10/2009-02/2012 **Technische Universität Wien**
Projektmitarbeiter
- 09/1999-08/2003 **Rittmeyer Mess-Leittechnik**
Projektsachbearbeiter
- 06/1997-08/1999 **ELIN-Wasserwerkstechnik**
Servicetechniker
- 01/1995-03/1996 **Österreichischen Bundesheer**

Sprachliche Fähigkeiten

Deutsch Muttersprache

Englisch fließend

Sonstige Fähigkeiten

Programmierkenntnisse JAVA, C++, C#, MATLAB, SQL, LabVIEW