

BACHELOR THESIS  
René Scheuer

# Entwicklung einer Experimentierplattform für den Vergleich verschiedener sozialer Agentenmodelle am Beispiel von Hamburger Weihnachtsmärkten

---

FAKULTÄT TECHNIK UND INFORMATIK  
Department Informatik

Faculty of Engineering and Computer Science  
Department Computer Science

René Scheuer

# Entwicklung einer Experimentierplattform für den Vergleich verschiedener sozialer Agentenmodelle am Beispiel von Hamburger Weihnachtsmärkten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung  
im Studiengang *Bachelor of Science Wirtschaftsinformatik*  
am Department Informatik  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen  
Zweitgutachter: Prof. Dr. Michael Köhler-Bußmeier

Eingereicht am: 06.10.2025

**René Scheuer**

**Thema der Arbeit**

Entwicklung einer Experimentierplattform für den Vergleich verschiedener sozialer Agentenmodelle am Beispiel von Hamburger Weihnachtsmärkten

**Stichworte**

SmartOpenHamburg, MARS, Agenten, Pedestrian Dynamics

**Kurzzusammenfassung**

In dieser Bachelorarbeit wird eine Experimentierplattform innerhalb des Frameworks SmartOpenHamburg entwickelt. Verschiedene Modelle der Fußgängerdynamiken werden in dieser Experimentierplattform miteinander verglichen.

**René Scheuer**

**Title of Thesis**

Development of a Simulation Platform for Comparing Social Agent Models Using the Example of Hamburg Christmas Markets

**Keywords**

SmartOpenHamburg, MARS, Agents, Pedestrian Dynamics

**Abstract**

This bachelor's thesis presents the development of an experimental platform integrated into the SmartOpenHamburg framework. Its purpose is to facilitate the comparison of different models within the field of pedestrian dynamics.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>vii</b>
<b>Tabellenverzeichnis</b>	<b>viii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Hintergrund und Motivation . . . . .	1
1.2 SmartOpenHamburg-Modell und MARS . . . . .	2
1.3 Agenten und Multi-Agenten-Systeme . . . . .	2
1.4 Forschungsfrage . . . . .	3
<b>2 Grundlagen</b>	<b>4</b>
2.1 Agentenbasierte Simulation mit MARS . . . . .	4
2.1.1 Agents . . . . .	4
2.1.2 Entities . . . . .	4
2.1.3 Layers . . . . .	5
2.1.4 Environments . . . . .	5
2.1.5 Simulationsablauf . . . . .	5
2.2 Das SmartOpenHamburg Modell . . . . .	6
2.2.1 Traveler . . . . .	6
2.2.2 Multimodal Layer . . . . .	7
2.3 Konfiguration und Szenariendefinition . . . . .	7
2.4 Pedestrian Dynamics . . . . .	8
2.4.1 Empirische Untersuchung und Datenerhebung . . . . .	8
2.4.2 Theoretische Modellierung . . . . .	8
2.4.3 Validierung der Modelle . . . . .	9
<b>3 Auswahl der sozialen Agentenmodelle</b>	<b>10</b>
3.1 Social Force Model . . . . .	10
3.1.1 Grundidee und Funktionsweise . . . . .	10

3.1.2	Modellklassifizierung . . . . .	10
3.1.3	Diskussion und Anwendungsbereiche . . . . .	11
3.2	Collision Free Speed Model . . . . .	11
3.2.1	Grundidee und Funktionsweise . . . . .	11
3.2.2	Modellklassifizierung . . . . .	11
3.2.3	Diskussion und Anwendungsbereiche . . . . .	12
3.3	Optimal Steps Model . . . . .	12
3.3.1	Grundidee und Funktionsweise . . . . .	12
3.3.2	Modellklassifizierung . . . . .	13
3.3.3	Diskussion . . . . .	13
<b>4</b>	<b>Entwurf der Experimentierplattform</b>	<b>14</b>
4.1	Anforderungsanalyse und Zielsetzung . . . . .	14
4.1.1	Funktionale Anforderungen . . . . .	14
4.1.2	Nichtfunktionale Anforderungen . . . . .	15
4.2	Konzeptionelle Modellierung der Domäne . . . . .	16
4.2.1	Das Domänenmodell . . . . .	16
4.2.2	Das fachliche Datenmodell . . . . .	17
4.3	Architekturentwurf der Experimentierplattform . . . . .	18
4.3.1	Struktur der Layer . . . . .	18
4.3.2	Entwurf der Agenten und Entitäten . . . . .	18
4.4	Grundlage für die Implementierung . . . . .	19
4.4.1	Definition der Simulationsszenarien . . . . .	19
<b>5</b>	<b>Technische Implementierung</b>	<b>20</b>
5.1	Technologiestack und Entwicklungsumgebung . . . . .	20
5.2	Softwarearchitektur und Klassenstruktur . . . . .	20
5.2.1	Das Designmodell . . . . .	20
5.2.2	Programmablauf und Initialisierung . . . . .	23
5.3	Implementierung der Kernkomponenten . . . . .	23
5.3.1	Implementierung der Agentenmodelle . . . . .	24
5.3.2	Anpassung des Travelers . . . . .	24
5.3.3	Verarbeitung des Inputs . . . . .	25
5.3.4	Modellieren des Weihnachtsmarktes . . . . .	26
<b>6</b>	<b>Ergebnisse und Diskussion</b>	<b>29</b>
6.1	Validierung der Anforderungen . . . . .	29

6.2	Messergebnisse . . . . .	31
6.3	Visualisierung des Bewegungsverhaltens . . . . .	32
6.4	Diskussion . . . . .	34
<b>7</b>	<b>Fazit</b>	<b>35</b>
7.1	Schlussfolgerung . . . . .	35
7.2	Ausblick . . . . .	35
	<b>Literaturverzeichnis</b>	<b>36</b>
	<b>Selbstständigkeitserklärung</b>	<b>47</b>

# Abbildungsverzeichnis

2.1	Simulationsablauf mit MARS [6]	5
2.2	Klassifizierung von Modellarten in der Pedestrian Dynamics [8]	8
3.1	Eine Veranschaulichung der benachbarten Felder eines Agenten in dem Optimal Steps Model [10]	12
4.1	Das Domänenmodell für die Plattform	16
4.2	Das fachliche Datenmodell für die Plattform	17
5.1	Das Designmodell in Form eines UML-Klassendiagramms für die Plattform mit Berücksichtigung von MARS	27
5.2	Lageplan des Weihnachtsmarktes am Hamburger Rathaus [1]	28
6.1	Visualisierung des Social Force Models in der Experimentierplattform mittels Kepler	32
6.2	Visualisierung des Collision Free Speed Models in der Experimentierplattform mittels Kepler	33
6.3	Visualisierung des Optimal Steps Models in der Experimentierplattform mittels Kepler	33

# Tabellenverzeichnis

6.1	Systematischer Vergleich der Leistungsmetriken der Agentenmodelle . . .	31
1	Detaillierte Besucherzahlen pro Stand für das OptimalSteps-Modell . . . .	37
2	Detaillierte Besucherzahlen pro Stand für das CollisionFreeSpeed-Modell .	40
3	Detaillierte Besucherzahlen pro Stand für das SocialForce-Modell . . . . .	43

# 1 Einleitung

## 1.1 Hintergrund und Motivation

Die Urbanisierung stellt sich vor die Herausforderung von komplexen Interaktionen im öffentlichen Raum. Insbesondere bei Großveranstaltungen mit einer hohen Dichte an Menschen, wie etwa Demonstrationen oder Weihnachtsmärkten, sind Faktoren wie die Sicherheit und der Komfort der Menschen ein wichtiges Ziel. Dies fordert ein besseres Verständnis für die Bewegungsmuster von Menschen in Menschenmengen [3].

Agentenbasierte Simulationen haben sich als ein wertvolles Werkzeug für dieses Verständnis etabliert. Agentenbasierte Simulationen ermöglichen es, Menschenmengen zu modellieren und dabei für jeden einzelnen Menschen das individuelle Verhalten zu definieren [13]. Um das Verhalten hierbei möglichst realistisch abzubilden, benötigt es spezialisierte Verhaltensmodelle. Diese Modelle stellen soziale und kognitive Fähigkeiten und eine eigene Entscheidungsfindung sicher. So werden beispielsweise Faktoren wie das Einhalten des Abstands zu anderen Menschen, das Verfolgen von eigenen Zielen oder das Ausweichen von Hindernissen in den Fokus genommen.

Im Kontext der Fußgängerdynamik in agentenbasierten Simulationen existiert in der Wissenschaft ein Abschnitt, der sich mit der Entwicklung und der Analyse solcher Modelle beschäftigt, die Pedestrian Dynamics [9]. Auf die grundlegenden Konzepte der Pedestrian Dynamics wird in Kapitel 2 näher eingegangen. Das Ziel ist es jedoch, im Rahmen dieser Bachelorarbeit mehrere Modelle der Pedestrian Dynamics in einer selbst entwickelten Experimentierplattform miteinander zu vergleichen und die Plattform für zukünftige Implementationen weiterer Verhaltensmodelle zur Verfügung zu stellen.

## 1.2 SmartOpenHamburg-Modell und MARS

Das SmartOpenHamburg Projekt ist in Zusammenarbeit mit der Stadt Hamburg ins Leben gerufen worden, um die Verkehrslandschaft in Hamburg zu analysieren, und zu optimieren. Die MARS-Group stellt hierzu ein Multi-Agenten-System zur Verfügung, mit dem der Verkehr der Stadt Hamburg modelliert werden kann [7]. Das Ziel des Projekts ist es, anhand von einer georeferenzierten Karte des Hamburger Verkehrs, georeferenzierten Strukturen der Stadt und komplexen Bewegungsalgorithmen eine Basis für die Stadtplanung in Hamburg und für weitere Smart Cities der Zukunft zu bilden [12].

Das SmartOpenHamburg Modell stellt eine Simulation aus einer von dem Nutzer bestimmten Konfiguration her. Die grundlegende Basis besteht darin, verschiedene Modalitäten wie das Gehen, Fahrradfahren, Autofahren, Bahnfahren etc. innerhalb eines frei bestimmbar Gebietes im Raum Hamburg über einen Zeitraum als Konfiguration festzulegen. Diese Konfiguration wird anhand von einer Simulation von dem SmartOpenHamburg Modell analysiert und dem Nutzer in Form des .geojson-Formats zurückgegeben, sodass der Benutzer selbst die Simulation in beliebigen Geodatenanalyse-Tools wie beispielsweise Kepler nachverfolgen kann.

SmartOpenHamburg wird als Basisframework für die Experimentierplattform, die die verschiedenen Verhaltensmodelle miteinander vergleichen wird, genutzt.

## 1.3 Agenten und Multi-Agenten-Systeme

SmartOpenHamburg ist prinzipiell ein riesiges Multi-Agenten-System, welches dem Nutzer eine agentenbasierte Simulation bereitstellt.

Ein Multi-Agenten-System besteht aus einer Menge von Agenten, die jeweils eigene Ziele, Wahrnehmungen und Entscheidungen treffen können [2]. Die Interaktion zwischen diesen Agenten erfolgt in der Regel durch Kommunikation oder koordinierte Aktionen, was sie vor gemeinsame Herausforderungen stellt. Agenten können eigene Entscheidungen treffen, Muster erkennen und strategisch planen, was sie in Bewegungsmustern und Problemlösung zeigen [13]. Dies ist insbesondere für das SmartOpenHamburg-Projekt interessant. Denn im Fall des SmartOpenHamburg Modells bieten sich Agenten als Akteure für menschliche Bürger der Stadt an. Diese Agenten haben einen Tagesplan, können verschiedene Modalitäten nutzen, und bewegen sich aktiv auf der Karte der Stadt, was

den Verkehr, oder spezielle Situationen wie Demonstrationen, oder Festivitäten wie beispielsweise im Rahmen dieser Bachelorarbeit, Weihnachtsmärkte, widerspiegeln kann.

Die Idee, Agenten menschlicher zu gestalten, ist es diese Agenten mit den nötigen Informationen zu parametrisieren. Das Ziel dieser Parametrisierung ist es, dass die Agenten möglichst realitätsnah die Bürger der Stadt repräsentieren und deren Eigenschaften gerecht werden. Im Rahmen dieser Arbeit ist es von höchster Bedeutung, mit wenig Aufwand zwischen verschiedenen sozialen Agentenmodellen zu wechseln und eine einfache Implementierung neuer Modelle zu garantieren. Es soll ein sichtbarer Unterschied des Verhaltens der Agenten mit dem Wechsel der Verhaltensmodelle entstehen, ohne dass die Parametrisierung der Agenten beeinflusst werden muss.

### 1.4 Forschungsfrage

Meine Bachelorarbeit wird sich mit der Frage auseinandersetzen, wie eine Experimentierplattform innerhalb des Frameworks SmartOpenHamburg gestaltet sein kann, um in dem beispielhaften Raum des Hamburger Weihnachtsmarktes am Rathausmarkt einen systematischen Vergleich von sozialen Agentenmodellen (Pedestrian Dynamics) zu ermöglichen.

## 2 Grundlagen

### 2.1 Agentenbasierte Simulation mit MARS

MARS (Multi-Agent Research and Simulation) ist ein in C# geschriebenes Framework zur Entwicklung und Durchführung agentenbasierter Simulationen. Das Kernkonzept von MARS basiert auf vier verschiedenen Klassenarten, die anhand der Dokumentation [5] im Folgenden erläutert werden.

#### 2.1.1 Agents

Agenten sind die aktiven, handelnden Entitäten in einer MARS-Simulation. Sie sind autonom, besitzen einen eigenen Zustand und können eigene Entscheidungen treffen. Die Verhaltenslogik wird in der jeweiligen `Tick()`-Methode implementiert. Die `Tick()`-Methode wird mit jedem Zeitschritt der Simulation erneut ausgeführt. Der Agent kann dabei seine Umgebung wahrnehmen, seinen internen Zustand verändern und Aktionen ausführen.

Im Kontext dieser Arbeit stellen Agenten die Besucher des Weihnachtsmarktes dar, deren Bewegungsverhalten im Laufe der Arbeit untersucht wird.

#### 2.1.2 Entities

Im Gegensatz zu Agenten sind die Entitäten passive Objekte, die mit dem Start der Simulation erstellt werden, aber keine eigene Verhaltenslogik und keine `Tick()`-Methode besitzen. Entitäten können anhand ihrer `UUID` von den Agenten wahrgenommen werden.

Im Kontext dieser Arbeit stellen Entitäten die Weihnachtsmarktstände dar, zu denen sich die Agenten bewegen.

### 2.1.3 Layers

In MARS sind Layers die Umgebung, in der die Agenten leben. Sie dienen als Container, welche Agenten und Entitäten, die auf ihnen liegen, verwalten.

Im Kontext dieser Arbeit wird ein Layer die Marktfläche bilden und somit die Marktbesucher und Marktstände verwalten.

### 2.1.4 Environments

Das Environment ist eine räumliche Datenstruktur, welche Agenten den Zugriff auf Ressourcen und Netzwerke ermöglicht. Die Hauptfunktion von Environments besteht darin, dem Agenten verschiedene Methoden zur Bewegung und Interaktion zur Verfügung zu stellen.

Unterstützte Methoden sind beispielsweise:

- `Insert () + Remove ()` zum Positionieren oder Entfernen von Agenten.
- `Move ()`, um Agenten oder Entitäten die Möglichkeit zum Bewegen zu gewähren.
- `Explore ()`, um Agenten zu erlauben, nach anderen Agenten, Entitäten oder Objekten zu suchen und mit ihnen zu interagieren.

### 2.1.5 Simulationsablauf

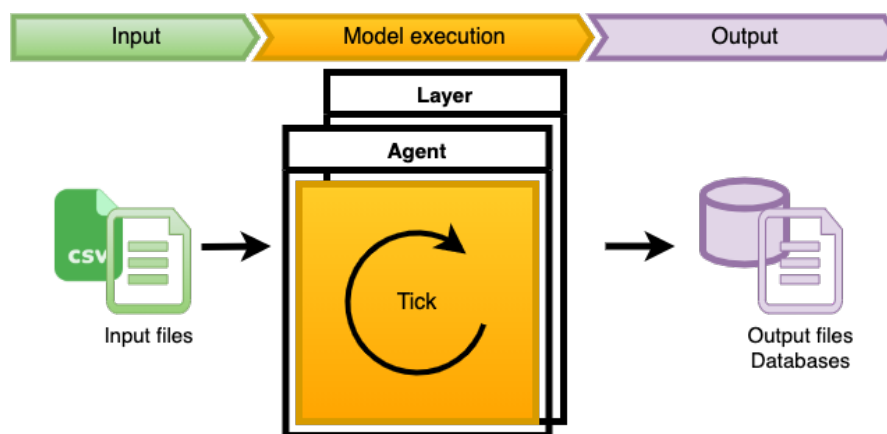


Abbildung 2.1: Simulationsablauf mit MARS [6]

Ein Simulationsablauf sieht wie folgt aus: Mit der Angabe von Eingabedateien kann die Simulation gestartet werden. Mit jedem Aufruf der `Tick()`-Methode bewegen sich Agenten, interagieren möglicherweise mit anderen Agenten oder Entitäten, und führen so lange `Tick()` aus, bis die Simulationszeit, die in den Eingabedateien festgelegt wurde, zu Ende läuft. Nach Ablauf der Simulation können dem Nutzer Dateien ausgegeben werden. Datenbankeinträge können dabei ebenfalls entstehen.

Typischerweise wird in den Eingabedateien definiert, welche Agenten, Entitäten und Layer für den jeweiligen Simulationsablauf erstellt werden.

## 2.2 Das SmartOpenHamburg Modell

SmartOpenHamburg ist eine konkrete Umsetzung des MARS-Frameworks, die speziell für die Modellierung und agentenbasierte Simulation des Metropolraums Hamburg entwickelt wurde. Es stellt eine Bibliothek verschiedener Szenarien von vorgefertigten Agenten, Entitäten, Layern und Environments bereit. Für diese Arbeit sind insbesondere die folgenden Klassen von großer Bedeutung.

### 2.2.1 Traveler

Der Traveler ist eine Basis-Agentenklasse in SOH, welche mobile Akteure darstellt. Er erweitert den generischen MARS-Agenten um grundlegende Fähigkeiten und Attribute. Dazu gehört Folgendes:

- Ein Start- und Zielpunkt ist definiert.
- Der Agent versucht zu dem Zielpunkt zu gelangen.
- Der Agent besitzt die Fähigkeit zur multimodalen Wegfindung.
- Der Agent entfernt sich aus der Simulation, sobald das Ziel erreicht wurde.

In dieser Arbeit dient der Traveler als Basisklasse für alle zu implementierenden Weihnachtsmarktbesucher. Er stellt die grundlegende Bewegung entlang eines Graphen zur Verfügung.

### 2.2.2 Multimodal Layer

SmartOpenHamburg stellt eine Vielzahl von Layern zur Verfügung, die auf die Verwaltung, insbesondere von Verkehrsnetzen, spezialisiert sind. Das Multimodal Layer ermöglicht verschiedene Arten zur Fortbewegung wie Fußweg, Fahrrad, Bahn und weitere. Im Rahmen dieser Arbeit wird jedoch ausschließlich der Fußverkehr beachtet. Für die zukünftige Arbeit an diesem Projekt ergibt es Sinn, das Multimodal Layer beizubehalten, damit das Implementieren anderer Modalitäten möglichst einfach ist.

## 2.3 Konfiguration und Szenariendefinition

Eine MARS-Simulation, insbesondere im Rahmen von SmartOpenHamburg, wird durch zwei zentrale Bestandteile definiert: die in C# geschriebene Modellbeschreibung und eine externe Konfigurationsdatei im JSON-Format.

Die Konfigurationsdatei `config.json` legt die spezifischen Details für den Simulationsdurchgang fest. Sie ist dabei für die meisten bereits vorhandenen Szenarien in drei Teile gegliedert:

- Globale Parameter: Hier werden die grundlegenden Rahmenbedingungen für die Simulationen definiert. Dazu gehören der zeitliche Startpunkt, der zeitliche Endpunkt, sowie die Dauer eines einzelnen Simulationsschritts.
- Layer-Konfiguration: Dieser Abschnitt initialisiert und parametrisiert die in der Modellbeschreibung registrierten Layer. Hier können unter anderem die Geodaten für die Umgebung oder die Positionen von Entitäten angegeben werden. Andererseits gibt es hier die Möglichkeit, eine Datei anzugeben, welche einen Plan für die Erzeugung von Agenten enthält.
- Agenten-Konfiguration: Hier können unter anderem die Trajektorien der Agenten angezeigt werden und weitere individuelle Parameter für das Agentenverhalten angegeben werden.

## 2.4 Pedestrian Dynamics

In dieser Arbeit werden daher Modelle aus dem Feld der Pedestrian Dynamics als eine Implementierung von sozialen Agentenmodellen verstanden.

Pedestrian Dynamics ist das wissenschaftliche Forschungsfeld, das sich im Kontext von agentenbasierten Simulationen mit der Modellierung, Simulation und Analyse der Bewegung von Menschenmengen, auf Fußgängerebene befasst. Zentrale Ziele sind dabei die Sicherheit, beispielsweise bei der Planung von Evakuationsruten, sowie der Steigerung der Effizienz von Personenflüssen, insbesondere in Großveranstaltungen, die von vielen Menschen besucht werden [9], wie einem Weihnachtsmarkt.

Die zentralen Aufgaben der Pedestrian Dynamics lassen sich in drei Kernbereiche unterteilen [8].

### 2.4.1 Empirische Untersuchung und Datenerhebung

Die Grundlage des Fachgebiets besteht darin, Fußgängerströme zu beobachten und zu messen, und daraus verlässliche und reproduzierbare Daten zu gewinnen.

### 2.4.2 Theoretische Modellierung

Aufbauend auf den empirischen Erkenntnissen werden mathematische Modelle entwickelt, um das Verhalten in agentenbasierten Simulationen zu beschreiben. Hierbei wird zwischen verschiedenen Modellarten unterschieden.

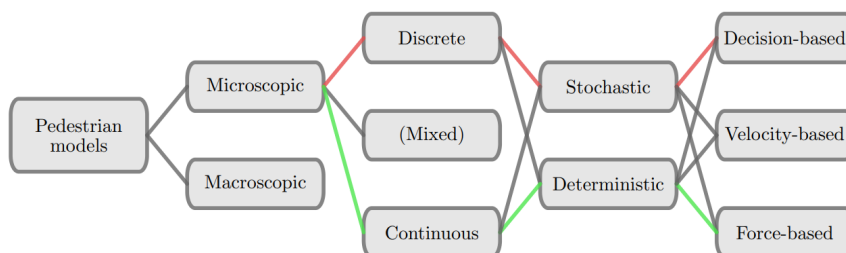


Abbildung 2.2: Klassifizierung von Modellarten in der Pedestrian Dynamics [8]

1. Makroskopische Modelle: Diese Modelle beschreiben den Fußgängerstrom als Ganzes. Sie unterscheiden nicht zwischen einzelnen Agenten, daher werden auch keine individuellen Entscheidungen oder Interaktionen modelliert.
2. Mikroskopische Modelle: Diese Modelle betrachten jeden Fußgänger als einen individuellen Agenten mit individuellen Eigenschaften (Position, Geschwindigkeit, Ziel), die ihre eigenen Entscheidungen treffen. Das Verhalten der Menge ergibt sich hierbei aus der Summe des Verhaltens aller einzelnen Agenten.
3. Beschleunigungsbasierte Modelle (Acceleration/Force-based): Diese Modelle nehmen an, dass die Bewegung von Agenten durch soziale Kräfte gesteuert wird. Es gibt eine treibende Kraft, die einen Agenten zu seinem Ziel hin beschleunigt und abstoßende Kräfte von anderen Agenten und Hindernissen, damit Kollisionen vermieden werden.
4. Geschwindigkeitsbasierte Modelle (Velocity-based): In diesen Modellen wird die Geschwindigkeit einer Person an die lokale Umgebung angepasst. Häufig basieren diese Modelle auf Optimierungsstrategien, bei der eine Person ihre Geschwindigkeit eventuell verändert, um Kollisionen zu vermeiden.
5. Entscheidungsbasierte Modelle (Decision-based): Diese Modelle basieren auf dem Konzept von zellulären Automaten. Die Agenten treffen mit jedem Zeitschritt die Entscheidung, wie sie sich bewegen sollen. Dazu analysieren sie stets ihre unmittelbare Umgebung.

### 2.4.3 Validierung der Modelle

Der finale Schritt ist der Abgleich der theoretischen Modelle mit den empirischen Daten. Die Modelle werden somit validiert, um sicherzustellen, dass sie korrekt die Realität abbilden und verlässliche Aussagen treffen können.

## 3 Auswahl der sozialen Agentenmodelle

### 3.1 Social Force Model

#### 3.1.1 Grundidee und Funktionsweise

Das Social Force Model [4], entwickelt von Dirk Helbing und Péter Molnár, basiert auf der Idee, dass sich die Bewegung von Fußgängern ändert, sobald sie von sozialen Kräften beeinflusst werden. Diese sozialen Kräfte sind keine physikalischen Kräfte, sondern stellen viel mehr innere Faktoren dar, die einen Einfluss auf die Bewegung eines Agenten haben können.

Die Bewegung eines Agenten wird durch mehrere Kräfte bestimmt, die zusammen seine Beschleunigung ergeben:

1. Antriebskraft: Diese Kraft beschreibt, dass sich der Agent möglichst auf seiner gewünschten Geschwindigkeit annähert, und somit in Richtung seines Ziels läuft.
2. Abstoßende Kraft von anderen Fußgängern: Jeder Agent in der Nähe übt eine abstoßende Kraft aus. Diese Kraft wird stärker, desto näher sich die Agenten in den Weg kommen. Dies sorgt für die Vermeidung von Kollisionen.
3. Abstoßende Kraft von Hindernissen: Wände und andere Hindernisse üben ebenfalls eine abstoßende Kraft aus, die Agenten davon abhält, durch sie hindurchzulaufen oder ihnen zu nahe zu kommen.

#### 3.1.2 Modellklassifizierung

Das Social Force Model wird als beschleunigungsbasiertes Modell (Acceleration/Force-based) angesehen. Die Bewegungsgleichung des Modells definiert die Beschleunigung, basierend auf der Summe der sozialen Kräfte.

### 3.1.3 Diskussion und Anwendungsbereiche

Laut dem Paper: Pedestrian Dynamics - From Empirical Results to Modeling [8] gehören kraft- und beschleunigungsbasierte Modelle zu den am häufigsten genutzten und am besten untersuchten Modellklassen. Es wird jedoch ebenfalls darauf hingewiesen, dass raumkontinuierliche Modelle (zu denen auch das Social Force Model gehört) dazu neigen, das Verhalten in Stausituationen nicht richtig zu beschreiben.

## 3.2 Collision Free Speed Model

### 3.2.1 Grundidee und Funktionsweise

Das von Tordeux et al. entwickelte Collision Free Speed Model [11] ist ein geschwindigkeitsbasiertes Modell, welches von Natur aus kollisionsfrei ist. Die Idee des Modells liegt darin, die Bewegungsgeschwindigkeit und die Bewegungsrichtung separat zu behandeln.

Die Geschwindigkeit hängt hierbei von dem Abstand zu dem vorderen Fußgänger ab. Die Bewegungsgeschwindigkeit wird durch eine Optimal-Velocity-Funktion geregelt. Der Agent bewegt sich in Richtung seines Ziels, wenn der Weg vor ihm frei ist. Nähert er sich einem vorderen Agenten, wird seine Geschwindigkeit reduziert. Im schlimmsten Fall wird die Geschwindigkeit auf null gesetzt, wenn sich zwei Agenten zu nahe kommen. Dies vermeidet jegliche Kollisionen.

Die Bewegungsrichtung ist ein Kompromiss. Jeder Nachbaragent schiebt den Agenten leicht von sich weg, wobei dieser Effekt mit abnehmendem Abstand stärker wird. Grundsätzlich orientiert sich der Agent aber an der direkten Linie zum Ziel. Die endgültige Richtung ist eine Kombination aus der Zielorientierung und den Ausweichmanövern.

### 3.2.2 Modellklassifizierung

Das Collision Free Speed Model wird als geschwindigkeitsbasiertes Modell (Velocity-based) angesehen. Das Modell definiert direkt die Geschwindigkeit des Agenten in jedem Zeitschritt, ohne von Kräften beeinflusst zu werden.

### 3.2.3 Diskussion und Anwendungsbereiche

In dem Paper: Pedestrian Dynamics - From Empirical Results to Modeling [8] wird ebenfalls, nicht für dieses explizite Modell, aber für generelle geschwindigkeitsbasierte Modelle, beschrieben, dass im Gegensatz zu beschleunigungsbasierten Modellen keine Verzögerungsmechanismen, außerhalb der Annäherung von Agenten, existieren.

## 3.3 Optimal Steps Model

### 3.3.1 Grundidee und Funktionsweise

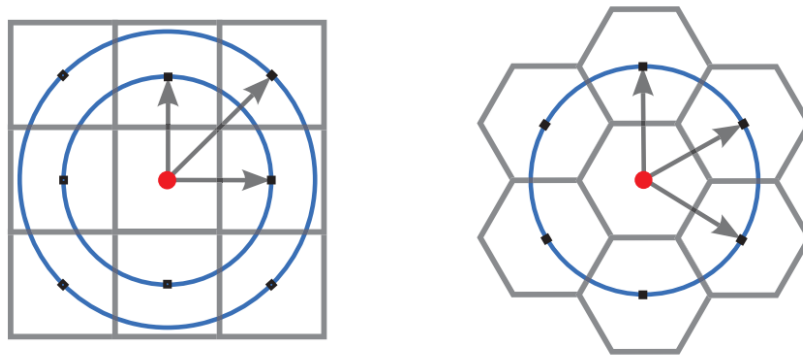


Abbildung 3.1: Eine Veranschaulichung der benachbarten Felder eines Agenten in dem Optimal Steps Model [10]

Das von Seitz und Köster entwickelte Optimal Steps Model [10] bildet die schrittweisen Bewegungen von Menschen nach. Anstatt eine kontinuierliche Bewegung zu berechnen, wählt ein Agent in einem Zeitschritt immer den nächsten optimalen Schritt aus. Dazu werden mehrere mögliche Zielpunkte um seine aktuelle Position bewertet.

Die Entscheidungsfindung des Modells wird in drei logische Schritte unterteilt:

1. Definition eines potenziellen Feldes: Die Umgebung eines Agenten wird als ein Umfeld von potenziellen Feldern definiert. Felder, die näher in Richtung des Ziels liegen, haben ein niedriges Potenzial und werden als attraktiv gewertet. Positionen, die anderen Agenten oder Hindernissen zu nahe sind, haben ein hohes Potenzial und sind somit abstoßend.

2. Identifikation möglicher Schritte: Der Agent betrachtet eine feste Anzahl von möglichen Zielpunkten für seinen nächsten Schritt. Diese Punkte liegen auf einem Kreis um ihn herum.
3. Entscheidung: Der Agent berechnet für jeden dieser möglichen Zielpunkte den Gesamtwert des Potenzials. Er wählt den Punkt mit dem niedrigsten Potenzial als sein nächstes Ziel, und bildet somit seinen Weg in Richtung des Ziels.

#### **3.3.2 Modellklassifizierung**

Das Optimal Steps Model ist als entscheidungsbasiertes Modell (Decision-based) einzuordnen. Die Bewegungen basieren auf einem Optimierungs- und Entscheidungsprozess in jedem Zeitschritt, mit dem Ziel, die nächste Position zu bestimmen.

#### **3.3.3 Diskussion**

Ein Vorteil [8] des Optimal Steps Models besteht darin, dass der Ansatz natürliche Trajektorien erlaubt, ohne Differentialgleichungen oder komplexes Steuerungsverhalten einzuführen.

# 4 Entwurf der Experimentierplattform

## 4.1 Anforderungsanalyse und Zielsetzung

In dem folgenden Teil der Arbeit werden die funktionalen und nichtfunktionalen Anforderungen der Arbeit definiert.

### 4.1.1 Funktionale Anforderungen

**FA-001: Modellierung einer räumlichen Umgebung aus Geodaten** Das System muss in der Lage sein, eine komplexe, räumliche Umgebung auf Basis von Vektordaten im GeoJSON-Format zu laden und zu repräsentieren. Der Weihnachtsmarkt soll nach dem originalen Lageplan erstellt werden.

**FA-002: Konfigurierbarkeit des Simulationslaufs** Die zentralen Parameter einer Simulation müssen über eine externe Konfigurationsdatei `config.json` gesteuert werden. Dies schließt mindestens den globalen Start- und Endzeitpunkt der Simulation sowie die Dauer eines Zeitschritts ein.

**FA-003: Parametrisierung von Agentenattributen** Individuelle Eigenschaften von Agenten, wie beispielsweise die bevorzugte Geschwindigkeit `PreferredSpeed` muss zur Laufzeit über die Konfigurationsdatei gesetzt werden können.

**FA-004: Unterstützung austauschbarer Verhaltensmodelle** Das System muss die Simulation von Agenten mit unterschiedlichen, modularen Verhaltenslogiken ermöglichen. Die Auswahl des zu verwendenden Agentenmodells (wie `OptimalStepsMarketTraveler`) für einen Simulationslauf muss extern über die Konfigurationsdatei steuerbar sein, ohne dass eine Veränderung im Quellcode nötig ist.

**FA-005: Autonome Agentenbewegung** Agenten müssen die Fähigkeit besitzen, selbstständig von ihrem Startpunkt zu einem zugewiesenen Zielpunkt innerhalb der Umgebung zu bewegen. Dies gilt sowohl für die Bewegungen außerhalb, als auch innerhalb des Weihnachtsmarktes.

**FA-006: Bewegungsmuster nach Verhaltensmodell** Die Agenten müssen sich auf dem `MarketLayer` mit dem Bewegungsmuster des Verhaltensmodells fortbewegen, welches für sie definiert wurde.

**FA-007: Zielgerichtetes Verhalten auf dem Markt** Der Agent bewegt sich nicht nur, sondern folgt aktiv einem Ziel. Das Ziel ist es, eine zufällige `MarketStall`-Entität zu wählen, und sich zu dieser zu bewegen.

**FA-008: Verlassen des Weihnachtsmarktes** Die Agenten verlassen nach einer bestimmten Zeit den Weihnachtsmarkt und begeben sich auf den Weg nach Hause.

**FA-009: Zustandswechsel des Agenten** Ein Agent muss verschiedene interne Zustände annehmen können, die sein Verhalten steuern. Folgende Zustände müssen implementiert sein: `WalkingToMarket`, `OnMarket`, `WalkingHome`. Der Übergang dieser Zustände muss durch Ereignisse gesteuert sein. Das Betreten des Weihnachtsmarktes bewirkt, dass sich der Agent in dem Zustand `OnMarket` befindet.

### 4.1.2 Nichtfunktionale Anforderungen

**NFA-001: 100-Agenten-Regel** Trotz der komplexen Berechnungen einiger Verhaltensmodelle müssen zu jedem Zeitpunkt immer mindestens 100 Agenten existieren. Das Programm darf bei der Ausführung der Simulation nicht abstürzen.

**NFA-002: Erweiterbarkeit** Die Softwarearchitektur muss es ermöglichen, neue Agenten-Verhaltensmodelle (eine neue `...MarketTraveler`-Klasse) hinzuzufügen, ohne bestehende Logik oder die Kernlogik des Programms grundlegend ändern zu müssen.

## 4.2 Konzeptionelle Modellierung der Domäne

Zwei grundsätzliche, konzeptionelle Modellierungen bieten sich hier an, um einen ersten Ansatz an die Visualisierung des Modells zu ermöglichen, das Domänenmodell und das fachliche Datenmodell.

### 4.2.1 Das Domänenmodell

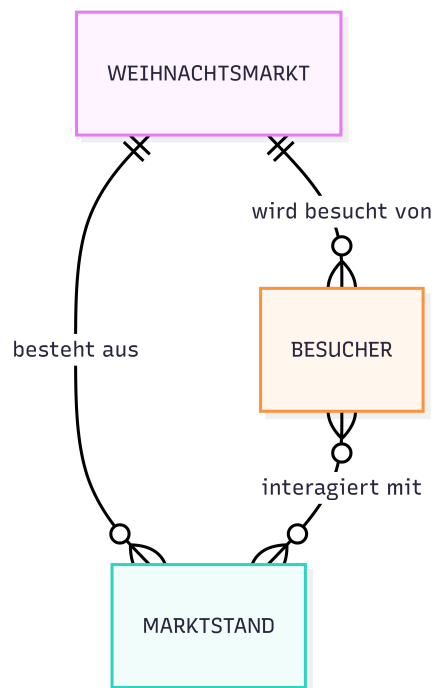


Abbildung 4.1: Das Domänenmodell für die Plattform

Das Domänenmodell ist der erste Ansatz, um die Zusammenhänge der verschiedenen Komponenten, ganz losgelöst von der Technik, darzustellen. Konzeptionell gibt es einen Weihnachtsmarkt, auf dem verschiedene Marktstände stehen. Besucher gibt es auf diesem Weihnachtsmarkt ebenfalls. Sie besuchen den Weihnachtsmarkt und haben die Aufgabe, die Marktstände zu besuchen.

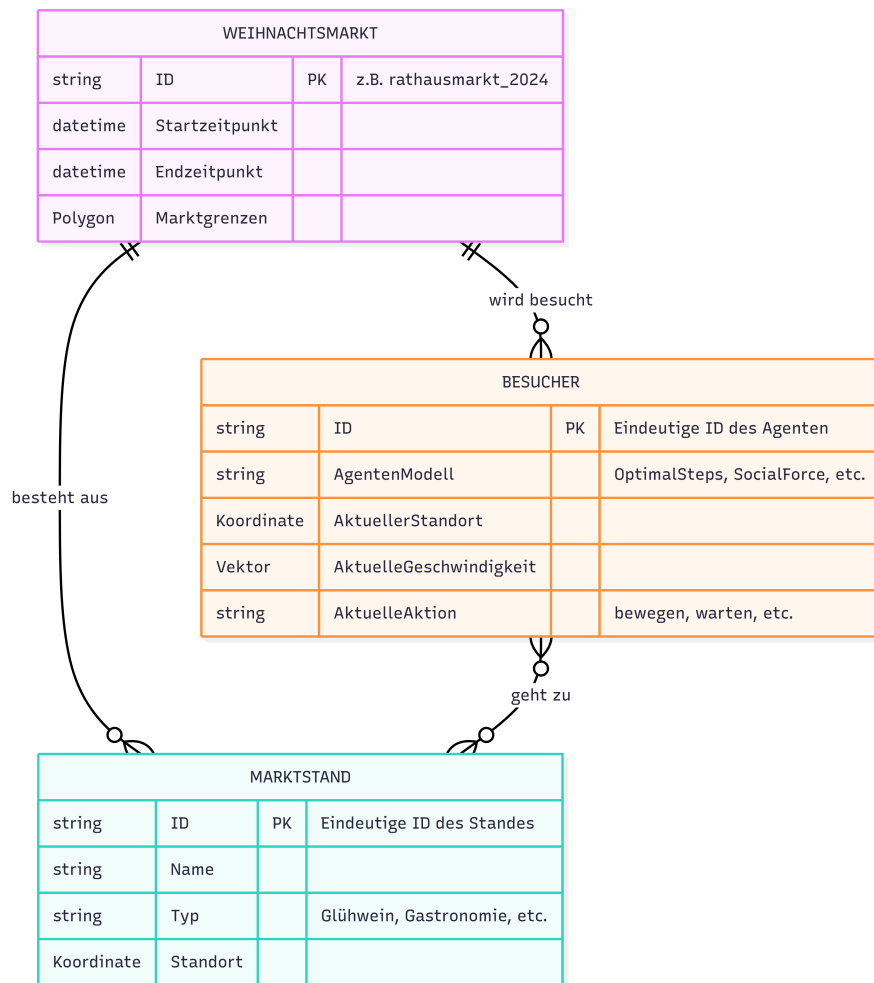


Abbildung 4.2: Das fachliche Datenmodell für die Plattform

### 4.2.2 Das fachliche Datenmodell

Das fachliche Datenmodell hingegen ist etwas detaillierter. Die Zusammenhänge zwischen den einzelnen Komponenten gelten genauso wie bei dem Domänenmodell. Allerdings kommen nun auch Attribute hinzu, welche bereits auf die Strategie hinweisen, wie die Agentenmodelle definiert werden.

Der Weihnachtsmarkt hat einen Startzeitpunkt, und einen Endzeitpunkt, an dem dieser besuchbar ist. Die Marktgrenzen werden mit einem Polygon definiert. Dieses Polygon ist die exakte Marktfläche, die auch auf dem Hamburger Weihnachtsmarkt verwendet wurde.

Die Besucher sollen zwischen verschiedenen Agentenmodellen wählen können. Zusätzlich werden für die Berechnungen der einzelnen Modelle ebenfalls Attribute wie der aktuelle Standort, oder die aktuelle Geschwindigkeit benötigt. Der Agent leitet daraus eine Aktion ab, die er pro Tick durchführt.

Die Marktstände werden über eine ID identifiziert. Ein Marktstand hat einen Namen, und einen Typ. Er kann ein Glühweinstand sein, ein Gastronomiestand, oder ein Verkaufsstand. Andere Standtypen gibt es nicht. Der Standort der jeweiligen Marktstände muss ebenfalls angegeben werden, damit die Agenten wissen, was die jeweiligen Zielkoordinaten sind.

### 4.3 Architektorentwurf der Experimentierplattform

#### 4.3.1 Struktur der Layer

Für die Experimentierplattform werden vier verschiedene Layer genutzt:

1. `SpatialGraphMediatorLayer`: Dieses Layer hat die Aufgabe, dem Agenten den Weg bis zum Weihnachtsmarkt zu ermöglichen. Außerhalb von dem Weihnachtsmarkt bewegt sich der Agent nämlich auf dem `SpatialGraphMediatorLayer`. Das Straßennetz wird aus einer GeoJSON-Datei geladen.
2. `AgentSchedulerLayer`: Dieses Layer hat die Verantwortung, die Agenten zu erzeugen. Gemäß der CSV-Input-Datei sollen die Agenten erzeugt werden.
3. `MarketLayer`: Dieses Layer verwaltet die `MarketStall`-Entitäten und stellt sie den Agenten als potenzielle Ziele zur Verfügung.
4. `MarketTravelerLayer`: Dieses Layer ist der Container für alle Marktbesucher-Agenten.

#### 4.3.2 Entwurf der Agenten und Entitäten

Für die Experimentierplattform wird eine Agentenklasse und eine Entitätsklasse genutzt.

1. `MarketStall` (Entität): Die Marktstände werden bei dem Start der Simulation erzeugt und bewegen sich nicht. Die jeweilige Entität hat eine eigene feste Position und ist ein bestimmter Stand-Typ.
2. `MarketTraveler` (Agent): Die Marktbesucher werden als aktive Agenten entworfen, die von der `Traveler`-Klasse erben. Sie begeben sich auf dem schnellsten Weg auf dem Straßennetz zum Weihnachtsmarkt. Danach wechseln die Agenten das Layer und betreten den Weihnachtsmarkt. Anschließend suchen sich die Agenten Marktstände als Ziele aus. Die Bewegung auf dem Weihnachtsmarkt ist komplett davon abhängig, welches Agentenmodell (`OptimalStepsMovementModel`, `SocialForcesMovementModel`, `CollisionFreeSpeedMovementModel`) sie verwenden.

## 4.4 Grundlage für die Implementierung

### 4.4.1 Definition der Simulationsszenarien

Hiermit werden die Spielregeln der Experimente in dieser Experimentierplattform festgelegt:

Diese Parameter bleiben konstant und müssen nicht geändert werden:

- Das allgemeine Layout der Simulation.
- Die Dauer der Simulation.
- Die Spawn-Raten aus der `human_traveler_rathausmarkt.csv`.
- Sonstige Komponenten der Simulationsumgebung.
- usw...

Der einzige Parameter, der variieren soll, sobald das Bewegungsmodell von dem Benutzer geändert wird, ist:

- Der verwendete Agententyp (`OptimalSteps...` vs. `SocialForce...`) in der Konfigurationsdatei `config.json` und der Programmdatei `Program.cs`.

# 5 Technische Implementierung

## 5.1 Technologiestack und Entwicklungsumgebung

- Programmiersprache: C# (.NET Framework Version).
- Simulationsframework: MARS mit der SmartOpenHamburg-Bibliothek.
- Entwicklungsumgebung: JetBrains Rider 2025.2.2.1.
- Datenformate: `.geojson` für Geodaten, `.csv` für tabellarische Daten, `.json` für die Konfiguration der Simulation.

## 5.2 Softwarearchitektur und Klassenstruktur

### 5.2.1 Das Designmodell

Das Designmodell (siehe Abbildung 5.1) zeigt auf, dass die Struktur der Experimentierplattform strikt dem MARS-Framework folgt. Die Hauptkomponenten sind sofort erkennbar:

- Layer: `MarketLayer`, `MarketTravelerLayer`.
- Entity: `MarketStall`.
- Agent: `MarketTraveler` und seine Ableitungen, die für jedes Modell definiert sind, beispielsweise `OptimalStepsMarketTraveler`.

Die Klasse `MarketTraveler` ist abstrakt und definiert die gemeinsame Logik für alle Besucher (Zustandslogik, Ankunft beim Markt, Verlassen des Marktes), während die konkrete Bewegung `CalculateNextMovementStep()` den Unterklassen überlassen wird.

Der ausschlaggebende Punkt ist aber die Nutzung des Strategy Patterns.

- Das Interface `IPedestrianMovementModel` definiert einen Vertrag für einen Bewegungsalgorithmus.
- Klassen wie `SocialForcesMovementModel` und `CollisionFreeSpeedMovementModel`, `OptimalStepsMovementModel` sind konkrete Strategien, die diesen Auftrag erfüllen.
- Die Agenten-Klassen `SocialForceMarketTraveler`, `CollisionFreeMarketTraveler`, `OptimalStepsMarketTraveler` haben eine Instanz einer dieser Strategien.
- In der `CalculateNextMovementStep()`-Methode rufen sie einfach die Methode der Strategie-Instanz auf.
- Somit ist sichergestellt, dass neue Bewegungsmodelle einfach hinzugefügt werden können, indem nur eine Strategie-Klasse geschrieben werden muss, ohne die Agenten selbst zu ändern.

Zusätzlich erzielt die Architektur eine strikte Trennung von Verantwortlichkeiten (Separation of Concerns).

`MarketTraveler` ist für die strategische Ebene verantwortlich. Sie bestimmt den Zustand des Agenten, bestimmt das nächste Ziel, oder entscheidet sich gegebenenfalls den Markt zu verlassen. `IPedestrianMovementModel` hingegen ist ausschließlich für die operative Ebene zuständig. Es definiert die Berechnung des nächsten Schritts, wobei Kollisionen, je nach Modell, auf unterschiedliche Weise vermieden und das Ziel entsprechend angesteuert wird.

Auch wird in der `MarketTraveler`-Klasse explizit das Template Method Pattern genutzt. Es überprüft hierbei den globalen Zustand des Agenten und führt daraufhin explizit Logik aus. Wenn sich der Agent beispielsweise in `VisitorState.WalkingHome` befindet, wird die Methode `TickWalkingHome()` aufgerufen.

Ein Ausschnitt des Template Method Pattern ist hier angegeben:

---

```
1 MarketTraveler.cs:
2
3 public override void Tick()
4 {
5     switch (_state)
6     {
7         case VisitorState.WalkingToMarket:
8             TickWalkingToMarket();
9             break;
10        case VisitorState.OnMarket:
11            SimulateFreeMovement();
12            break;
13        case VisitorState.WalkingHome:
14            TickWalkingHome();
15            break;
16    }
17 }
```

---

Agenten haben drei verschiedene Zustände, und sie unternehmen verschiedene Dinge, je nachdem, in welchem Zustand sich die Agenten befinden. Hiermit möchte ich unterstreichen, dass alle Klassen, die von `MarketTraveler` erben, die gleiche Funktionsweise haben, und sich nur darin unterscheiden, wie das Bewegungsmuster der jeweiligen implementierenden Klasse aufgebaut ist:

- `VisitorState.WalkingToMarket`: Der Agent ist momentan noch auf dem Straßennetz und begibt sich zum Weihnachtsmarkt. Dafür nutzt er die Methode `TickWalkingToMarket()`.
- `VisitorState.OnMarket`: Der Agent bewegt sich nun auf dem Markt. Die Template-Methode `SimulateFreeMovement()` wird in die jeweilige implementierende Klasse delegiert. So hat jedes Modell seine eigene Methode für die Bewegung.
- `VisitorState.WalkingHome`: Der Agent hat nun den Weihnachtsmarkt verlassen und geht nach Hause. Er nutzt seinen Spawnpunkt als Zielposition und sieht diesen als sein Zuhause an.

### 5.2.2 Programmablauf und Initialisierung

Die `Program.cs`-Klasse und die darin enthaltene `Main()`-Methode sind der Einstiegspunkt der Anwendung. Hier wird zuerst das `ModelDescription`-Objekt instanziiert und mit den spezifischen Layern und Agenten des Modells befüllt. Beim Ändern von Bewegungsmodellen muss sichergestellt werden, dass sowohl der `AddLayer<>`, als auch der `AddAgent<>` Befehl für das jeweilige Bewegungsmodell verändert wird. In der `config.json`-Datei muss ebenfalls der Name des Agenten beim Wechseln von Bewegungsmodellen, angepasst werden.

---

```
1 Program.cs:
2
3     description.AddLayer<AgentSchedulerLayer<
4         OptimalStepsMarketTraveler, MarketTravelerLayer>>("
5         MarketTravelerSchedulerLayer");
6     description.AddAgent<OptimalStepsMarketTraveler,
7         MarketTravelerLayer>();
```

---

In der `config.json`-Datei muss ebenfalls der Name des Agenten bei dem Wechseln von Bewegungsmodellen, angepasst werden.

---

```
1 config.json:
2
3     "agents": [
4     {
5         "name": "OptimalStepsMarketTraveler",
6         ...
```

---

## 5.3 Implementierung der Kernkomponenten

In dem folgenden Abschnitt geht es um die Implementierung der Kernkomponenten der Experimentierplattform.

### 5.3.1 Implementierung der Agentenmodelle

Alle Bewegungsmodelle nutzen das Interface `IPedestrianMovementModel` und implementieren die Methode `CalculateNextPosition()`. In dieser Methode wird definiert, wie sich Agenten in einem Tick auf dem Marktplatz fortbewegen. Hier werden die mathematischen Formulierungen der einzelnen Modelle implementiert und sind somit komplett von sonstiger Verantwortlichkeit getrennt (Separation of Concerns). Die implementierenden Klassen, wie beispielsweise `OptimalStepsMovementModel` definieren nur diese Methode. Alles andere, wie die Zielfindung, ist in `MarketTraveler` definiert.

---

```
1 IPedestrianMovementModel.cs:
2
3     public Position CalculateNextPosition(
4         MarketTraveler traveler,
5         Position targetPosition,
6         IEnumerable<MarketTraveler> nearbyTravelers,
7         List<(double lon, double lat)> marketBoundary,
8         double dt)
```

---

wobei

- `traveler`: Der Agent, der sich bewegt.
- `targetPosition`: Die aktuelle Zielposition (beispielsweise ein Marktstand).
- `nearbyTravelers`: Andere Agenten in der Nähe (für potenzielle Kollisionsvermeidungen).
- `marketBoundary`: Das Polygon, das die Grenzen des Marktes definiert.
- `dt`: Die Dauer des Simulationsticks (in Sekunden).

### 5.3.2 Anpassung des Travelers

Der `MarketTraveler` übernimmt die restlichen Aufgaben, die ein implementiertes Modell ausführt. Dazu gehört zum Beispiel die Bewegung zum Weihnachtsmarkt (`TickWalkingToMarket`) und das Betreten des Marktes (`EnterMarket`). Hier sind alle Methoden aufgelistet, die

der `MarketTraveler` bereitstellt. Hilfsmethoden, welche nicht in anderen Klassen genutzt werden, sind hier nicht erwähnt.

---

```
1 MarketTraveler.cs:
2
3     protected void SimulateFreeMovement(); <- Template Methode
4     protected abstract CalculateNextMovementStep(); <- wird an das zu
        implementierende Modell weitergegeben
5
6     private void TickWalkingToMarket();
7     private void EnterMarket();
8     protected void FinishMarketVisit();
9     private void TickWalkingHome();
10    private void ArriveHome();
11    private void CheckForNearbyStalls();
12    private void ChooseNewTargetStall();
13    ...
```

---

Die Methodennamen erklären das Verhalten des Marktbesuchers bereits.

### 5.3.3 Verarbeitung des Inputs

Erwähnenswert ist, dass in der Konfigurationsdatei `config.json` Folgendes untergebracht werden muss:

- Der Start- und Endzeitpunkt der Simulation.
- Das `SpatialGraphMediatorLayer` muss ebenfalls mitangegeben werden. Es beschreibt das Straßennetz und somit die Wege vor und nach dem Weihnachtsmarktbesuch.
- Das `MarketLayer`, welches die Marktstände enthält, sowie die Ecken des Marktes mit zusätzlichen Koordinatenpaaren für das Format Longitude / Latitude.
- Das `MarketTravelerSchedulerLayer`, welches die Agenten erzeugen lässt.

### 5.3.4 Modellieren des Weihnachtsmarktes

Der Weihnachtsmarkt hat drei verschiedene Standtypen: Glühweinstände, Gastronomiestände und Verkaufsstände. Jeder einzelne Stand aus dem Lageplan (siehe Abbildung 5.2) wurde exakt in `market_stalls.geojson` übernommen. Insgesamt wurden 93 Marktstände aus dem Lageplan in die `geojson`-Datei übernommen und in die Experimentierplattform übergeben.

---

```
1 market_stalls.geojson:
2
3   {
4     "type": "Feature",
5     "properties": {
6       "type": 2,
7       "name": "Verkaufsstand A"
8     },
9     "geometry": {
10      "coordinates": [
11        9.993347231235816,
12        53.55062051455096
13      ],
14      "type": "Point"
15    },
16    "id": 1
17  },
18  ...
```

---





Abbildung 5.2: Lageplan des Weihnachtsmarktes am Hamburger Rathaus [1]

# 6 Ergebnisse und Diskussion

## 6.1 Validierung der Anforderungen

Im Folgenden werden die funktionalen und nichtfunktionalen Anforderungen aufgegriffen und validiert.

**FA-001: Modellierung einer räumlichen Umgebung aus Geodaten** Die Verwendung des `SpatialGraphMediatorLayer` erfüllt diese Anforderung, da dieses Layer in der Lage ist, GeoJSON-Dateien zu parsen und als Navigationsgraph zu nutzen. Zudem wurde in 5.3.4 die Implementierung der Standtypen anhand des Lageplans definiert.

**FA-002: Konfigurierbarkeit des Simulationslaufs Erfüllt.** Die zentralen Simulationsparameter werden über die Konfigurationsdatei `config.json` gesteuert. So kann der Startzeitpunkt der Simulation, der Endzeitpunkt der Simulation und die Geschwindigkeit der Simulation festgelegt werden.

**FA-003: Parametrisierung von Agentenattributen Erfüllt.** Individuelle Agenteneigenschaften können über die Konfigurationsdatei `config.json` konfiguriert werden. Dies wird beispielsweise durch den Parameter `PreferredSpeed` veranschaulicht.

**FA-004: Unterstützung austauschbarer Verhaltensmodelle Erfüllt.** Die Architektur basiert auf dem Strategy-Pattern (Kapitel 5.2.1), in dem die Bewegungslogik in separate Klassen aufgeteilt wird. Klassen, die das Interface `IPedestrianMovementModel` implementieren können hiervon Gebrauch machen.

**FA-005: Autonome Agentenbewegung Erfüllt.** Außerhalb des Marktes benutzen Agenten die multimodale Wegfindung. Innerhalb des Marktes bewegen sich Agenten mit ihrer jeweiligen Bewegungslogik zu den Marktständen.

**FA-006: Bewegungsmuster nach Verhaltensmodell Erfüllt.** Die Implementierung stellt sicher, dass jeder Agententyp sein eigenes Bewegungsmuster nutzt. Die Template-Methode `SimulateFreeMovement()` ruft die abstrakte Methode `CalculateNextMovementStep` auf. Jedes Verhaltensmodell implementiert so sein eigenes Bewegungsmuster, da sie Unterklassen von `MarketTraveler` sind.

**FA-007: Zielgerichtetes Verhalten auf dem Markt Erfüllt.** Agenten bewegen sich nicht zufällig auf dem Markt, sie folgen Zielen. Bei dem Eintritt in den Markt wird durch `ChooseNewMarketStall()` festgelegt, wohin das aktuelle Ziel gelegt wird. Der Agent bewegt sich dann auf die Position dieses Standes zu.

**FA-008: Verlassen des Weihnachtsmarktes Erfüllt.** Für die Vergleichbarkeit der Agentenmodelle wird das Verlassen durch die `LeaveProbability` gesteuert. Mit jedem Tick hat der Agent die Möglichkeit, sich zu entscheiden, den Markt zu verlassen. Daraufhin wird der Zustand zu `WalkingHome` geändert.

**FA-009: Zustandswechsel des Agenten Erfüllt.** Die Agentenlogik ist durch eine klar definierte Zustandsmaschine in `MarketTraveler` gesteuert, die auf `VisitorState` basiert. Diese besitzt die Zustände `WalkingToMarket`, `OnMarket` und `WalkingHome`. Die `Tick()`-Methode implementiert das Zustandsverhalten, wie in Kapitel 5.2.1 beschrieben. Ereignisse wie das Betreten des Weihnachtsmarktes lösen die gegebenen Zustandsübergänge aus.

**NFA-001: 100-Agenten-Regel Erfüllt.** Die Simulation aller ausgeführten Experimente, deren Messergebnisse in Kapitel 6.2 und im Anhang dargestellt sind, lief mit einer weitaus höheren Anzahl an Agenten. Die Simulation lief dabei stabil wies keine Abstürze auf.

**NFA-002: Erweiterbarkeit Erfüllt.** Die gewählte Softwarearchitektur (Kapitel 5.2), insbesondere die Entkopplung der Bewegungslogik durch das Strategy Pattern, gewährleistet eine hohe Erweiterbarkeit. Ein neues Verhaltensmodell kann durch die Implementierung des `IPedestrianMovementModel`-Interfaces und die Erstellung einer neuen `MarketTraveler`-Subklasse hinzugefügt werden, ohne die Kernlogik der bestehenden Implementierung zu beeinträchtigen.

## 6.2 Messergebnisse

Alle Agenten haben für die Messung die gleichen Randbedingungen.

1. Das Experiment hat folgenden Startzeitpunkt: 2024-12-10 um 16:00
2. Das Experiment hat folgenden Endzeitpunkt: 2024-12-10 um 18:00
3. Die Agenten bewegen sich mit einer Basisgeschwindigkeit (`PreferredSpeed`) von 1,4.
4. Alle Agenten werden aus der gleichen `human_traveler_rathausmarkt.csv` erzeugt. Dies garantiert, dass für jede Simulation genau die gleiche Menge an Agenten erzeugt wird mit genau den gleichen Start- und Zielkoordinaten.

Die Messergebnisse des Experimentes sind nun in mehreren Tabellen dargestellt.

Tabelle 6.1: Systematischer Vergleich der Leistungsmetriken der Agentenmodelle

<b>Metrik</b>	<b>OptimalSteps</b>	<b>CollisionFreeSpeed</b>	<b>SocialForce</b>
Gesamte Standbesuche	3362	2463	473
Durchschnittl. Verweildauer (s)	105.37	105.40	102.75
Kürzeste Verweildauer (s)	1	1	1
Längste Verweildauer (s)	799	797	761

In Tabelle 6.1 sind die wichtigsten Metriken dargestellt. Unter anderem sind die gesamten Standbesuche, die durchschnittliche Verweildauer pro Stand, die kürzeste Verweildauer und die längste Verweildauer pro Modell angegeben.

In Tabelle 1, 2 und 3 (siehe Anhang) werden für alle Modelle die detaillierten Besucherzahlen pro Stand angegeben. Sie sind absteigend nach der Anzahl der Besucher geordnet.

Die Messergebnisse werden in der Diskussion (6.4) hinterfragt.

### 6.3 Visualisierung des Bewegungsverhaltens

Das Bewegungsverhalten der verschiedenen Agentenmodelle wurde mit Hilfe von Kepler visualisiert. Kepler ist eine Open-Source Webanwendung, welche Analysen für agentenbasierte Simulationen in einem georeferenzierten Raum ermöglicht.

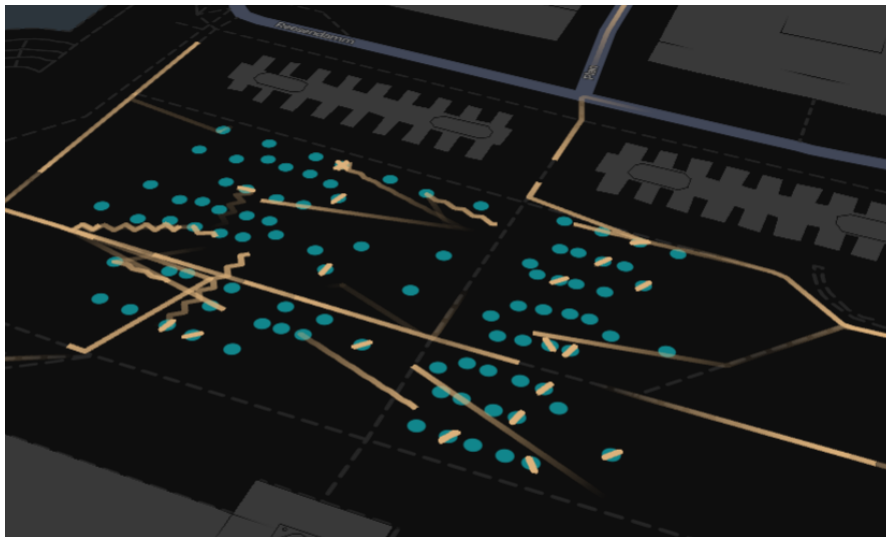


Abbildung 6.1: Visualisierung des Social Force Models in der Experimentierplattform mittels Kepler

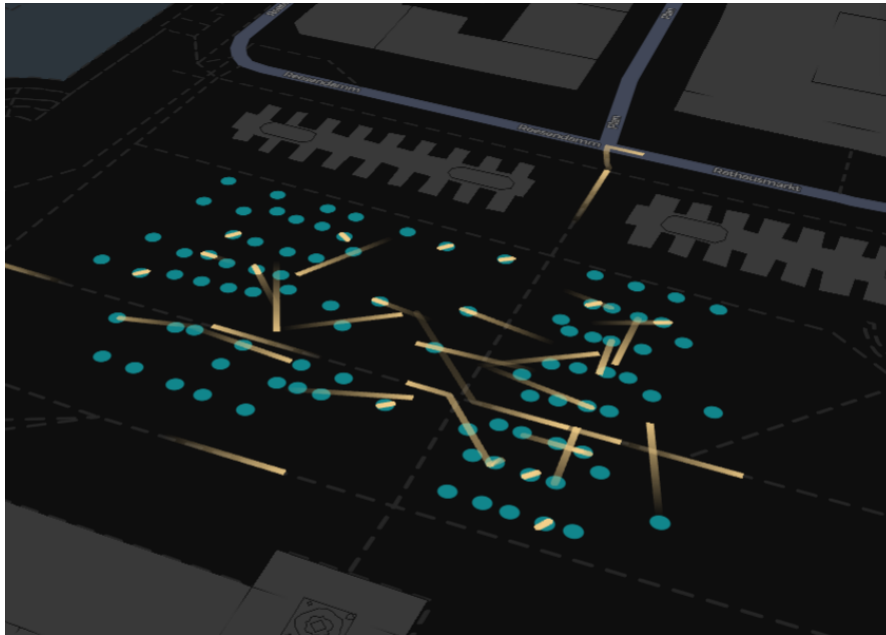


Abbildung 6.2: Visualisierung des Collision Free Speed Models in der Experimentierplattform mittels Kepler

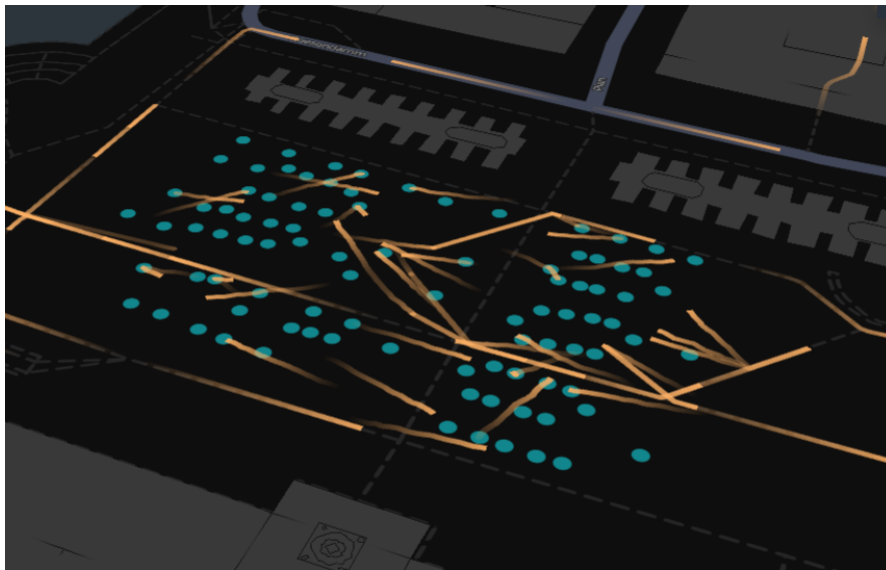


Abbildung 6.3: Visualisierung des Optimal Steps Models in der Experimentierplattform mittels Kepler

## 6.4 Diskussion

Die durchgeführten Experimente und die daraus resultierenden Messergebnisse ermöglichen einen quantitativen Vergleich der implementierten Verhaltensmodelle. Die Unterschiede in den Leistungsmetriken sind signifikant und lassen sich auf die Stärken und die Schwächen der Modelle zurückführen.

Das `OptimalStepsMarketTraveler`-Modell zeigt mit 3362 Standbesuchen die mit Abstand höchste Effizienz. Die entscheidungsbasierte Logik erlaubt es den Agenten, proaktiv und mit Kurvenbewegungen zu navigieren und somit Hindernissen aus dem Weg zu gehen. Die weichen Kurvenbewegungen ergeben hierbei Sinn, denn das `OptimalSteps` Model berechnet mit jedem `Tick()` den idealen nächsten Schritt.

Das `CollisionFreeSpeedMarketTraveler`-Modell stellt sich mit 2463 Standbesuchen zwischen die beiden anderen Modelle. Der reaktive Ansatz, den das `CollisionFreeSpeed` Model verfolgt verhindert zwar Kollisionen (wie auch in der Visualisierung anschaulich), aber hat trotzdem eine geringere Gesamteffizienz im Vergleich zu dem `OptimalSteps` Model.

Der `SocialForceMarketTraveler` hingegen schneidet mit 473 besuchten Ständen deutlich schlechter ab. In der Visualisierung des Bewegungsverhaltens des `SocialForce` Models sieht man einige Agenten, die von den Kräften zur Seite gedrängt werden und somit ihr Ziel später erreichen. Dies ist ein klares Indiz dafür, dass in dem engen Marktbereich mit vielen Agenten das `SocialForce` Model zu Schwierigkeiten gelangt.

Zusammenfassend belegen die Ergebnisse, dass die Wahl des Modells das Simulationsergebnis maßgeblich beeinflusst. Die identische Verweildauer an Marktständen, dafür aber drastisch unterschiedliche Zielerreichung von Marktständen weist auf einen deutlichen Effizienzunterschied zwischen den Modellen. Für dieses spezifische Szenario erweist sich der entscheidungsbasierte Ansatz des `OptimalSteps` Modells als deutlich überlegen. .

# 7 Fazit

## 7.1 Schlussfolgerung

Die Experimentierplattform bietet sich gut für den Vergleich verschiedener sozialer Agentenmodelle an. Deutliche Unterschiede konnten im Rahmen der Messung zwischen den verschiedenen Agentenmodellen wahrgenommen werden. Die Plattform ermöglicht es ebenfalls, aufgrund der gewählten Architektur, neue Bewegungsmodelle mit wenig Aufwand implementieren zu können.

## 7.2 Ausblick

Die entwickelte Experimentierplattform bietet eine gute Basis für zukünftige Arbeiten. Modelle der Pedestrian Dynamics lassen sich einfach einbauen. Aber der Aufbau dieser Experimentierplattform ermöglicht es auch, viele andere Agentenmodelle darzustellen, sofern sie ein Bewegungsmuster definieren. Zukünftig wäre auch die Implementierung von LLM-basierten Agenten in der Experimentierplattform denkbar, ein Ansatz, der aktuell in der MARS-Forschungsgruppe verfolgt wird.

# Literaturverzeichnis

- [1] *Lageplan des Weihnachtsmarktes am Hamburger Rathausmarkt*. – URL <https://www.hamburger-weihnachtsmarkt.com/>. – Zugriff am: 4. Juli 2025
- [2] FERBER, Jacques ; WEISS, Gerhard: *Multi-Agent Systems: an introduction to distributed artificial intelligence*. Bd. 1. Addison-Wesley Reading, 1999
- [3] HELBING, Dirk: Traffic and related self-driven many-particle systems. In: *Reviews of Modern Physics* 73 (2001), Dezember, Nr. 4, S. 1067–1141. – URL <http://dx.doi.org/10.1103/RevModPhys.73.1067>. – ISSN 1539-0756
- [4] HELBING, Dirk ; MOLNAR, Peter: Social Force Model for Pedestrian Dynamics. In: *Physical Review E* 51 (1998), 05
- [5] MARS GROUP: *MARS Dokumentation*. – URL <https://www.mars-group.org/docs/category/development>. – Zugriff am: 27. September 2025
- [6] MARS GROUP: *MARS Dokumentation: Agenten*. – URL <https://www.mars-group.org/docs/tutorial/development/agent>. – Zugriff am: 27. September 2025
- [7] MARS GROUP: *Smart Open Hamburg*. – URL <https://www.mars-group.org/projects/smartopenhamburg>. – Zugriff am: 11. September 2025
- [8] SCHADSCHNEIDER, Andreas ; CHRAIBI, Mohcine ; SEYFRIED, Armin ; TORDEUX, Antoine ; ZHANG, Jun: *Pedestrian Dynamics: From Empirical Results to Modeling*. S. 63 – 102. In: *Crowd Dynamics, Volume 1 / Gibelli, Livio (Editor) ; Cham : Springer International Publishing, 2018, Chapter 4 ; ISSN: 2164-3679=2164-3725 ; ISBN: 978-3-030-05128-0=978-3-030-05129-7 ; doi:10.1007/978-3-030-05129-7*. Cham : Springer International Publishing, 2018 (Modeling and Simulation in Science, Engineering and Technology). – URL <https://juser.fz-juelich.de/record/860643>. – ISBN 978-3-030-05128-0 (print)

- [9] SCHADSCHNEIDER, Andreas ; KLINGSCH, Wolfram ; KLÜPFEL, Hubert ; KRETZ, Tobias ; ROGSCH, Christian ; SEYFRIED, Armin: *Evacuation Dynamics: Empirical Results, Modeling and Applications*. S. 3142–3176. In: MEYERS, Robert A. (Hrsg.): *Encyclopedia of Complexity and Systems Science*. New York, NY : Springer New York, 2009. – URL [https://doi.org/10.1007/978-0-387-30440-3\\_187](https://doi.org/10.1007/978-0-387-30440-3_187). – ISBN 978-0-387-30440-3
- [10] SEITZ, Michael J. ; KÖSTER, Gerta: Natural discretization of pedestrian movement in continuous space. In: *Phys. Rev. E* 86 (2012), Oct, S. 046108. – URL <https://link.aps.org/doi/10.1103/PhysRevE.86.046108>
- [11] TORDEUX, Antoine ; CHRAIBI, Mohcine ; SEYFRIED, Armin: *Collision-free speed model for pedestrian dynamics*. 2015. – URL <https://arxiv.org/abs/1512.05597>
- [12] WEYL, Julius ; LENFERS, Ulfa ; CLEMEN, Thomas ; GLAKE, Daniel ; PANSE, Fabian ; RITTER, Norbert: Large-scale Traffic Simulation for Smart City Planning with Mars. In: DURAK, Umut (Hrsg.): *Proceedings of the 2019 Summer Simulation Conference, SummerSim 2019*, ACM, 2019, S. 2:1–2:12
- [13] WOOLDRIDGE, Michael: *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009

Tabelle 1: Detaillierte Besucherzahlen pro Stand für das OptimalSteps-Modell

Stand	Anzahl Besuche
Verkaufsstand V	58
Verkaufsstand BE	53
Gastronomie A	47
Verkaufsstand P	47
Verkaufsstand AV	46
Gastronomie F	46
Verkaufsstand U	45
Gastronomie O	45
Verkaufsstand AL	45
Verkaufsstand BG	45
Verkaufsstand L	44

Tabelle 1 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand BR	44
Verkaufsstand AS	43
Verkaufsstand X	43
Verkaufsstand AO	42
Gastronomie N	42
Gastronomie D	42
Verkaufsstand M	42
Verkaufsstand AZ	42
Verkaufsstand AR	41
Verkaufsstand BH	41
Glühweinstand B	41
Gastronomie E	41
Verkaufsstand BQ	40
Verkaufsstand K	40
Verkaufsstand AJ	40
Gastronomie B	39
Verkaufsstand BP	39
Verkaufsstand AP	39
Verkaufsstand S	39
Verkaufsstand N	39
Verkaufsstand BT	39
Verkaufsstand BM	39
Verkaufsstand AM	39
Verkaufsstand F	38
Verkaufsstand AC	38
Verkaufsstand J	38
Glühweinstand C	38
Verkaufsstand AB	38
Verkaufsstand Y	38
Verkaufsstand AA	37
Gastronomie C	37
Verkaufsstand BA	37
Glühweinstand D	37

Tabelle 1 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Gastronomie J	36
Verkaufsstand AD	36
Verkaufsstand T	36
Gastronomie G	36
Verkaufsstand AQ	35
Verkaufsstand BU	35
Verkaufsstand BN	35
Verkaufsstand W	35
Verkaufsstand E	35
Gastronomie K	35
Gastronomie H	34
Verkaufsstand AU	34
Verkaufsstand C	34
Verkaufsstand G	34
Verkaufsstand BI	34
Verkaufsstand AY	34
Verkaufsstand BC	33
Gastronomie L	33
Verkaufsstand AT	32
Gastronomie I	32
Verkaufsstand AX	32
Verkaufsstand AF	32
Verkaufsstand AG	32
Verkaufsstand AK	31
Verkaufsstand Q	31
Verkaufsstand O	31
Verkaufsstand AE	31
Verkaufsstand A	31
Verkaufsstand BJ	31
Verkaufsstand Z	31
Verkaufsstand BD	31
Glühweinstand A	31
Verkaufsstand R	30

Tabelle 1 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand H	30
Verkaufsstand AW	30
Verkaufsstand BL	30
Gastronomie M	30
Verkaufsstand AH	28
Glühweinstand E	28
Verkaufsstand AI	28
Verkaufsstand D	28
Verkaufsstand BK	27
Verkaufsstand B	27
Verkaufsstand BB	27
Verkaufsstand BO	26
Verkaufsstand AN	26
Verkaufsstand I	26
Verkaufsstand BS	26
Verkaufsstand BF	18

“ “

Tabelle 2: Detaillierte Besucherzahlen pro Stand für das CollisionFreeSpeed-Modell

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand AN	35
Verkaufsstand BP	34
Verkaufsstand AM	34
Gastronomie A	33
Verkaufsstand BM	32
Glühweinstand A	32
Verkaufsstand Q	32
Verkaufsstand AD	32
Verkaufsstand BR	32
Verkaufsstand D	31

Tabelle 2 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Gastronomie F	31
Verkaufsstand BH	31
Verkaufsstand AZ	30
Glühweinstand C	30
Glühweinstand D	30
Verkaufsstand N	30
Verkaufsstand AK	29
Verkaufsstand BC	29
Verkaufsstand P	29
Gastronomie M	29
Verkaufsstand R	29
Verkaufsstand I	29
Verkaufsstand BS	29
Verkaufsstand M	29
Verkaufsstand AF	28
Verkaufsstand AG	28
Verkaufsstand O	28
Verkaufsstand AW	28
Verkaufsstand H	28
Verkaufsstand AC	27
Verkaufsstand BJ	27
Verkaufsstand BI	27
Verkaufsstand AO	27
Gastronomie H	27
Verkaufsstand F	27
Verkaufsstand V	26
Verkaufsstand AR	26
Verkaufsstand J	26
Verkaufsstand Z	26
Verkaufsstand U	26
Verkaufsstand BF	26
Verkaufsstand AQ	26
Verkaufsstand AP	25

Tabelle 2 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand C	25
Gastronomie L	25
Verkaufsstand A	25
Verkaufsstand BQ	25
Verkaufsstand BK	25
Verkaufsstand BU	25
Verkaufsstand W	25
Verkaufsstand Y	25
Gastronomie G	25
Verkaufsstand T	24
Verkaufsstand AX	24
Verkaufsstand G	24
Verkaufsstand X	24
Verkaufsstand BT	24
Verkaufsstand AV	24
Verkaufsstand B	24
Verkaufsstand K	24
Gastronomie J	24
Verkaufsstand AB	23
Verkaufsstand AE	23
Verkaufsstand BE	23
Gastronomie C	23
Glühweinsstand B	22
Verkaufsstand AJ	22
Verkaufsstand BN	21
Gastronomie D	21
Verkaufsstand BD	20
Verkaufsstand BO	20
Verkaufsstand L	20
Verkaufsstand AL	20
Gastronomie B	20
Gastronomie I	19
Verkaufsstand AH	19

Tabelle 2 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Gastronomie N	19
Verkaufsstand BL	19
Verkaufsstand BB	18
Gastronomie K	18
Glühweinstand E	18
Verkaufsstand AY	18
Verkaufsstand AS	17
Verkaufsstand BG	17
Gastronomie E	17
Verkaufsstand S	16
Verkaufsstand AA	16
Verkaufsstand BA	16
Verkaufsstand AT	16
Verkaufsstand E	15
Gastronomie O	15
Verkaufsstand AI	14
Verkaufsstand AU	12

Tabelle 3: Detaillierte Besucherzahlen pro Stand für das SocialForce-Modell

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand AA	16
Gastronomie B	12
Gastronomie N	12
Gastronomie L	11
Verkaufsstand U	11
Verkaufsstand BE	10
Verkaufsstand AK	9
Gastronomie C	9
Verkaufsstand Y	9
Verkaufsstand I	9

Tabelle 3 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand BG	9
Verkaufsstand F	9
Verkaufsstand W	9
Gastronomie M	9
Glühweinstand D	9
Verkaufsstand AE	9
Verkaufsstand AU	9
Verkaufsstand AQ	8
Verkaufsstand AR	8
Gastronomie J	8
Verkaufsstand V	8
Verkaufsstand O	8
Verkaufsstand AS	8
Verkaufsstand BD	8
Gastronomie A	7
Verkaufsstand BQ	7
Verkaufsstand R	7
Verkaufsstand AC	7
Verkaufsstand BB	6
Verkaufsstand BP	6
Verkaufsstand AZ	6
Verkaufsstand A	6
Verkaufsstand BU	6
Gastronomie E	6
Verkaufsstand T	6
Verkaufsstand BA	6
Glühweinstand C	6
Verkaufsstand J	6
Verkaufsstand BK	6
Verkaufsstand G	6
Verkaufsstand BR	6
Verkaufsstand BH	5
Verkaufsstand BJ	5

Tabelle 3 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand BO	5
Verkaufsstand AP	5
Verkaufsstand X	5
Verkaufsstand L	5
Gastronomie I	5
Verkaufsstand AG	5
Verkaufsstand AI	5
Verkaufsstand BS	5
Verkaufsstand BM	5
Verkaufsstand C	5
Verkaufsstand BN	5
Glühweinstand B	4
Verkaufsstand BT	4
Verkaufsstand AW	4
Verkaufsstand AL	4
Verkaufsstand AF	4
Verkaufsstand M	4
Gastronomie G	4
Verkaufsstand Q	4
Verkaufsstand AV	4
Verkaufsstand N	4
Verkaufsstand AH	4
Gastronomie D	4
Verkaufsstand K	3
Verkaufsstand B	3
Gastronomie K	3
Verkaufsstand AB	3
Verkaufsstand D	3
Verkaufsstand AD	3
Glühweinstand E	3
Verkaufsstand Z	3
Verkaufsstand BI	3
Verkaufsstand AN	3

Tabelle 3 – Fortsetzung

<b>Stand</b>	<b>Anzahl Besuche</b>
Verkaufsstand AX	2
Verkaufsstand S	2
Verkaufsstand P	2
Verkaufsstand E	2
Glühweinstand A	2
Verkaufsstand AY	2
Gastronomie H	2
Verkaufsstand AJ	2
Gastronomie F	2
Verkaufsstand AO	1
Verkaufsstand H	1
Verkaufsstand AT	1
Gastronomie O	1
Verkaufsstand BL	1
Verkaufsstand BF	1
Verkaufsstand BC	1

## **Erklärung zur selbständigen Bearbeitung**

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

---

Ort

Datum

Unterschrift im Original