Zellulärer Zustandsautomat

3. Projekt zu Modellierung und Simulation

Daniel Graf, Dimitrie Diez, Arne Schöntag, Peter Müller

Inhaltsverzeichnis

1 Einführung

Im Zuge der ersten Studienarbeit wurde das Laufverhalten von Probanden in der Ebene und auf der Treppe untersucht. Basierend auf den Erkenntnissen bezüglich der individuellen Wunschgeschwindigkeiten werden in dieser Studienarbeit Personenbewegungen in der Ebene simuliert. Mit Hilfe solcher Simulationen können, beispielsweise im Zuge von Gebäudeplanungen unterschiedliche Gebäude- und Raumgestaltungen simuliert und hinsichtlich schnellster Räumungen im Krisenfall optimiert werden.

2 Beschreibung des Modells

Für die Simulation der Personenbewegungen wird ein zellulärer Zustandsautomat implementiert. Jede Zelle kann entweder leer oder durch eine Person oder ein Hindernis besetzt sein. Sie kann außerdem ein Ziel oder eine Quelle enthalten. Ein Hindernis kann von keiner Person betreten werden. Die Personen versuchen im Laufe der Simulation von der Quelle (Startposition) zum Ziel zu gelangen. Hierbei können sie sich in der sog. Moore-Umgebung bewegen. Es kann jede Zelle betreten werden, die frei ist und eine Nachbarzelle der aktuellen Zelle ist. Als Nachbarzelle wird jede Zelle bezeichnet, die mit der aktuellen Zelle eine Kante oder Ecke teilt.
Die Ziele werden als attraktiv für die Personen angesehen. Die Personen steigern ihren persönlichen Nutzen, je näher sie dem Ziel kommen. In der Realität fühlt sich eine Person jedoch unwohl, wenn ihr eine fremde Person zu nahe kommt. Im Modell wird dies dadurch

berücksichtigt, dass sich der Nutzen einer Person verringert, wenn sie einer anderen zu nahe kommt. Um die Simulation möglichst realistisch zu gestalten, bewegen sich die Personen mit einer individuellen Wunschgeschwindigkeit. Diese wird aus den Ergebnissen der ersten Studienarbeit ermittelt.
Das erläuterte Modell gibt die einzelnen Personenbewegungen aus und visualisiert diese zusätzlich. Die Bewegung der Personen wird in drei getrennten Simulationen mit jeweils unterschiedlichen Algorithmen berechnet. Zunächst wird für jedes Feld der negative euklidische Abstand als Zielnutzen berechnet. Jede Person versucht durch die Wahl des umliegenden Feldes mit dem maximalen Wert seinen Nutzen zu steigern.
Darüber hinaus besteht die Möglichkeit die Personenbewegung mit Hilfe des Floor-Flooding, basierend auf dem Dijkstra Algorithmus zu ermitteln. Jede Person wählt von den anliegenden Feldern das Feld, welches die geringste Anzahl an notwendigen Bewegungen bis zum Ziel hat.
Als dritte Variante erfolgt die Ermittlung der Personenbewegung ebenfalls mit Hilfe des Floor-Flooding. Als Grundlage dient hierbei jedoch die Lösung der Eikonal-Gleichung, die die Ausbreitung einer Welle beschreibt. Hierfür wird der Fast-Marching Algorithmus verwendet.
Im Laufe der Studienarbeit werden die beschriebenen Modelle anhand unterschiedlicher Tests verglichen und ausgewertet. Eine detaillierte Erläuterung der Tests ist im folgenden Kapitel 3.13 beschrieben.

3 Anforderungen/Requirements

3.1 Allgemeines

Es soll ein zellulärer Zustandsautomat implementiert werden. Die Software soll Personenbewegungen in der Ebene simulieren. Die unterschiedlichen Szenarien (Zustand der Zellen zu Beginn der Simulation), wie beispielsweise die Position der Hindernisse oder die Startposition jeder Person werden der Software als .png Bilddatei übergeben. Diese sind im Unterkapitel 3.13 näher erläutert.

3.2 Zellen

Das gesamte Feld (zweidimensionaler Bereich indem sich die Personen bewegen können) soll in Zellen eingeteilt werden. Die Größe des Feldes und die Anzahl an Zellen sollen der Software als Parameter übergeben werden.
Jede Zelle soll quadratisch sein und kann verschiedene Zustände haben. Sie kann entweder leer oder besetzt sein. Ist eine Zelle besetzt, wird noch weiter unterschieden, ob sie durch ein Hindernis, das Ziel- (Personensenke) oder Startfeld (Personenquelle) oder durch eine Person besetzt ist.

3.3 Hindernisse

Es soll davon ausgegangen werden, dass jedes Hindernis eine oder mehrere Zellen ganz belegt. Hindernisse können von den Personen nicht betreten werden. Falls sich ein Hindernis zwischen einer Person und ihrem Ziel befindet, ist die Person gezwungen um das Hindernis herum zu gehen.
In der Realität versucht eine Person einem Hindernis möglichst frühzeitig auszuweichen. Sie senkt somit ihren Nutzen, je näher sie einem Hindernis kommt. Dies wird im weiteren Verlauf dieser Studienarbeit nicht berücksichtigt.

3.4 Ziele

Abhängig vom entsprechenden Testszenario werden ein oder mehrere Ziele definiert. Ziele werden als attraktiv für die Personen angesehen. Jede Person erhöht ihren Nutzen, wenn sie sich dem Ziel nähert. Befinden sich mehrere Ziele im Feld, wird die Wahl des Ziels für jede Person abhängig vom verwendeten Algorithmus bestimmt.

3.5 Personen

Jede Person soll in ihrem Startfeld starten und sich im Laufe der Simulation von Zelle zu Zelle auf ihr Ziel hin bewegen und dadurch ihren persönlichen Nutzen, je näher sie dem Ziel kommt, steigern. Jede Person kann sich in der Moore-Umgebung (8 mögliche Bewegungsrichtungen) bewegen. Die möglichen Bewegungsrichtungen sind in Abbildung 1 blau markiert.

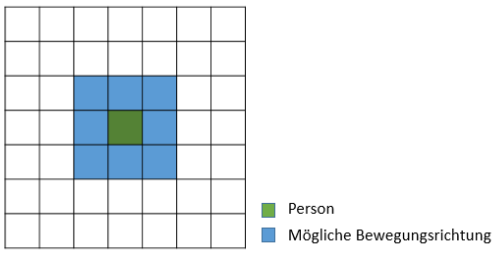

Abbildung 1: Moore-Umgebung

Die Personen können nur freie Zellen betreten. Jede Person wählt genau die Zelle als nächste Zelle, welche ihr den steilsten Nutzenanstieg ermöglicht. Der Nutzen der verschiedenen Zellen wird abhängig vom verwendeten Algorithmus unterschiedlich berechnet. Darüber hinaus soll der Nutzen durch die Position der anderen Personen beeinflusst werden.

Hierfür wird zwischen zwei Bereichen, abhängig vom Abstand zur nächsten Person unterschieden. Beträgt der Abstand zu einer fremden Person weniger als $45cm$, wird der vertraute Bereich (intimate space) der Personen betreten. Beträgt der Abstand zwischen $45cm$ und $120cm$, wird der persönliche Bereich (personal space) der Personen betreten. Für den vertrauten Bereich ist der Nutzenverlust (Nv(x)), abhängig vom Abstand x, nach Funktion 2 definiert. Für den persönlichen Bereich wird der Nutzenverlust (Np(x)), wiederum abhängig vom Abstand x zur anderen Person, durch die Funktion 1 definiert.

$$Np(x) = \mu * \exp(\frac{4}{(\frac{d(x)}{\delta_p * r})^2 - 1}) \tag{1}$$

$$Nv(x) = Np + \frac{\mu}{a_p} * \exp(\frac{4}{(\frac{d(x)}{\delta_v * r})^{2*b_p} - 1}) \tag{2}$$

$d(x)$ beschreibt den aktueller Abstand zur nächsten Person. Die folgende Tabelle enthält die Bedeutung der einzelnen Parameter und die entsprechenden Werte:

Parameter	Beschreibung	Wert
μ	Stärke des Nutzenverlustes	5.0
a_p	Dämpfung zwischen vertrautem und persönlichem Bereich	1.0
b_p	Stärke des Übergangs zwischen vertrautem und persönlichen Bereich	1
δ_p	persönlicher Bereich [m]	1.20
δ_v	vertrauter Bereich [m]	0.45
r	Radius einer Person [m]	0.2

Tabelle 1: Parameter und Werte für die Berechnung des Nutzenverlustes

Der Radius einer Person sowie die Abstände für den vertrauten und persönlichen Bereich wurden auf Basis des Modells von Hall aus [**?**] übernommen. Weitere Erläuterungen zu den Parametern, den Werten und den verwendeten Funktionen sind im Kapitel 4 beschrieben.

3.6 Geschwindigkeit der Personen

Jede Person soll eine individuelle Wunschgeschwindigkeit besitzen. Sie bewegt sich bei freier Bahn mit dieser Geschwindigkeit fort. Die Wunschgeschwindigkeiten sollen als normalverteilt angesehen werden. Als Mittelwert und Standardabweichung sollen die errechneten Werte aus der ersten Studienarbeit verwendet werden.

3.7 Zeitmodellierung

Die Simulation unterliegt keinen harten Echtzeitanforderungen. Sie soll für die gesamte Simmulationsdauer so schnell wie möglich durchgeführt werden. Es soll keine globale Uhr

modelliert werden.
Die Visualisierung der Simulation soll hingegen die Bewegungen der Personen möglichst in Echtzeit darstellen. Sie unterliegt weichen Echtzeitanforderungen. Abweichungen aufgrund mangelnder Rechner-Ressourcen sind tolerierbar.

3.8 Parameter

Die Tabelle 2 zeigt alle möglichen Parameter, welche die Software steuern. Alle Parameter sind optional.

Parameter	Beschreibung	Default
–free-flow-velocity	Mittelwert der Wunschgeschwindigkeit [m/s]	1,0
–free-flow-deviation	Standardabweichung der Wunschgeschwindigkeiten [m/s]	0
–algorithm	Algorithmus (euclid, dijkstra, fast-marching)	dijkstra
–output-folder	Ordner in den der Output gespeichert wird	../data/
–cellsize	Größe der einzelnen Zellen [m]	1
–input-map	Pfad zum Bild der map (Auswahl des Testfalls)	default.png

Tabelle 2: Parameter und Defaultwerte

3.9 Berechnungen

Die Aufgabe der Software ist die Simulation von Personenbewegungen. Für möglichst genaue Zeitangaben wird die Java Klasse BigDecimal verwendet. Für die Division werden 32 Nachkommastellen berücksichtigt und als Rundungsmodus wird Half_Even verwendet, da dieser kumulative Fehler bei sich wiederholenden Berechnungen minimiert. Die anschließende Berechnung und Aufbereitung der Daten (insebesondere für den Test 3.13.6) erfolgt mit Mathematica.

3.10 Report

Alle errechneten Ergebnisse sollen über einen Report ausgeleitet werden. Für jedes Testszenario soll ein eigenes Verzeichnis angelegt werden, in dem der Report und die verwendete Map (vgl. Abschnitt 3.13) gespeichert werden sollen. Der Report soll im .xml Format ausgeleitet werden um eine gute Weiterverarbeitung der Ergebnisse (Visualisierung) zu gewährleisten. Er soll alle benötigten Informationen, wie beispielsweise die einzelnen Events der Personen, die durch den Algorithmus berechneten Entfernungen der einzelnen Zellen vom Ziel und eine Übersicht über den Zustand der einzelnen Zellen (Fieldmap), für Simulation beinhalten.
Darüber hinaus sollen im Verzeichnis 3 .csv Dateien angelegt werden. Wird kein Messfeld (vgl. Abschnitt 3.13) definiert, sollen diese nicht befüllt werden. Falls ein Messfeld

in der Map definiert wurde, soll in der Datei *timeDensity.csv* die Personendichte im Messbereich zu verschiedenen Messzeitpunkten gespeichert werden. In der Datei *timeVelocity.csv* sollen die durchschnittliche Geschwindigkeit der Personen im Messbereich zu verschiedenen Messzeitpunkten gespeichert werden. Die Datei *fundamental.csv* soll die ermittelten Personendichten und Geschwindigkeiten enthalten, sodass daraus bei Bedarf ein Fundamentaldiagramm erstellt werden kann. Messzeitpunkte sollen in einem Intervall von $0,5s$ definiert werden.

3.11 Berechung des Zielnutzens

Es sollen drei unterschiedliche Algorithmen für die Berechnung des Zielnutzens (Attraktivität) einer Zelle implementiert werden.

Algorithmus 1: Der Nutzen bzw. die Attraktivität jeder Zelle entspricht dem negativen euklidischen Abstand. Hierbei können keine Hindernisse umgangen werden. Jede Person versucht den Nutzen zu maximieren und wählt jeweils die Zelle mit maximalen Wert aus den umliegenden aus.

Algorithmus 2: Der Nutzen bzw. die Attraktivität jeder Zelle wird durch Floor-Flooding, basierend auf dem Dijkstra Algorithmus, berechnet. Für jede Zelle wird die geringste Anzahl an notwendigen Bewegungen bis zum Ziel ermittelt.

Algorithmus 3: Analog zum Algorithmus 2 wird auch in diesem hier der Nutzen bzw. die Attraktivität jeder Zelle mittels Floor-Flooding berechnet. Es wird jedoch der Fast-Marching Alforithmus (Lösung der Eikonal-Gleichung) als Grundlage verwendet.

3.12 Werte für die Auswertung

Alle in der Tabelle 3 aufgeführten Werte sollen auf Basis der Simulation ermittelt und ausgewertet werden. Einige Werte werden nur für bestimmte Tests benötigt.

Ermittelte Werte	Beschreibung	Wann benötigt
mittlere Geschwindigkeit	durchschnittliche Geschwindigkeit der Personen im Feld	Rimea-Test 4
Fluss (Personen/Meter/s)	Anzahl der Personen mit Geschwindigkeit $>$ 1m/s im Feld	Rimea-Test 4
Simulationsdauer	Dauer bis der Test beendet ist	bei allen Tests

Tabelle 3: Werte für die Auswertung

3.13 Testszenarien für die Simulation

Die implementierten Algorithmen (vgl. Unterkapitel 3.11) sollen anhand unterschiedlicher Testszenarien verglichen und verifiziert werden. Die verschiedenen Szenarien sollen der Simulation als .png Datei übergeben werden.

3.13.1 Erstellung der Testszenarien

Es soll die Möglichkeit bestehen, unterschiedliche Testszenarien mittels eines Bildbearbeitungsprogrammes zu erstellen und an die Simulation (als .png Datei) zu übergeben. Bei der Erstellung ist sowohl die Anzahl der Personen, als auch die Anzahl der Hindernisse und Ziele (Personensenken) beliebig. Hinsichtlich der Farben der Objekte müssen jedoch die in der folgenden Tabelle 4 beschriebenen Farbcodes verwendet werden. Jedes Pixel wird durch eine Zelle des Automaten repräsentiert. Darüber hinaus soll die Möglichkeit bestehen ein Messfeld zu definieren. In diesem Feld wird die durchschnittliche Geschwindigkeit der Personen ermittelt in einer .csv Datei ausgegeben. Für das Messfeld müssen in der .png Datei lediglich die beiden diagonalen Eckpunkte angegeben werden. Die Implementierung ermittelt die Koordinaten und erstellt daraus ein rechteckiges Feld.

Farbcode	Farbe	Bedeutung
#3F48CC	Blau	Ziel/ Personensenke
#000000	Schwarz	Ziel/ Personensenke
#FFFFFF	Weiß	Leere Zelle
#22B14C	Grün	Person
#FF0000	Rot	Messfeld

Tabelle 4: Werte für die Auswertung

3.13.2 Testszenario: Freier Fluss

Im Zuge dieses Tests soll überprüft werden, ob sich Personen bei freier Bahn (keine Hindernisse zwischen Start und Ziel) mit ihrer jeweiligen Wunschgeschwindigkeit auf das Ziel hin bewegen. Des weiteren soll überprüft werden, ob die Person den kürzesten Weg verwendet. Der Test soll mit allen 3 Betriebsmodi durchgeführt werden. Abbildung 2 stellt den Testaufbau schematisch dar.

3.13.3 Testszenario: Hühnertest

Zwischen Start und Ziel soll ein U-förmiges Hindernis (Öffnung in Richtung des Startes) eingefügt werden. Abhängig vom verwendeten Algorithmus wird ein unterschiedliches Ergebnis erwartet:
Euklid: Die Personen sollen sich auf dem kürzesten Weg auf das Ziel zubewegen. Sobald

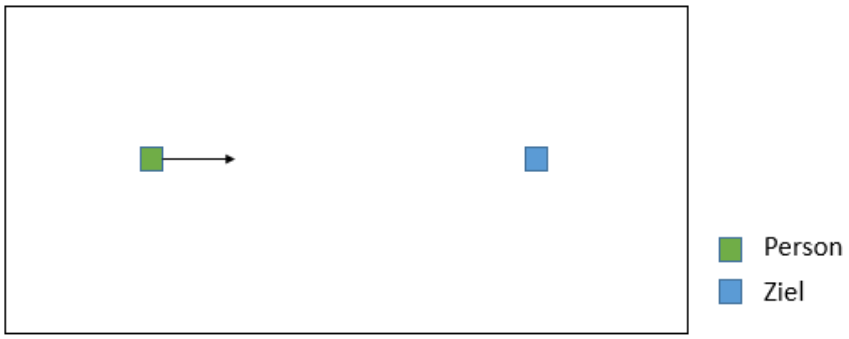

Abbildung 2: Testfall Freier Fluss (schematisch)

sie das Hindernis erreicht haben, sollen sie stehen bleiben. Erwartungsgemäß wird der Test nicht erfolgreich verlaufen.
Dijkstra und Fast-Marching Algorithmus: Die Personen sollen das Hindernis umgehen und das Ziel erreichen. Bei einem „Feststecken im Hindernis“(wie bei Verwendung des Euklid Algorithmus erwartet) wird der Test als fehlgeschlagen gewertet. Abbildung 3 stellt den Testaufbau schematisch dar.

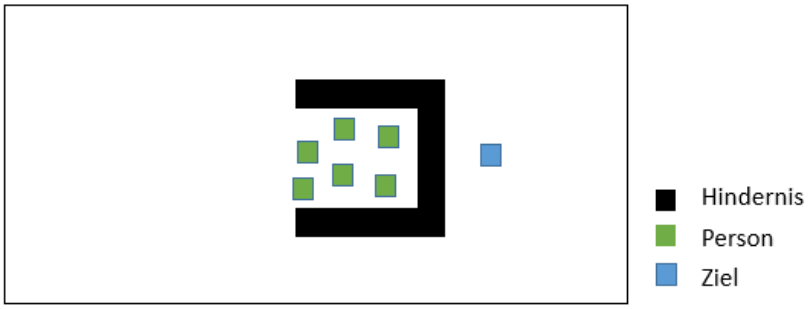

Abbildung 3: Testfall Hühnertest (schematisch)

3.13.4 Testszenario: Evakuierung eines Raumes mit 2 Türen

Die Personen befinden sich zu Beginn in einen von Hindernissen umgebenen, quadratischen Raum. An 2 Seiten sind Öffnungen platziert (freie Zellen). Hinter diesen Öffnungen befinden sich jeweils Ziele. Im Laufe des Tests sollen die Personen durch die beiden Engstellen zum Ziel gelangen.
Dieser Test soll auf Basis des Dijkstra und Fast-Marching Algorithmus in unterschiedlichen Versionen durchgeführt werden. Zunächst sollen die Türen an benachbarten Seiten

mit sehr geringem Abstand zueinander platziert werden. Anschließend sollen die Türen an gegenüberliegenden Seiten mit sehr großem Abstand zueinander platziert werden. Unterschiede hinsichtlich der Evakuierungsdauer, aufgrund der Algorithmen oder Positionen der Türen sind festzuhalten. Die Abbildung 3.13.4 zeigt die unterschiedlichen Testversionen schematisch.

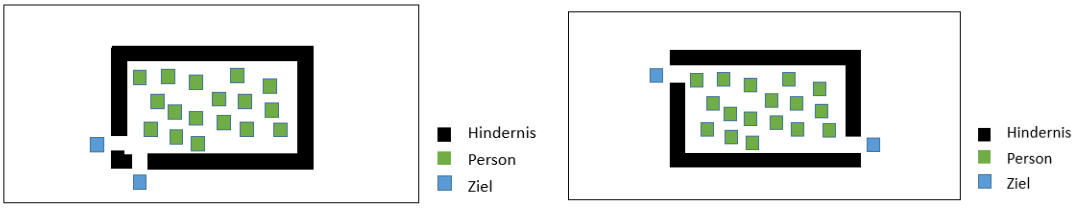

Abbildung 4: Testfall Evakuierung eines Raumes mit 2 Türen, links: geringer Abstand zwischen den Türen, rechts: großer Abstand zwischen den Türen

3.13.5 Testszenario: Evakuierung eines Raumes mit 4 Türen

Für diesem Testfall soll sich an jeder Raumseite eine Tür befinden. Der Raum soll absolut symmetrisch aufgebaut sein und die Türen sollen sich jeweils in der Mitte jeder Raumseite befinden. In einem weiteren Szenario sollen zwei Türen an gegenüber liegenden Seiten entfernt werden. Ziel dieser beiden Testszenarien soll es sein, die Unterschiede hinsichtlich der Evakuierungsdauer eines identischen Raumes zu ermitteln, wenn sich in diesem zwei oder vier Türen befinden. Die Simulation soll anhand dieses Aspekts verifiziert werden. Es wird erwartet, dass sich die Evakuierungszeit durch die doppelte Anzahl an Türen nahezu halbiert.

3.13.6 Testszenario: Fundamentaldiagramm - Rimea-Test 4

In diesem Testfall soll ein $65m$ langer und $12m$ breiter Gang angelegt werden. Auf der einen Seite des Ganges befindet sich die Personenquelle, auf der anderen Seite die Personensenke. Im Laufe des Testes sollen die Personen den Gang entlangschreiben. Die Anzahl der Personen ist zu variieren. In einer weiteren Version des Tests sollen Personen, welche die Personensenke erreicht haben, erneut bei der Personenquelle starten (Endlosschleife). Abbildung 5 zeigt den Testaufbau schematisch.

3.13.7 Testszenario: Engstelle (unidirektional)

In diesem Testfall soll, analog zum Rimea-Test 4, ein $65m$ langer und $12m$ breiter Gang angelegt werden. Am Ende des Ganges befindet sich eine Engstelle mit $6m$ Breite und dahinter die Personensenke. Es soll mit variierender Anzahl an Personen getestet werden,

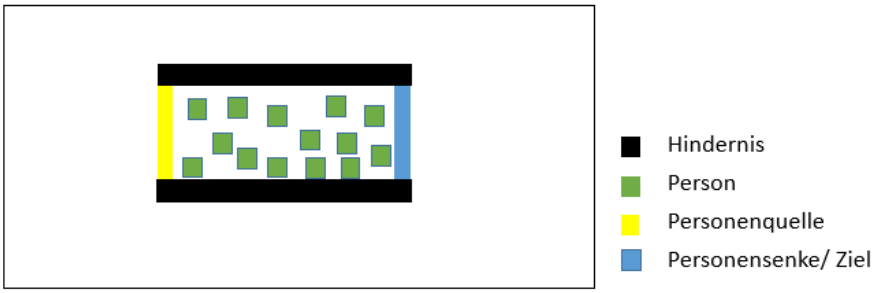

Abbildung 5: Testfall Rimea 4 (schematisch)

wie lange es dauert bis die Engstelle durchschritten wurde. Der schematische Testaufbau ist in Abbildung 6 aufgeführt.

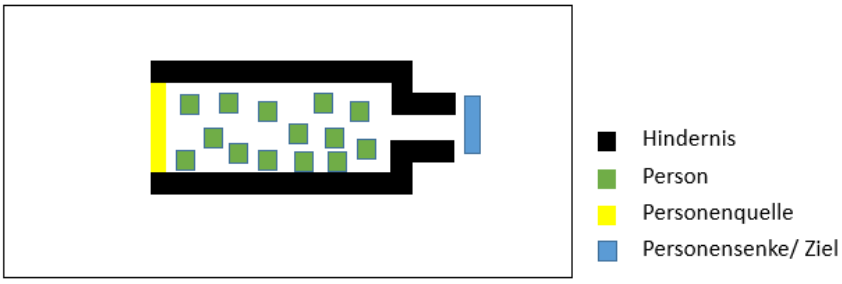

Abbildung 6: Testfall Engstelle (schematisch)

3.14 Visualisierung

Die Visualisierung soll die errechneten Personenbewegungen graphisch darstellen. Als Grundlage soll der ausgegebene Report (vgl. Abschnitt 3.10) dienen. Konkrete Anforderungen bezüglich der Visualisierung sind in Abbildung 7 abgebildet.

Die Oberfläche der Visualisierung soll aus 4 verschiedenen Teilen bestehen, welche in der Abbildung durch die blauen Kästen A-D hervorgehoben wurden. Im Teil A sollen Konfigurationsmöglichkeiten für die Visualisierung aufgeführt sein. Beispielsweise soll zwischen einer normalen Darstellung der Simulation und einer Heatmap gewechselt werden können. Darüber hinaus soll die Möglichkeit bestehen, die Visualisierung zu stoppen und schrittweise fortzusetzen.
Teil B enthält die eigentliche Visualisierung der Personen, Hindernisse und Ziele. In Teil C sind alle Informationen bezüglich der Visualisierung aufgeführt und Teil D enthält eine Legende der verwendeten Symbole und Farben. Wird ein Messfeld definiert, soll dieses in der Visualisierung hervorgehoben werden.

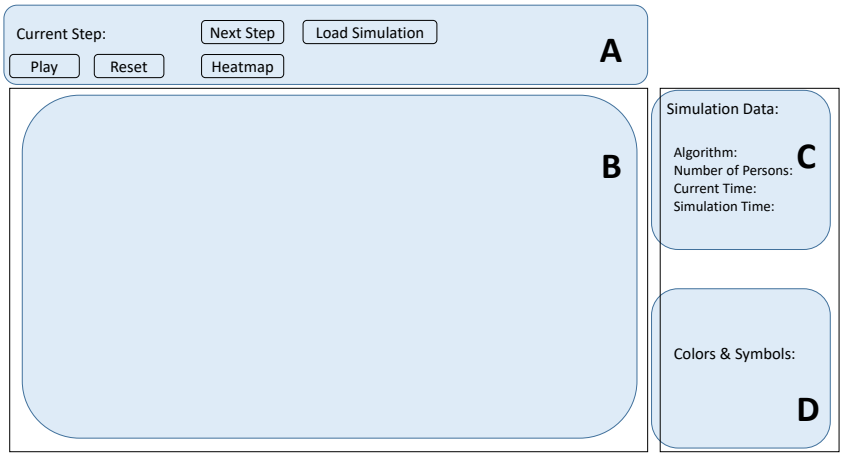

Abbildung 7: Visualisierung (schematisch)

4 Beeinflussung durch andere Personen

Wie bereits im Abschnitt 3.5 kurz erläutert, wird die Entscheidung, welche Zelle eine Person als nächstes betritt abhängig vom verwendeten Algorithmus und von den Positionen der anderen Personen getroffen. In der Realität fühlen sich Personen unwohl, wenn sie anderen, insbesondere fremden Personen zu nahe kommen. Sie versuchen folglich eine solche Situation wenn möglich zu vermeiden. Beispielsweise bei der Fahrt in einem überfüllten Aufzug lässt sich diese Situation unter Umständen nicht vermeiden. Bei der Fortbewegung in der Ebene jedoch häufig schon. Personen neigen in der Realität dazu einen Umweg in kauf zu nehmen, um einem Gedränge zu entkommen. Um dieses Verhalten in der Simulation zu berücksichtigen existieren verschiedene mathematische Funktionen, mithilfe derer ein Abstoßungspotenzial berechnet wird. Während auf der einen Seite der Nutzen einer Person steigt, je näher sie dem Ziel kommt, verringert sich dieser Nutzen auf der anderen Seite wieder, wenn sie anderen Personen zu nahe kommen. Mit Hilfe von Funktionen kann somit der Nutzenverlust einer Person bestimmt werden. Insbesondere glatte Funktionen auf kompakten Trägern, die außerhalb eines begrenzten Gebietes komplett zu verschwinden (f(x) = 0) eignen sich hierfür gut. Würde man eine Funktion verwenden, deren Funktionswert auch außerhalb des begrenzten Gebietes größer als 0 ist, führe dies zu einer Abweichung von der Realität und damit zu einem Fehler. In diesem Fall, wird eine Person im Modell auch durch eine sehr weit entfernte Person, wenn auch nur leicht, beeinflusst. Dies widerspricht der Realität, da das Abstoßungspotenzial von Personen auf einen Bereich um die entsprechende Person beschränkt ist.
In der Fachliteratur wird beispielsweise folgende Funktion vorgeschlagen:

$$N(d) = \begin{cases} -h \exp(\frac{1}{(\frac{d}{w})^2 - 1}) & falls\, |d| < w \\ 0 & sonst \end{cases}$$

Hierbei wird der Nutzenverlust ($N(d)$), abhängig vom Abstand d zur nächsten Person definiert. w beschreibt die Reichweite der Funktion und h die Stärke der Beeinflussung bzw. des Nutzenverlustes [**?**].
Für die Ermittlung des Abstoßungspotenzials bzw. des Nutzenverlustes aufgrund anderer Personen wurden jedoch anstelle dieser Funktion, die in Abschnitt 3.5 aufgeführten Funktionen 1 und 2 verwendet. Diese basieren auf dem Modell von Hall, das den Bereich um eine Person in mehrere Teilbereiche einteilt. In einem Umkreis von bis zu $45cm$ wird der vertraute Bereich einer Person betreten. Befinden sich zwei Personen so nah beieinander, können Berührungen nicht vermieden werden. Zwischen $45cm$ und $120cm$ um eine Person befindet sich der persönliche Bereich, welcher beispielsweise von Freunden oder Familienmitgliedern betreten werden darf. Ein Betreten dieses Bereichs durch fremde Personen wird als unangenehm und aufdringlich empfunden und Personen versuchen es folglich zu vermeiden. Darüber hinaus definiert Hall noch zwei weitere Bereiche. Zwischen $120cm$ und $360cm$ um die Person befindet sich der soziale oder gesellschaftliche Bereich der Person und ab $360cm$ der öffentliche Bereich. Für die Simulation in dieser Studienarbeit sind die beiden letzten Bereiche jedoch nicht relevant, da der Einfluss auf den Nutzen der Person zu gering ist, um die Entscheidung, wohin die Person als nächstes geht zu beeinflussen. Außerdem lässt sich das Betreten dieser beiden Bereiche durch andere Personen bei einer höheren Personendichte generell nicht vermeiden.
Hintergrund dieser Entscheidung ist, dass nach unserer Einschätzung und nach [**?**] die Berücksichtigung des Modells von Hall durch die Verwendung unterschiedlicher Funktionen für den vertrauten und persönlichen Bereich zu einem Verhalten, das der Realität näher kommt, führt. Berührt eine Person eine fremde Person, fühlt sie sich deutlich unwohler, als wenn sie dieser lediglich nahe kommt mit einem Abstand von beispielsweise $50cm$. Diese Eigenschaft wird durch die Funktion 4 kaum berücksichtigt. Außerdem lassen sich in dieser Simulation, durch die Wahl einer geringen Zellgröße des zellulären Automatens von $40cm$, die Auswirkungen der unterschiedlichen Bereiche auf den Nutzenverlust bzw. das Abstoßungspotenzial der Personen feststellen.
Darüber hinaus beinhalten die Funktionen 1 und 2 deutlich mehr Parameter. Diese Parameter können an mögliche gesellschaftliche oder länderspezifische Unterschiede angepasst werden und so die Realität besser widerspiegeln. Für diese Studienarbeit wurden die Parameter wie in der Tabelle 1 aufgeführt gewählt. Grundlage hierfür sind die in [**?**] erläuterten Erkenntnisse aus unterschiedlichen Versuchen. Außerdem zeigten mehrere, in der Gruppe durchgeführten Simulationen mit unterschiedlichen Parameterkombinationen, dass sich das Abstoßungspotenzial durch die Verwendung dieser Parameter so auswirkte, wie man es in der Realität erwarten würde.

5 Softwaredesign

Der Aufbau der Anwendung wurde im Team diskutiert und anschließend mittels UML spezifiziert.

6 Softwaretest

7 Vergleich der Algorithmen

Die Algorithmen werden in Abbildung 8 in Vergleich gesetzt. Dafür wird eine Karte erstellt mit einem Raum, einer Tür und einem Hindernis vor der Tür. Die Karten werden jeweils nach 1, 20, 100, 1000 Schritten geplottet. Als letztes wird noch das Gesamtergebnis dargestellt.

7.1 Euklid

Um den Euklidischen Abstand zu berechnen, benötigt man lediglich die Position des Ziels, die Position der aktuellen Zelle und die Zellbreite. Man geht alle Zellen der Reihe nach durch und berechnet den Abstand der aktuellen Zelle zum Ziel. Die Suche nach noch nicht berechneten Zellen läuft nicht von oben nach unten sondern mit der Funktion „parallelStream()“. Diese teilt das Feld in Teilfelder auf und sucht parallel nach freien (noch nicht berechneten) Feldern. Dadurch entstehen die in Abbildung 8 sichtbaren Stufen. [1] Darüber hinaus ist zu erkennen, dass das Hindernis vor der Tür für den Euklid Algorithmus keinen Einfluss hat. Dieses Hindernis wird bei der Berechnung ignoriert.

7.2 Dijkstra

Beim Dijkstra Algorithmus geht man ausgehend vom Ziel immer ein Feld weiter und berechnet den Abstand des aktuellen Feldes anhand des Abstand des Nachbarfeldes. Dadurch wird die Karte ausgehend vom Ziel aufgebaut wie in Abbildung 8 zu sehen ist. Es ist auch zu erkennen, dass der Dijkstra Algorithmus die Abstände um das Hindernis berechnet. Das Hindernis wird also in die Berechnung einbezogen.

7.3 Fast Marching

Ähnlich wie beim Dijkstra arbeitet der Fast Marching Algorithmus vom Ziel aus. Das Zielfeld wird als Basis angenommen und die Nachbarfelder um das Ziel werden als „considered“also als „in Betracht kommend“markiert. Das in Betracht kommende Nachbarfeld mit dem besten Wert wird als „accepted“also als „akzeptiert“markiert. Danach wird

für alle Nachbarn dieses akzeptieren Feldes die Entfernung mittels der Eikonalgleichung berechnet. Auch hier ist also die Berechnung abhängig von den vorher berechneten Feldern. Es ist deutlich zu erkennen, dass der Fast Marching Algorithmus eine deutlich rundere Karte erzeugt, als der Dijkstra.

7.4 Einfluss der Zellgröße auf die Abstandsberechnung

Bei der Berechnung der Abstände hat die Wahl der Zellgröße je nach Algorithmus einen entscheidenden Einfluss auf die Ergebnisse der Simulation. Die Zellgröße gibt nicht nur die minimale Schrittweite vor, sondern hat auch Einfluss auf die Genauigkeit der Nutzenberechnungen. An dieser Stelle wird der Euklid Algorithmus dem Fast Marching Algorithmus gegenübergestellt. Um einen Vergleich darzustellen wurde ein Quadratisches Feld angelegt. Die Zellgröße des Feldes sowie die Anzahl der Felder wurden so variiert, dass die Entfernung von der rechten unteren Ecke zur linken oberen Ecke (am weitesten entfernter Punkt) immer den gleichen Abstand hat. Es wird also die Diagonale des Rechtecks berechnet.

7.4.1 Grafische Überprüfung

Die Abbildung 10 Zeigt den Vergleich beider Algorithmen bei der der Abstandsberechnung. Üblicherweise wird die Farbskala so gewählt um alle Werte vom kleinsten Abstand zum weitesten Punkt abzubilden. Um den Vergleich an dieser Stelle deutlich zu machen, wurde für beide Bilder die selbe Farbskala verwendet. Mit Bloßem Auge ist die Differenz nur schwer zu erkennen. Es ist jedoch sichtbar, dass das Feld, welches mit dem Euklid Algorithmus berechnet wurde, einen deutlicheren Bogen aufweist.

Eine Möglichkeit die Unterschiede zu visualisieren ist es, die Bilder mittels einer Differenzbildung der Beiden Bilder. Dafür wird ein Bild als Basis hergenommen und das zweite Bild als eine Ebene darüber eingefügt. Die Farben der zweiten Ebene werden von der ersten Ebene Subtrahiert. Gleichen sich die Farben beider Ebenen, so ist das Ergebnis schwarz. Dieser Vergleich macht nur Sinn, wenn für beide Darstellungen die selbe Farbskala verwendet wurde, also die Entfernungen mit dem gleichen Farbcode codiert wurden. Die Abbildung 11 zeigt die Differenzmenge der in Abbildung 10 dargestellten Karten. An diesem Bild ist deutlich zu erkennen, dass die Zellen, welche sich nur in X und nur in Y Richtung vom Ziel entfernen schwarz sind. Es gibt also keinen Unterschied zwischen dem Euklidischen Abstand und dem Abstand welcher mittels Fast Marching berechnet wurde, bei einer Betrachtung der Zellen nur in X oder nur in Y Richtung. Entfernt man sich schräg vom Ziel (Also X und Y Anteil gleichzeitig), sieht man deutlich Differenzen der beiden Algorithmen.

Um dies nochmals zu verdeutlichen werden die Algorithmen in Abbildung 12 mit einer kleineren Zellgröße verglichen. Auch hier wurde auf eine einheitliche Farbskala geachtet. Die Differenzbildung ist in Abbildung 13 zu sehen. Es ist eine Art Kegel zu erkennen.

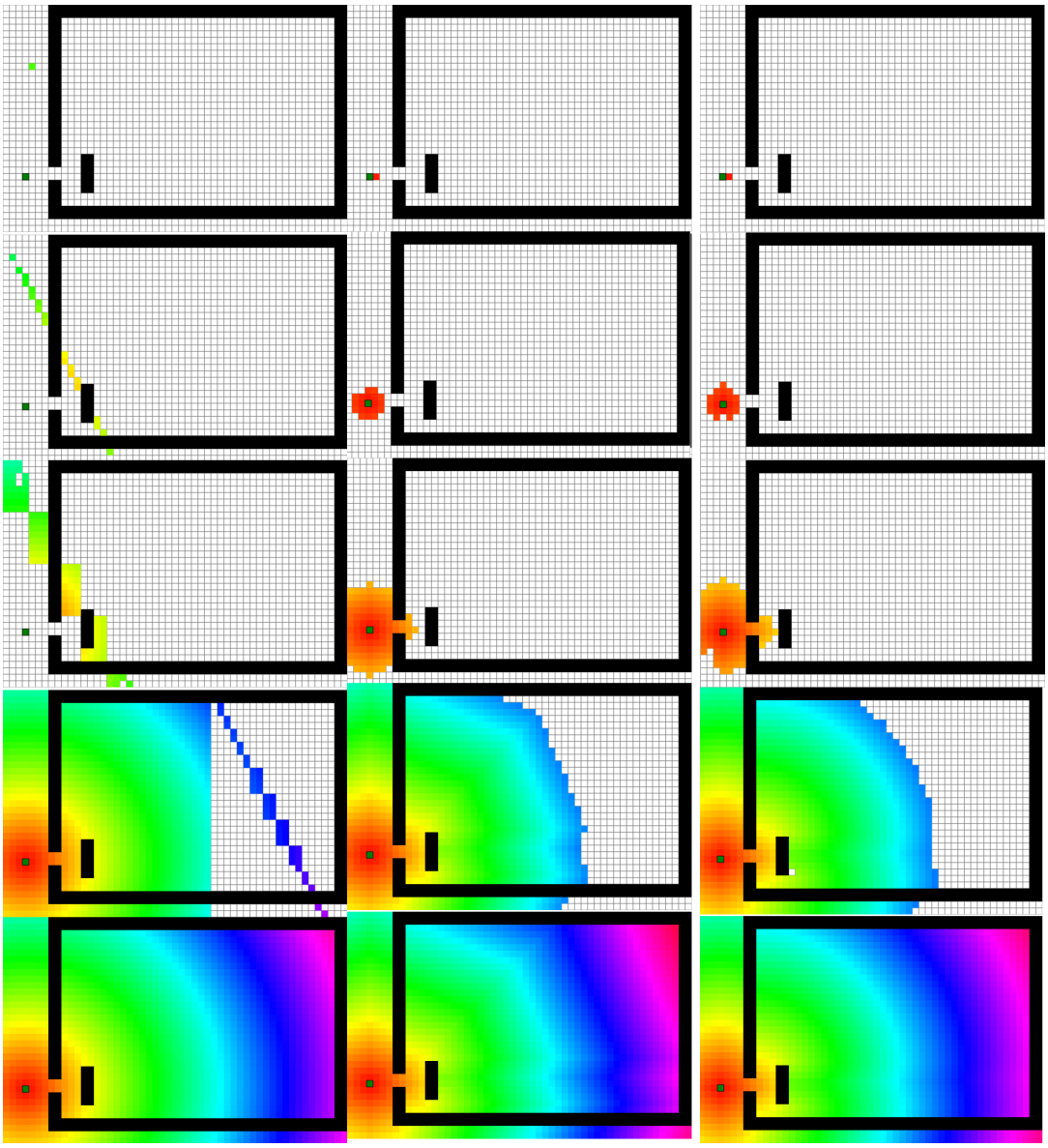

Abbildung 8: Die berechnung der Abstände mit verschiedenen Algorithmen im Vergleich. Euklid (links), Dijkstra (mitte), Fast Marching (rechts), jeweils von oben nach unten nach 1, 20, 100, 1000 Schritten und Gesamtbild

Die Kanten unten und rechts sind Schwarz, da sich die Abstände beim Euklid und Fast Marching gleichen. Näher zur Diagonalen wird das Bild heller, da es einen größeren Unterschied gibt.

7.4.2 Rechnerische Überprüfung

Eine rechnerische Überprüfung wird auf den diagonalen Abstand durchgeführt. Die Tabelle 5 zeigt die Ergebnisse dieses Experiments. Die Cellsize gibt dabei die Kantenlänge einer quadratischen Zelle an. Wichtig ist der Wert „Distance in X and Y“, welcher besagt, dass das Ziel in X und in Y Richtung immer jeweils 14 Zellen entfernt ist. Nach folgender Formel wurde der Abstand vom Ziel zum am weitesten entfernten Punkt (also diagonal) berechnet:

$$a = \sqrt{x^2 + y^2} = \sqrt{(14\ m)^2 + (14\ m)^2} = 19,799\ m$$

Wie in der Tabelle zu sehen ist, berechnet der Euklid Algorithmus den Abstand sehr präzise. Da der Euklid Algorithmus kein „Gedächtnis“hat, also ohne Einfluss vorheriger Zellen berechnet und die oben genannte Formel anwendet, ist der Algorithmus gut geeignet um die exakte Entfernung zu berechnen. Interessant in der Tabelle ist vor allem die Abweichung von 1,45 m, welche beim Fast Marching Algorithmus entsteht. Diese Abweichung entspricht einem Fehler von von 7,3 % bezogen auf die tatsächliche Entfernung. Verringert man die Zellgröße, so konvergiert der Fast Marching Algorithmus gegen den euklidischen Abstand und der Fehler gegen 0. Die prozentualen Fehler in Abhängigkeit der Zellgröße sind in Abbildung 14 dargestellt. Dies ist die Darstellung des Fehlers in Abhängigkeit von der Zellgröße in dem Punkt, von dem angenommen wird, dass dieser den größten Fehler hat. Eine grafische Darstellung des Fehlers über alle Punkte ist in der Abbildung 9 zu sehen. In dieser Abbildung entspricht die Position (0|0) der linken, oberen Bildecke. Das Ziel ist diagonal gegenüber festgelegt (rechte untere Bildecke). Auf der Z Achse ist der Absolute Fehler aufgetragen. Diese Abbildung zeigt deutlich, dass der Fehler an den Achsen 0 beträgt und in der Diagonalen am höchsten ist. Der Fehler wird kleiner, wenn man die Zellgröße verringert.

8 Einfluss der Zellgröße auf die maximale Dichte

Darüber hinaus hat die Zellgröße direkten Einfluss auf die Dichte $[\frac{Personen}{m^2}]$ in einem Feld. Würde man die Zellgröße von 2 m^2 wählen, erhielte man eine maximale Dichte von 0,5 $\frac{1}{m^2}$. Es ist darauf zu achten, dass die Zellgröße angemessen gewählt wird. Bei weiteren Versuchen wurde stets eine Zellbreite von 0,4 bis 0,5 m verwendet, sofern nichts anderes angegeben.

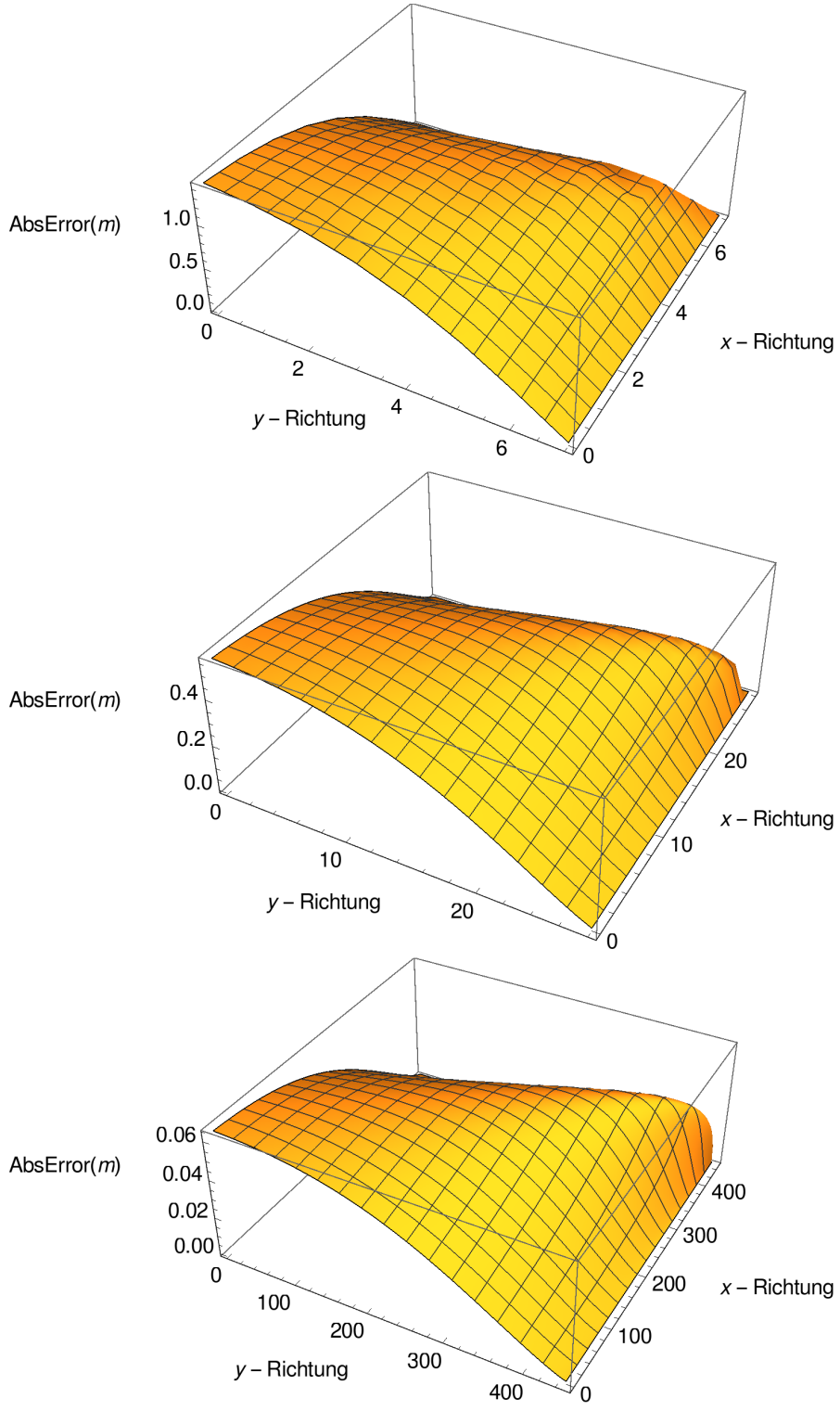

Abbildung 9: Darstellung des Absoluten Fehlers in allen Punkten der Karte. X und Y stellen die Zellkoordinaten dar (Ohne Einheit)

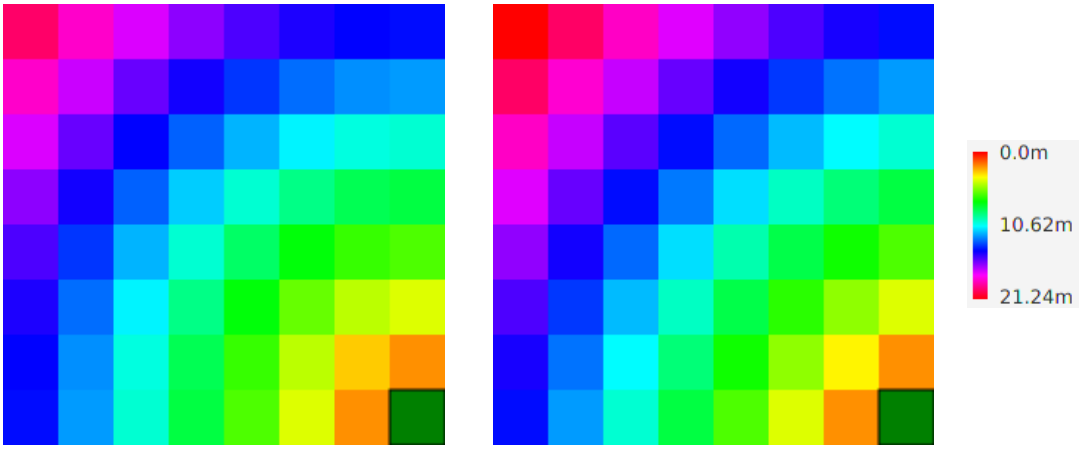

Abbildung 10: Berechnung der Abstände mittels Euklid (links) und Fast Marching (rechts) an einem Feld mit einer Zellbreite von 2 m. Dunkelgrün: Ziel, Rot: am weitesten entfernter Punkt im Feld

Cellsize [m]	2	1	0,5	0,25	0,125	0,0625	0,03125
Cells in X and Y	7	14	28	56	112	224	448
Distance in X and Y [m]	**14**	**14**	**14**	**14**	**14**	**14**	**14**
Tatsächliche Entfernung [m]	19,799	19,799	19,799	19,799	19,799	19,799	19,799
Euklid [m]	19,799	19,799	19,799	19,799	19,799	19,799	19,799
Fast Marching [m]	**21,245**	**20,717**	**20,363**	**20,137**	**19,997**	**19,913**	**19,863**
Diff [m]	1,45	0,92	0,5645	0,3380	0,1979	0,1138	0,0644
Error [%] diff/distance	**7,30**	**4,64**	**2,85**	**1,71**	**1,00**	**0,57**	**0,33**

Tabelle 5: Vergleich der Algorithmen Fast Marching und Euklid. Cellsize ist die Kantenlänge einer quadratischen Zelle in m (Zellbreite)

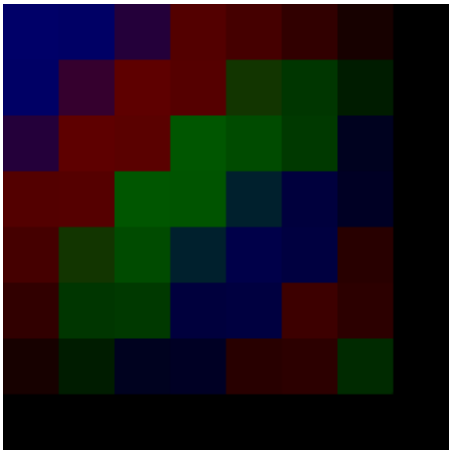

Abbildung 11: Differenzbildung der beiden Karten Euklid und Fast Marching bei einer Zellbreite von 2 m

8.1 Verifikation

8.1.1 Verteilung Wunschgeschwindigkeit in der Ebene

Um die korrekte Funktion der Simulation gewährleisten zu können, muss die Geschwindigkeit der Personen normal verteilt sein. Für die Überprüfung wurden die erzeugten Geschwindigkeiten von Fußgängern mitgeloggt und in einem Histogramm, das in Abbildung 15 zu sehen ist, dargestellt. Da die Werte sehr nah an der erwarteten Linie (in der Abbildung blau) liegen kann von einer Normalverteilung ausgegangen werden.

Dadurch, dass die Normalverteilung nicht das Thema dieser Arbeit ist, wurde von weiteren Tests auf Normalverteilung abgesehen.

In der Simulation ist es möglich einen Seed für die Geschwindigkeiten einzustellen um sicherzugehen, dass bei verschiedenen Durchläufen, die gleichen Verteilung von Personengeschwindigkeiten erfolgt.

8.1.2 Hühnertest

Der Hühnertest wurde auf Basis des in Abbildung 16 aufgeführten Feldes durchgeführt. Die einzelnen Farben der einzelnen Elemente sind analog zu den in Tabelle 4 beschriebenen gewählt. Personen werden grün, Hindernisse schwarz und das Ziel blau dargestellt. Da mit Hilfe dieses Testes lediglich überprüft werden soll, ob die Personen den Weg aus dem Hindernis finden, ist die Anzahl von 39 simulierten Personen ausreichend. Der Test wurde dreimal durchgeführt. Zunächst wurde der Euklid Algorithmus, anschließend Dijkstra und zuletzt der Fast-Marching Algorithmus für die Berechnung des Zielnutzens verwendet.

Die Abbildungen 17 und 18 zeigen die Heatmap des Testes zu Beginn und am Ende des

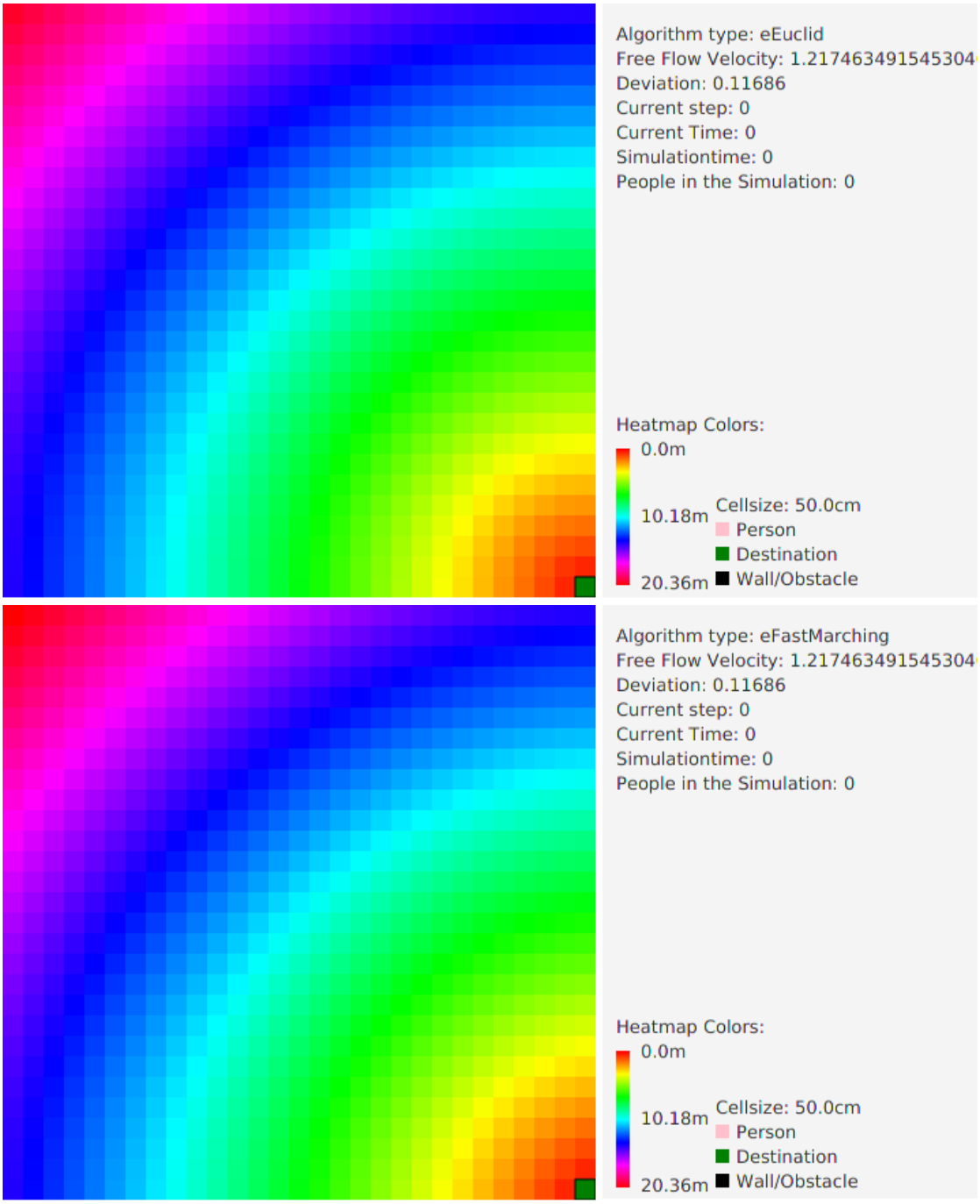

Abbildung 12: Berechnung der Abstände mittels Euklid (oben) und Fast Marching (unten) an einem Feld mit einer Zellbreite von 0,5 m

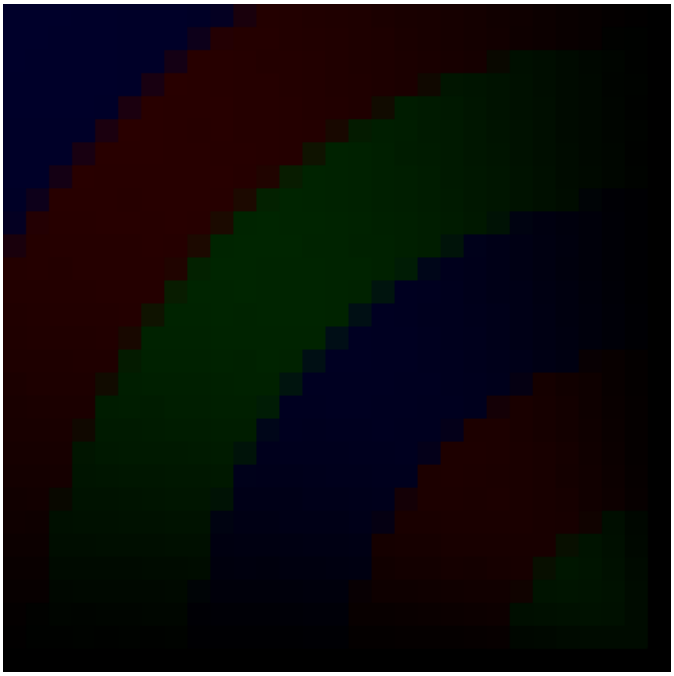

Abbildung 13: Differenzbildung der beiden Karten Euklid und Fast Marching bei einer Zellbreite von 0.5 m

Testes auf Basis des Euklid Algorithmus. Wie zu erwarten war, laufen alle Personen im Verlauf des Tests auf dem kürzesten Weg in Richtung des Ziels. Sie sind nicht in der Lage das Hindernis zu Umlaufen und sammeln sich vor dem Hindernis, an der Stelle mit der geringsten Distanz zum Ziel. Dies wird durch die Heatmap verdeutlicht. Abhängig von der Farbe der Heatmap werden die unterschiedlichen Entfernungen der einzelnen Felder zum Ziel, und damit die Attraktivität der Felder dargestellt. Wie die Abbildungen zeigen, werden Hindernisse bei der Ermittlung der Entfernungen nicht berücksichtigt.

Im Vergleich dazu zeigen die Abbildungen 19 und 20 jeweils die Heatmap des Floor-Flooding mit dem Dijkstra Algorithmus zu Beginn der Simulation und nach einer SImulationsdauer von ca. $8s$. Es fällt auf, dass diese Heatmap deutlich anders aussieht, als die des Euklid Algorithmus. Die Hindernisse werden bei der Ermittlung der Entfernung der Zellen vom Ziel berücksichtigt und folglich farblich anders dargestellt. Abbildung 20 zeigt außerdem, dass sich die Personen in 2 Gruppen aufteilen und das Hindernis an beiden Seiten umlaufen. Nach einer Simulationsdauer von ca. $34,32s$ sind alle Personen im Ziel angekommen.

Abschließend zeigt Abbildung 21 die Heatmap des Fast-Marching Algorithmus.

Zusammenfassend ist festzustellen, dass die Personen in der Simulation mit dem Euklid Algorithmus im Hindernis hängen blieben. Bei den beiden anderen Algorithmen fanden sie den Weg ins Ziel. Beide Male teilten sie sich in 2 Gruppen auf. Für eine bessere

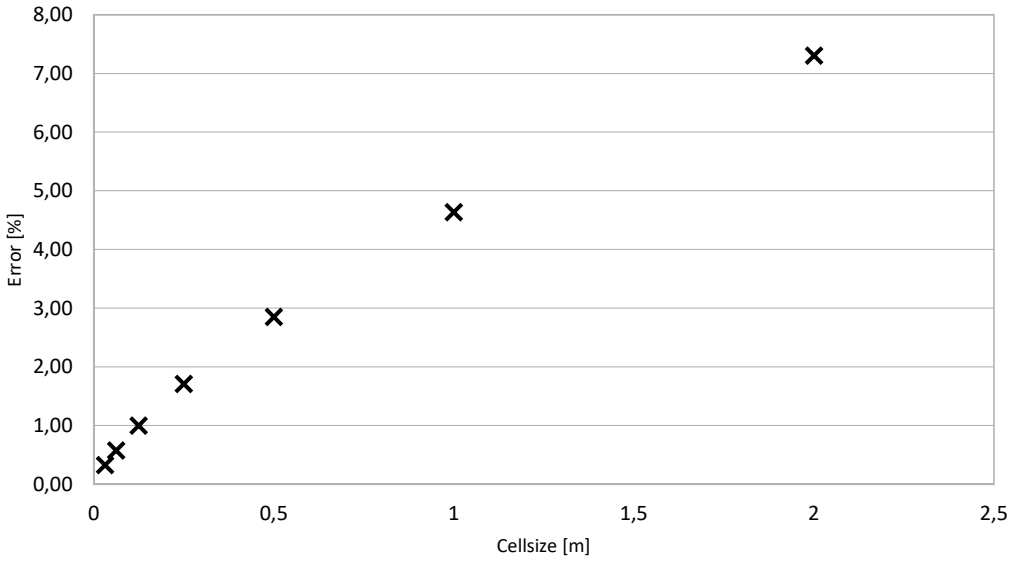

Abbildung 14: Prozentualer Fehler für die diagonale Entfernung des Fast Marching Algorithmus in Abhängigkeit der Zellbreite.

Übersichtlichkeit wurde auf Abbildungen, welche das Ende des Tests mit dem Dijkstra bzw. Fast-Marching Algorithmus zeigen, verzichtet. Außerdem konnte festgestellt werden, dass die Simulationsdauer durch die Verwendung des Fast-Marching Algorithmus mit ca. $34,33s$ minimal kürzer als bei der Verwendung des Dijkstra Alogirithmus mit ca. $34,47s$ war.

8.1.3 Evakuierung

Mit Hilfe der verschiedenen Evakuierungstests sollen die Auswirkungen unterschiedlicher Positionen und Anzahl von Türen auf die Dauer der Evakuierung eines Raumes ermittelt werden. Die Berechnung des Zielnutzens auf Basis des Euklid Algorithmus wird hierbei nicht berücksichtigt. Dieser Algorithmus ist zwar gut geeignet um den kürzesten Weg zu einem Ziel zu finden, Hindernisse können jedoch nicht umgangen werden. Somit kann nicht ausgeschlossen werden, dass eine Person, beispielsweise in einer Ecke, „hängen bleibt“. Dieses Verhalten würde nicht der Realität entsprechen.

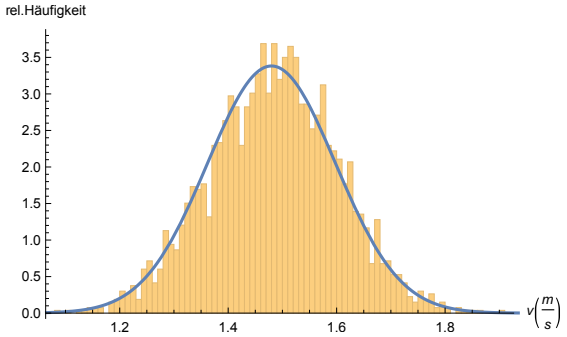

Abbildung 15: Verteilung der Geschwindigkeiten von Personen

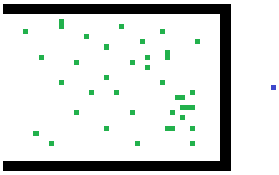

Abbildung 16: Ausgangsfeld des Hühnertests

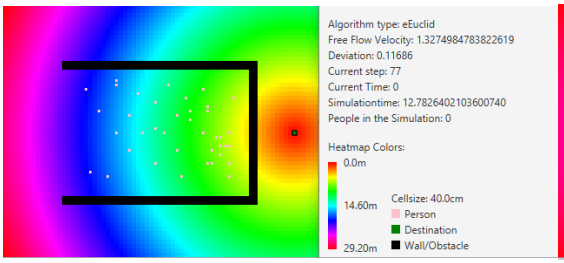

Abbildung 17: Heatmap des Hühnertests mit dem Euklid Algorithmus (zu Beginn)

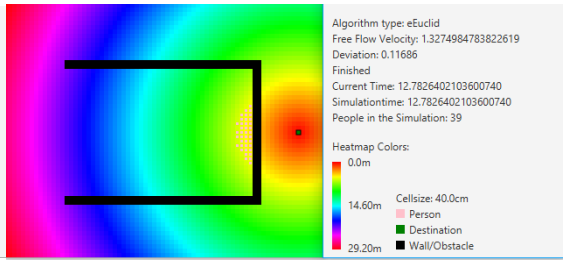

Abbildung 18: Heatmap des Hühnertests mit dem Euklid Algorithmus (Ende des Tests)

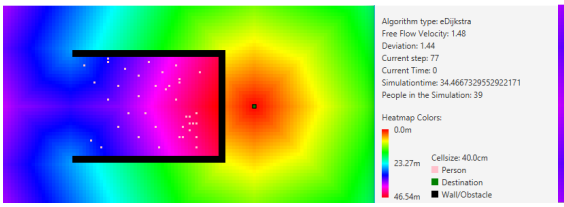

Abbildung 19: Heatmap des Hühnertests mit dem Dijkstra Algorithmus (zu Beginn)

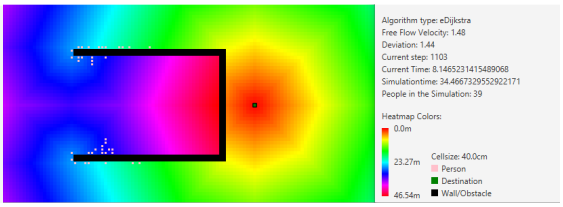

Abbildung 20: Heatmap des Hühnertests mit dem Dijkstra Algorithmus (während der Simulation)

8.1.3.1 Evakuierung eines Raumes (Personen befinden sich in der Mitte) Für die Verifikation der Simulation sollen sich die Personen zunächst alle dicht gedrängt, symmetrisch in der Mitte des Raumes befinden. An 2 gegenüberliegenden Seiten des Raumes soll sich jeweils eine Tür befinden (Evakuierung eines Raumes mit 2 Türen). In einer weiteren Simulation sollen sich an den anderen beiden Seiten jeweils eine weitere Tür befinden (Evakuierung eines Raumes mit 4 Türen). Die Evakuierungsdauer des Raumes mit 4 Türen müsste nun exakt halb so lange sein, wie die des Raumes mit 2 Türen.

Um eine möglichst symmetrische Anordnung der Personen sicher zu stellen, wurden alle Zellen des Raumes mit Personen belegt. Es wurden Evakuierungen mit 676 Personen simuliert. Die beiden Räume, für die im weiteren Kapitel die Evakuierungszeiten verglichen

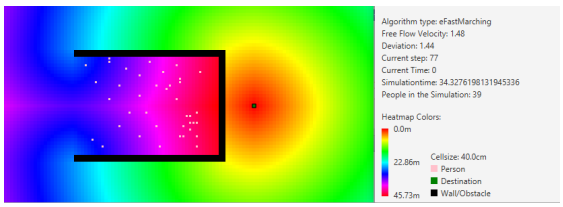

Abbildung 21: Heatmap des Hühnertests mit dem Fast-Marching Algorithmus (zu Beginn)

werden, sind in Abbildung 22 dargestellt. Für die Evakuierungsdauer ist nicht die gesamte Simulationsdauer (Zeit bis die letzte Person das Ziel erreicht), sondern die Zeit, bis die letzte Person den Raum verlässt relevant. Dieser Bereich wurde in den Abbildungen mit einem roten Rahmen hervorgehoben. Alle Szenarien wurden zunächst auf Basis des Dijkstra Algortihmus simuliert. Die Abbildung 23 zeigt die Heatmap der Szenarien. Der symmetrische Nutzenanstieg in zwei bzw. vier Richtungen ist durch den Farbverlauf dargestellt.

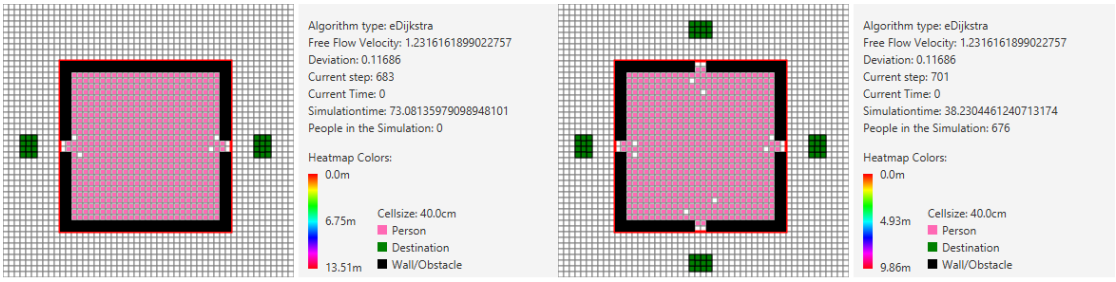

Abbildung 22: Evakuierung eines Raumes mit 2 Türen (links) und mit 4 Türen (rechts)

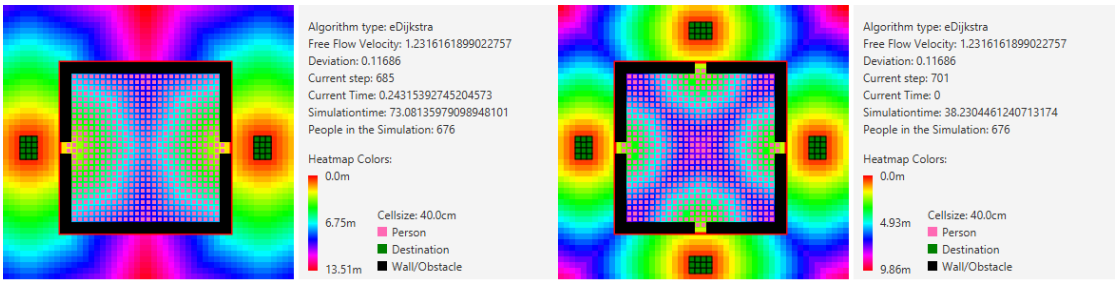

Abbildung 23: Evakuierung eines Raumes mit 2 Türen (links) und mit 4 Türen (rechts), Heatmap

In Abbildung 24 ist ein Ausschnitt der Simulationen nach einer Simulationszeit von ca. $24,1s$ (links) bzw. $15,5s$ (rechts) dargestellt. Es ist deutlich zu erkennen, dass sich jeweils ca. die Hälfte (2 Türen) bzw. ein Viertel der Personen (4 Türen) auf jede Tür zubewegen.

Abschließend zeigt Abbildung 25 die Evakuierungsdauer der beiden Räume. Die entsprechenden Ausschnitte zeigen den Moment, in dem die letzte Person den rot markierten Bereich verlassen hat. Bei der Simulation mit 2 Türen ist das nach ca. $73,07s$, bei der Simulation mit 4 Türen nach ca. $36,98s$. Durch die beiden zusätzlichen Türen verläuft die Evakuierung somit um das $1,98$ fache, also fast das 2 fache schneller. Diese Übereinstimmung mit der Theorie kann als Indiz für die Richtigkeit der Simulation angesehen werden.

Die Abweichung ist zum einen dadurch zu erklären, dass die Geschwindigkeiten der Personen zufällig verteilt sind. Es kann also vorkommen, dass sich viele, etwas schneller

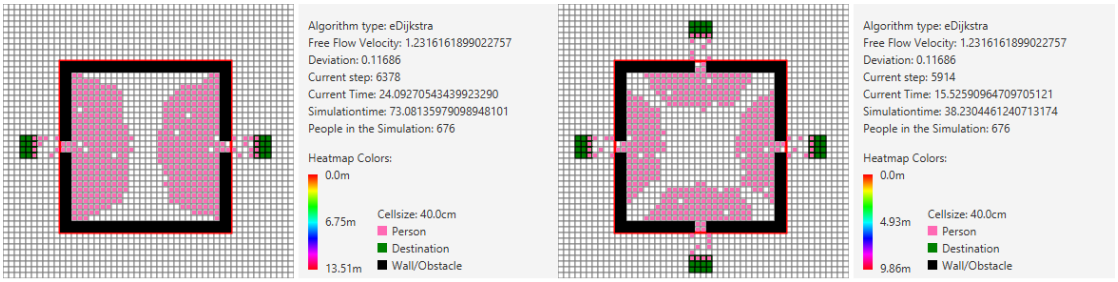

Abbildung 24: Evakuierung eines Raumes mit 2 Türen (links) und mit 4 Türen (rechts), Ausschnitt

gehende Personen im linken Teil des Feldes befinden und langsamer gehende rechts. Folglich würde die Evakuierung des linken Teils nicht so lange dauern, wie die des rechten und die anfängliche Symmetrie wäre nicht mehr vorhanden. Eine weitere Ursache könnte die Gestaltung der Implementierung sein. Eine Person, welche aktuell nicht ziehen kann (Felder mit größerem Nutzen als aktuell besetztes Feld sind belegt) wartet so lange, wie sie bräuchte um zum gewünschten Feld zu gehen, bis sie erneut versucht zu gehen. Das Verhalten ist unabhängig davon, ob das gewünschte Feld früher schon frei wird oder nicht. Somit warten Personen, obwohl sie theoretisch schon gehen könnten.

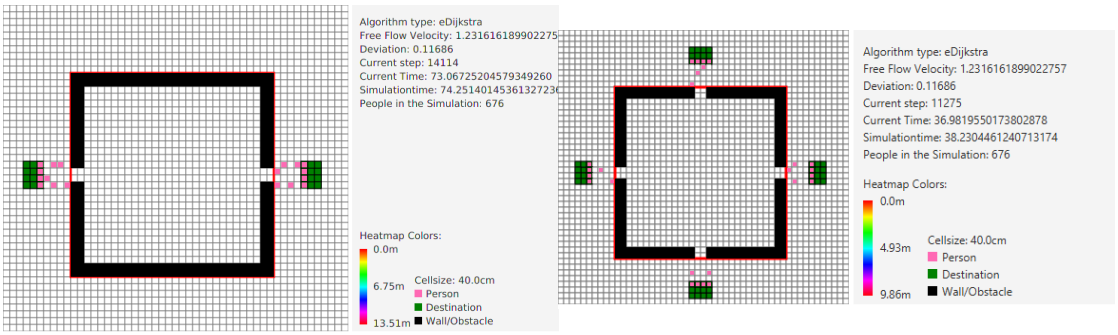

Abbildung 25: Evakuierung eines Raumes mit 2 Türen (links) und mit 4 Türen (rechts), Ende der Evakuierung

Zum Vergleich zeigt Abbildung 26 die Evakuierungszeiten des identischen Raumes auf Basis des Fast-Marching Algorithmus. Auch hier ist die Dauer der Evakuierung bei zwei zusätzlichen Türen um den Faktor $1,97$ kürzer.

Stellt man die erläuterten Evakuierungsszenarien in Bezug zur Realität müssen einige Punkte beachtet werden. Zum einen befinden sich in der Realität niemals so viele Personen symmetrisch in einem Raum. Zum anderen ist es aufgrund von Brandschutz und anderen Sicherheitsvorgaben nicht zulässig 676 Personen in einen ca. $108qm$ großen Raum zu lassen. Des weiteren wurden menschliche Faktoren wie Orientierungslosigkeit oder Panik in der Simulation nicht berücksichtigt. Die geringen Evakuierungszeiten sind überwiegend

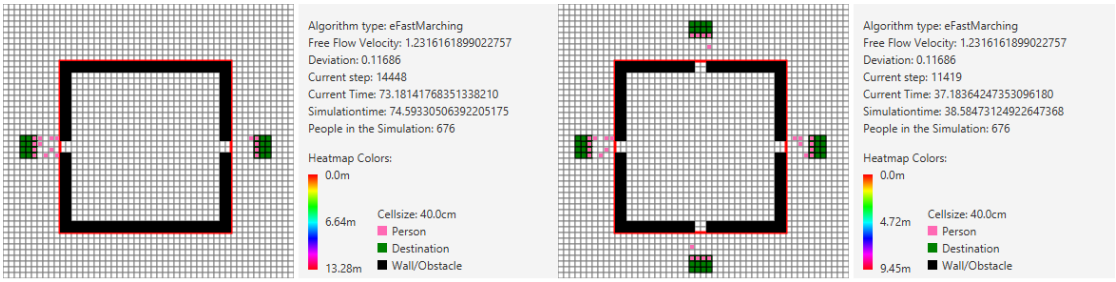

Abbildung 26: Evakuierung eines Raumes mit 2 Türen (links) und mit 4 Türen (rechts), Fast-Marching Algorithmus (Ende der Evakuierung)

darauf zurückzuführen, dass jede Person zu jedem Zeitpunkt die perfekte Orientierung hat und genau die Zellen betritt, die auf dem für sie effizientesten Weg zum Ziel liegen. Diese Simulationen sind somit zusammenfassend als theoretisches Konstrukt zu sehen. Anhand dessen kann die Simulation gegen die Anforderung, dass sich die Evakuierungszeit bei Verdopplung der Türen Halbieren muss, verifiziert werden.

8.1.3.2 Evakuierung eines Raumes (Personen zufällig verteilt) In diesem Abschnitt werden weitere Evakuierungstests gegenübergestellt, bei denen die Personen zufällig im Raum verteilt wurden. Anhand dieser Tests sollen die Auswirkungen unterschiedlicher Positionen der Türen auf die Evakuierungsdauer simuliert werden.

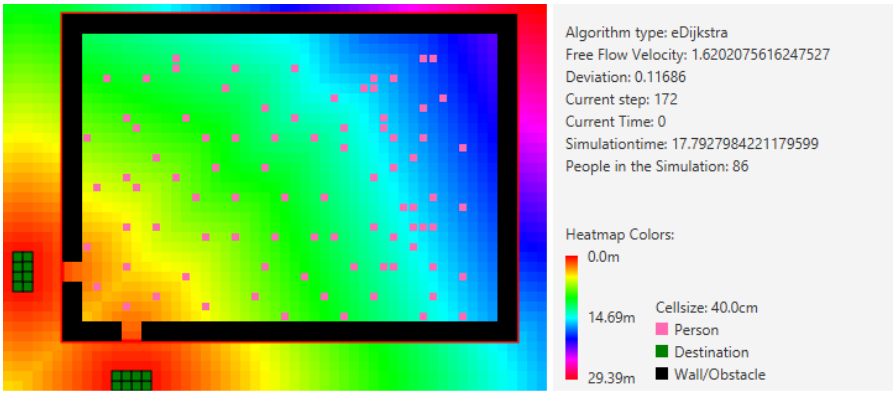

Abbildung 27: Verteilung der Personen (Heatmap Dijkstra Algorithmus)

Abbildung 27 zeigt die Verteilung der Personen, die für unterschiedliche Evakuierungstests verwendet wird. Somit wird eine bessere Vergleichbarkeit der Ergebnisse erzielt. In der Abbildung ist außerdem die Heatmap des Dijkstra Algorithmus für dieses Szenario zu sehen.

Die Abbildung 28 zeigt einen Ausschnitt der Simulation nach einer Simulationszeit von

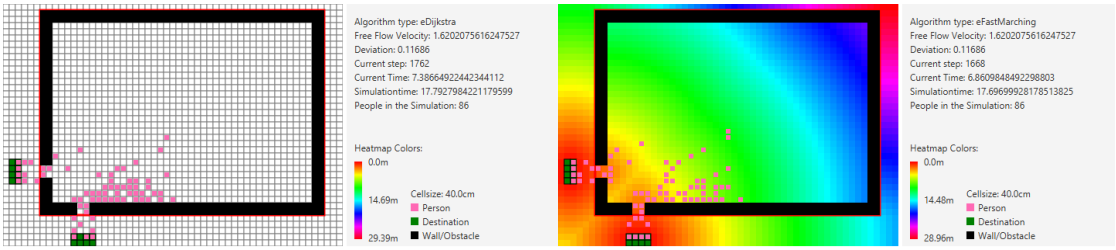

Abbildung 28: Evakuierung eines Raumes mit 2 Türen (geringer Abstand), Dijkstra Algorithmus (links), Fast-Marching Algorithmus (rechts)

ca. $6s$. Wie zu erwarten ist, drängen alle Personen in die Ecke links unten. Dort bilden sich vor den Engstellen (Ausgängen) kleinere Staus. Die Evakuierungsdauer beträgt für die Simulation mit dem Dijkstra Algorithmus ca. $16,8s$. Da sich die Personen bei Verwendung des Fast Marching Algorithmus effizienter auf die Ziele zu bewegen, bilden sich etwas kleinere Staus an den Engstellen und die Evakuierungsdauer ist mit ca. $16,5s$ etwas kürzer.

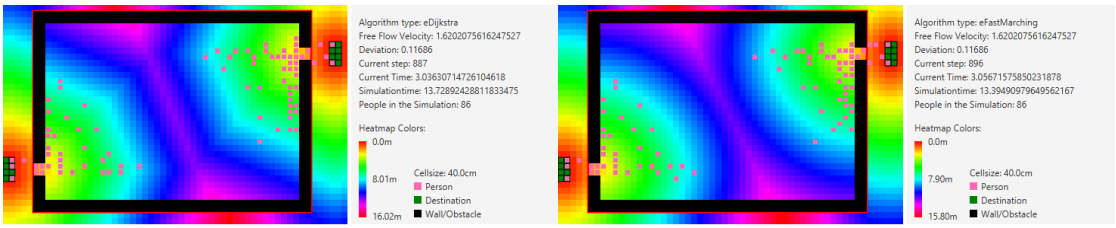

Abbildung 29: Evakuierung eines Raumes mit 2 Türen (großer Abstand), Dijkstra Algorithmus (links), Fast-Marching Algorithmus (rechts)

Im Vergleich dazu zeigt die Abbildung 29 einen Ausschnitt der Simulation, in der die Ausgänge bzw. Ziele sehr weit voneinander entfernt gewählt wurden. Beide Algorithmen werden gegenübergestellt. In der Abbildung ist sofort zu sehen, dass sich in etwa die Hälfte der Personen (Personen deren Startposition in der entsprechenden Hälfte lag) auf die Ziele unten links und die andere Hälfte auf die Ziele oben rechts zubewegen. Die Heatmaps zeigen die entsprechende Entfernung an. Die maximale Entfernung, die eine Person von einer Tür haben kann, ist in der Simulation mit den nahe beieinander liegenden Türen fast doppelt so groß (ca. $28m$), wie in der Simulation in der die Türen weit voneinander entfernt sind (ca. $15m$).
Die Evakuierungsdauer ist mit ca. $12,53s$ (Dijkstra) bzw. $12,6s$ (Fast-Marching) deutlich kürzer als im zuvor beschrieben Szenario (vgl. Abbildung 28). Um diesen Zusammenhang weiter zu überprüfen, wurde eine zusätzliche Map mit deutlich mehr Personen im Raum angelegt. Die Positionen der Türen wurde nicht verändert. Da hierbei vor allem die Auswirkung der Position der Türen auf die Evakuierungsdauer ermittelt werden soll, wurde dieser zusätzliche Test nur auf Basis des Dijkstra Algorithmus durchgeführt. Die Implementierung der Simulation ermöglicht jedoch, diesen bei Bedarf auf Basis des

Fast-Marching Algorithmus zu wiederholen.

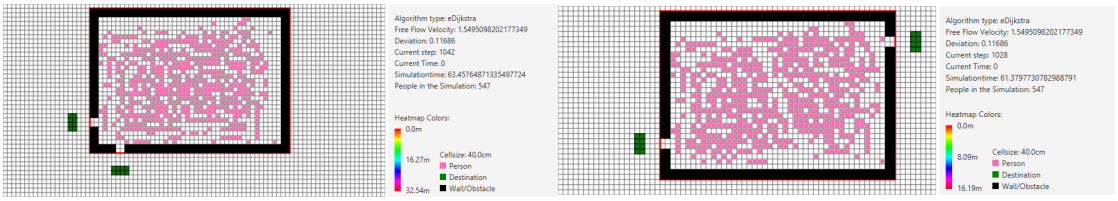

Abbildung 30: Evakuierung eines Raumes mit 2 Türen, links: geringer Abstand zwischen den Türen, rechts: großer Abstand zwischen den Türen

Abbildung 30 zeigt die Aufteilung der 547 Personen und die Positionen der Türen in den beiden Szenarien.

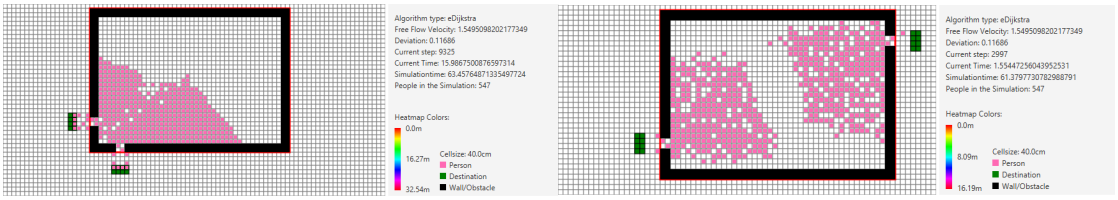

Abbildung 31: Evakuierung eines Raumes mit 2 Türen (Ausschnitt der Simulation), links: geringer Abstand zwischen den Türen, rechts: großer Abstand zwischen den Türen

Die Abbildung 31 zeigt einen Ausschnitt der Simulation bei $15,97s$ (links) bzw. $1,55s$ (rechts). In beiden Abbildungen ist deutlich zu sehen, dass sich die Personen, aufgrund der hohen Personenanzahl, vor allen Türen stauen.

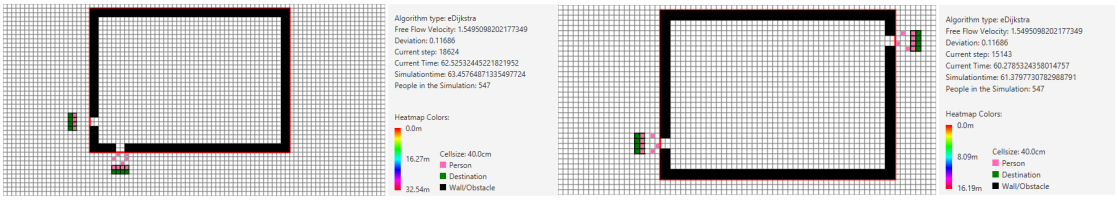

Abbildung 32: Evakuierung eines Raumes mit 2 Türen (Ausschnitt: Ende der Evakuierung), links: geringer Abstand zwischen den Türen, rechts: großer Abstand zwischen den Türen

Abschließend zeigt die Abbildung 32 das Ende der Evakuierung der beiden Räume. Die Evakuierung eines Raumes, bei dem die Türen weit voneinander entfernt sind, verläuft mit einer Dauer von ca. $60,28s$ um ca. $2,3s$ schneller als die eines Raumes, bei dem die Türen nah beieinander liegen ($62,53s$). Durch eine höhere Anzahl an Personen, wirkt sich die Position der Türen somit nur noch sehr gering auf die Evakuierungsdauer aus.

Hintergrund hierfür könnte die lange Wartezeit bzw. der große Stau vor den jeweiligen Türen sein. Da die Wartezeit der einzelnen Personen teilweise sehr hoch ist, wirkt sich der längere Weg bis zur Tür kaum auf die Evakuierungsdauer aus.

8.1.3.3 Fazit und Vergleich

8.2 Validation

9 Ausblick und Fazit

Literatur

[1] Oracle Documentation. Java parallelism, https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html, 2017.