REPRÄSENTATION UND SIMULATION VON UMGEBUNGEN FÜR MULTIAGENTENSYSTEME

STEFAN TITTEL

Diplomarbeit

Fakultät für Informatik
Lehrstuhl 1 – Logik in der Informatik
Technische Universität Dortmund

Oktober 2010

Betreuer:
Prof. Dr. Gabriele Kern-Isberner
Dipl.-Inf. Matthias Thimm

Stefan Tittel: *Repräsentation und Simulation von Umgebungen für Multiagentensysteme,* Diplomarbeit, © Oktober 2010

ZUSAMMENFASSUNG

Diese Diplomarbeit umfasst die Erstellung eines Konzepts und einer Implementierung für ein flexibles Framework zur Repräsentation und Simulation von Umgebungen für Multiagentensysteme. Mit Hilfe dieses Frameworks lassen sich Umgebungen spezifizieren und simulieren, die als Experimentierplattform für verschiedene (vor allem wissensbasierte) Agenten dienen. Das Framework dient dabei der Realisierung von Umgebungen, in welchen Agenten auf einem Grid verortet sind, und bietet Funktionen für die Interaktion von Agenten mit der Umgebung und mit anderen Agenten in der Umgebung. Dabei lassen sich die Wahrnehmungen der Agenten abhängig von den Eigenschaften der Agenten und sichtblockierenden Hindernissen einschränken. Ebenso wird die Kommunikation mit anderen Agenten unterstützt, wobei auch diese Einschränkungen unterworfen werden kann. Das Framework wird dabei vor allem mit Hilfe eines neu eingeführten und auf STRIPS-Notation basierenden Formalismus beschrieben.

Neben dem Framework selbst werden zudem grafische Clients zum Testen und zur Beobachtung von Umgebungen bereitgestellt. Am Ende der Arbeit erfolgt eine ausführliche Beschreibung, wie mit Hilfe des Frameworks Umgebungen spezifiziert und entwickelt werden können.

DANKSAGUNG

Ich danke Prof. Dr. Gabriele Kern-Isberner und Dipl.-Inf. Matthias Thimm für die Betreuung meiner Arbeit. Darüber hinaus danke ich Bernhard Groll und Gabriele Tittel für ihre Hilfe beim Korrekturlesen.

INHALTSVERZEICHNIS

1 Einleitung 1
1.1 Umgebungen in Multiagentensystemen 1
1.2 Motivation 2
1.3 Zielsetzung und Aufbau der Arbeit 3

2 Grundlagen 5
2.1 Eigenschaften von Umgebungen 5
2.1.1 Zugänglichkeit 5
2.1.2 Determinismus 5
2.1.3 Sequenzialität 5
2.1.4 Dynamik 6
2.1.5 Kontinuität 6
2.2 Bestehende Ansätze 7
2.2.1 ELMS 7
2.2.2 MASSim 10

3 Zielsetzung 13
3.1 Konzeptionelle Ziele 13
3.1.1 Unterstützte Eigenschaften 13
3.1.2 Weitere Funktionalität 15
3.1.3 Umgebungszustand und Zustandsüberführung 18
3.2 Implementierung 18
3.2.1 Kommunikation 19
3.2.2 Spezifikation 20

4 Umgebungszustand und Zustandsüberführung 21
4.1 Einleitung 21
4.1.1 Prototypisches Beispiel 21
4.1.2 Zustand und Zustandsüberführung 22
4.1.3 Kapazitäten und Verschiebekraft 25
4.1.4 Übergabe von Gegenständen 26
4.1.5 Diagonalkorrektur 28
4.2 Aufbau von Umgebungszuständen 29
4.3 Integritätsbedingungen 31
4.4 Aktionsanforderungen 32
4.5 Funktionen 34
4.6 Vorgehen zur Zustandsüberführung 35
4.7 Zustandsüberführungsregeln 38
4.7.1 Aufbau 38
4.7.2 Regeln 40
4.8 Interne Einflüsse 56

4.9 Terminierung 58
4.10 Überlegungen zur Modellierung 58
4.10.1 Turingvollständigkeit 58
4.10.2 Funktionen und Regeln 60
4.11 Erweiterte Funktionalität 60
4.11.1 Nebel 61
4.11.2 Schließfächer 64
4.11.3 Funktionalitätsumfang und Erweiterungen 66
4.12 Auslösung von Zeitfortschritt 67

5 Wahrnehmung 69
5.1 Einleitung 69
5.1.1 Prototypisches Beispiel 69
5.1.2 Hindernisse und Sichtweite 70
5.2 Sichtbehindernde Hindernisse 70
5.2.1 Realweltliche Sichtlinien 70
5.2.2 Diskrete Sichtlinien 74
5.2.3 Wahrnehmung bzgl. Hindernissen 77
5.3 Sichtweite 80
5.4 Bestimmung der Beobachtungsmenge 81

6 Kommunikation 85
6.1 Einleitung 85
6.1.1 Prototypisches Beispiel 85
6.1.2 Allgemeines Vorgehen 86
6.2 Nachrichtentypen 86
6.3 Kommunikationsblockierende Hindernisse 89
6.4 Empfangsempfindlichkeit und Sendestärke 90
6.5 Erzeugung von Nachrichten 90
6.5.1 Private Nachrichten 91
6.5.2 Öffentliche Nachrichten 91
6.6 Wahrnehmung von Nachrichten 93
6.7 Entfernung alter Nachrichten 93

7 Implementierung 97
7.1 Inhalt der CD-ROM 97
7.2 Eigenschaften und Einordnung 99
7.2.1 Umgebungszustand 99
7.2.2 Integritätsbedingungen 99
7.2.3 Zustandsüberführungsregeln 100
7.2.4 Wahrnehmung 101
7.2.5 Kommunikation 101
7.2.6 Ausführungsreihenfolge 102
7.2.7 Hinzufügen und Entfernen von Agenten und Objekten 103
7.2.8 Parameter und Properties 105
7.2.9 Bereitschaft zum Empfang von Objekten 106

7.2.10 Offene und geschlossene Agententypen 106
7.3 Spezifikationsformat 107
7.3.1 Sprachtypus 108
7.3.2 Beschreibung 108
7.4 Kommunikationsprotokoll 123
7.4.1 KQML 124
7.4.2 Aktionsanforderungen 124
7.4.3 Wahrnehmung 127
7.5 Clients 133
7.5.1 Agenten-Client 133
7.5.2 Beobachter-Client 137
7.6 Erweiterungen 139
7.6.1 Agent 140
7.6.2 Objekt 142
7.6.3 Regel 144
7.7 Server-API 146
7.8 Server-Implementierung 149
7.8.1 Thread-Synchronisation 149
7.8.2 Erzeugung von Datenstrukturen 150
7.8.3 Erzeugung von Wahrnehmungen 152
7.8.4 Kontrollfluss der Hauptklasse 152

8 Bewertung und Ausblick 155
8.1 Vergleich mit ELMS 155
8.2 Vergleich mit MASSim 157
8.3 Fazit und Ausblick 158

LITERATURVERZEICHNIS 160

1 EINLEITUNG

Multiagentensysteme stellen ein verbreitetes Paradigma zur verteilten Lösung von Problemen und zur Simulation sozialer Strukturen dar. Ein Multiagentensystem (MAS) ist ein Computersystem, in welchem Agenten miteinander kooperieren oder konkurrieren, um individuelle oder gemeinschaftliche Ziele zu erreichen [Kru]. Unter einem Agenten wird dabei ein Computersystem verstanden, welches in einer *Umgebung* eingebettet ist und durch autonome Aktionen in dieser Umgebung versucht, seine Ziele zu erreichen [WJ95].

1.1 UMGEBUNGEN IN MULTIAGENTENSYSTEMEN

Unter einer *Umgebung* wird im Rahmen dieser Arbeit eine simulierte Welt verstanden, welche mit den Agenten über spezielle Nachrichten kommuniziert, z. B. indem sie Agenten das jeweils sichtbare Umfeld mitteilt oder von Agenten Anweisungen über auszuführende Aktionen entgegennimmt. Dieses Begriffsverständnis ist in der Literatur nicht allgemeingültig. So verstehen einige Autoren unter dem Begriff der Umgebung die Software- oder Hardware-Infrastruktur, die das Multiagentensystem betreibt, oder die physikalische Umgebung, mit welcher Agenten (z. B. in Form von Robotern) interagieren. Näheres dazu siehe [WVH05].

Multiagentensysteme sind zum Beispiel ein geeignetes Paradigma zur Simulation des ÖPNV-Netzes einer Stadt. Jeder Agent entspräche dabei einem Stadtbewohner, der gemäß seines individuellen Mobilitätsbedürfnisses versucht, verschiedene Orte zu verschiedenen Zeiten unter Berücksichtigung seiner Präferenzen (z. B. Fahrtkosten, Fahrtzeit, Umweltfreundlichkeit) und Eigenschaften (z. B. Verfügbarkeit eines Autos, Gehbehinderung) zu erreichen. Die Umgebung würde der Modellierung der Topologie der Stadt, insbesondere des Straßen- und Schienennetzes, dienen. Ebenso könnte die Umgebung die gemäß Fahrplan vorgesehenen Busse und Bahnen realisieren, wobei für diese auch eine Modellierung als Agenten denkbar wäre. Eine solche Simulation könnte der Betreiber des ÖPNV-Netzes zur Hilfe nehmen, um die Fahrpläne des städtischen ÖPNV zu optimieren.

Die notwendigen Eigenschaften und Funktionen einer Umgebung hängen zuvorderst von der Art der zu simulierenden Welt ab. Beispielsweise wird eine Umgebung für ein Multiagenten-

system, welches den Wertpapierhandel an einer Börse simuliert, ein von Kauf- und Verkaufsangeboten dominiertes Handelssystem realisieren, während die Modellierung eines räumlichen Gebiets eher nicht zu erwarten ist. Die Umgebung für ein Multiagentensystem, dessen Agenten in Hinblick auf die Suche und Beseitigung von Staub konzipiert wurden, wird hingegen kein Handelssystem realisieren, dafür jedoch das räumliche Gebiet, in welchem sich die Agenten auf der Suche nach Staub bewegen.

Darüber hinaus kann sich je nach Art der zu simulierenden Welt die Notwendigkeit ergeben, dass die Umgebung Funktionen erfüllen muss, welche traditionell eher das verwendete MAS-Framework übernimmt. Können bspw. in der zu simulierenden Welt Agenten nicht beliebig, sondern nur abhängig vom Umgebungszustand miteinander kommunizieren, so muss sämtliche Kommunikation zwischen Agenten über die Umgebung stattfinden.[1] Die Umgebung müsste in diesem Fall die notwendigen Mittel zur Kommunikation bereitstellen.

1.2 MOTIVATION

Obwohl Umgebungen als wichtiger Bestandteil von Multiagentensystemen anerkannt sind, werden sie in MAS-Literatur oft nur implizit behandelt oder gar als gegeben vorausgesetzt. Die Behandlung von MAS-Umgebungen als Abstraktion erster Klasse findet vor allem im Bereich der *situated MAS*[2] statt (vgl. [WO07, WPM+04]). Der Mangel an expliziter und ausführlicher Behandlung von Umgebungen in allgemeiner MAS-Literatur setzt sich im Mangel an expliziter Unterstützung von Umgebungen in MAS-Frameworks (wie z. B. Jade oder Jadex) fort.

In der Praxis wird dieser Mangel häufig dadurch umgangen, dass die Umgebung als Agent realisiert wird, welcher den anderen Agenten „Umgebungsdienste“ in Form entsprechender Nachrichten bereitstellt. Zum einen ist ein solches Vorgehen jedoch konzeptionell problematisch, da eine Umgebung mitnichten ein Agent ist, zum anderen unnötig kompliziert und unflexibel in der Handhabung, da dabei eine Umgebung nicht aus Umgebungs-, sondern aus Agentensicht spezifiziert werden muss.

Dieser Umstand führte zur Entwicklung von Frameworks für Umgebungen, wie dem MASSim-Server des *Multi Agent Contests* (vgl. [DDN06]) und dem ELMS-Interpreter von Okuyama, Bordini und da Rocha Costa (vgl. [OBdRC04]). Bei ELMS werden

1 Da Agenten aus Sicht der Umgebung als nicht vertrauenswürdig angesehen werden müssen und den Agenten der Umgebungszustand u. U. gar nicht hinreichend bekannt ist, um zu entscheiden, ob die beabsichtigte Kommunikation stattfinden darf, wäre eine Selbstrestriktion seitens der Agenten keine Lösung.

2 Dabei handelt es sich um Multiagentensysteme, in denen Agenten und Objekte explizite Positionen in der Umgebung besitzen (vgl. [WH02]).

Umgebungen dabei nicht in einem Beschreibungsformat spezifiziert, sondern müssen in Java implementiert werden. Der ELMS-Ansatz ist Bestandteil des MAS-SOC-Projekts (vgl. [BOdO$^+$04]) und für die in den folgenden Abschnitten diskutierten Arten von Umgebungen nur bedingt geeignet. Zudem hat der ursprüngliche Versuch der Autoren, einen ELMS-Interpreter zu realisieren, nie den zur Veröffentlichung notwendigen Reifegrad erreicht; ein ELMS-Interpreter ist daher nicht öffentlich verfügbar.[3]

Es sei jedoch bereits vorweggenommen, dass derzeit kein öffentlich verfügbares Umgebungs-Framework existiert, welches hinsichtlich der Spezifikation von Umgebungen einfach ist und dabei gleichzeitig mächtig genug, um die in den folgenden Abschnitten diskutierten Arten von Umgebungen zu unterstützen. Dieser Mangel soll durch das im Rahmen dieser Arbeit entwickelte Framework namens GridWorldSim behoben werden.

1.3 zielsetzung und aufbau der arbeit

Ziel der Diplomarbeit soll die Konzeption und Implementierung eines flexiblen Frameworks zur Realisierung von Umgebungen für Multiagentensysteme sein. Das Framework soll dabei primär der Realisierung von Umgebungen dienen, in denen Agenten und Objekte in einem zweidimensionalen räumlichen Gebiet verortet sind, welches durch ein Gitternetz in (Grid-)Zellen unterteilt ist. Dabei ist jedem Agenten oder Objekt genau eine Zelle zugeordnet. Eine Zelle kann hingegen i. A. mehrere Agenten oder Objekte aufnehmen. Eine solche Umgebung sei *Gridwelt* genannt.[4]

Da eine Unterstützung aller denkbaren Welten unmöglich wäre, ist es notwendig, zunächst festzulegen, welche Arten von Umgebungen unterstützt werden sollen und welche Eigenschaften das zu entwickelnde Framework dazu besitzen und welche Funktionen es dazu bereitstellen muss. Grundlegende Eigenschaften von Umgebungen und bestehende Ansätze werden dazu in Kapitel 2 diskutiert. Basierend darauf konkretisiert Kapitel 3 die Zielsetzung dieser Arbeit.

Kapitel 4 erläutert das Vorgehen bei der Beschreibung und Überführung von Umgebungszuständen und spezifiziert das Verhalten der Umgebung hinsichtlich Aktionsanforderungen, Objekten und Umwelteinflüssen.

---

3 Die derzeitige Version des ELMS-Interpreters wird von den Autoren nicht mehr gepflegt oder weiterentwickelt; es ist jedoch eine komplette Neuentwicklung geplant, die im Jahr 2011 beginnen soll.

4 Ausführliche Beispiele für Gridwelten finden sich in den Szenarien „Hunter Prey", „Cleanerworld" und „Garbage Collector" unter `http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/Usages/Examples`.

In Kapitel 5 wird beschrieben, unter welchen Umständen die Agenten bestimmte Aspekte der Umgebung wahrnehmen können und wie diese Wahrnehmung bestimmt wird.

Kommunikation zwischen Agenten findet über die Umgebung statt. Die dazu notwendigen Mechanismen werden in Kapitel 6 erläutert.

Kapitel 7 behandelt alle Aspekte der Implementierung des im Rahmen dieser Arbeit erstellten Frameworks, insbesondere das Spezifikationsformat für Umgebungen, die Kommunikationsprotokolle zur Kommunikation mit Agenten und die Möglichkeiten, das Framework um eigene Komponenten zu erweitern.

Eine Einordnung der Arbeit und einen Ausblick auf mögliche Erweiterungen gibt Kapitel 8.

2 GRUNDLAGEN

In diesem Kapitel sollen die grundlegenden Eigenschaften von Umgebungen erläutert werden. Diese werden im nächsten Kapitel dazu verwendet, die von GridWorldSim bereitgestelle Funktionalität einzuordnen. Ebenso werden bestehende Ansätze zur Realisierung von Frameworks für Umgebungen vorgestellt.

2.1 EIGENSCHAFTEN VON UMGEBUNGEN

Umgebungen können gemäß der nachfolgend beschriebenen Eigenschaften kategorisiert werden (vgl. [RN09]).

2.1.1 *Zugänglichkeit*

Wenn jeder Agent die Umgebung immer vollständig wahrnehmen kann, handelt es sich um eine *vollständig wahrnehmbare Umgebung*, anderenfalls um eine *partiell wahrnehmbare Umgebung*. In vielen Anwendungsfällen kann ein Agent die Umgebung nicht vollständig wahrnehmen, z. B. weil seine Sicht durch Hindernisse blockiert ist.

2.1.2 *Determinismus*

Hängt der nächste Zustand der Umgebung ausschließlich vom aktuellen Zustand und den zum Zeitpunkt der Zustandsüberführung vorliegenden Aktionsanforderungen von Agenten ab, so liegt eine *deterministische Umgebung* vor, anderenfalls wird die Umgebung als *stochastisch* bezeichnet.[1]

2.1.3 *Sequenzialität*

Eine *episodische Umgebung* unterteilt das Geschehen in der Umgebung in atomare Episoden, in denen ein Agent einen Umgebungszustand empfängt und daraufhin genau eine Aktion durchführt. Der Umgebungszustand in der nächsten Episode hängt dabei nicht von den Aktionen eines Agenten in vorhergehenden Epi-

1 [RN09] bezeichnet Umgebungen im Gegensatz zur hier verwendeten Definition nur dann als deterministisch, wenn der nächste Zustand ausschließlich vom aktuellen Zustand und der Aktion *eines* Agenten abhängt.

soden ab. Umgebungen, in denen Aktionen die nachfolgenden Umgebungszustände beeinflussen, heißen *sequentiell*.

Umgebungen sind häufig dann episodisch, wenn sie für Agenten entworfen wurden, die Klassifizierungsaufgaben durchführen. Ein Beispiel für eine solche Umgebung ist eine Umgebung, die ein Fließband mit Bauteilen realisiert, welche von einem Agenten sukzessiv als fehlerfrei oder fehlerhaft klassifiziert werden. Das Vorliegen des aktuellen Bauteils sowie die anschließende Klassifikation durch den Agenten stellen dabei eine Episode dar, wobei die aktuelle Klassifikation durch den Agenten keinen Einfluss darauf hat, ob das nächste Bauteil auf dem Fließband fehlerfrei oder fehlerhaft ist.

2.1.4 *Dynamik*

Wenn sich der Umgebungszustand ändern kann, während sich ein Agent seine nächste Aktion überlegt, dann ist die Umgebung für diesen Agenten *dynamisch*, anderenfalls *statisch*.

2.1.5 *Kontinuität*

Es ist zwischen *diskreten* und *kontinuierlichen* Umgebungen zu unterscheiden, wobei vier verschiedene Bereiche zu betrachten sind: Der Umgebungszustand, die Art der Zeitmodellierung, die möglichen Beobachtungen und die möglichen Aktionen, die Agenten durchführen können. Sind alle vier Bereiche diskret, so handelt es sich um eine *volldiskrete* Umgebung, anderenfalls um eine *(teil)kontinuierliche Umgebung*.

Eine Umgebung, die z. B. Schachspiele ohne Schachuhr realisiert, besitzt abzählbar (und sogar endlich) viele Umgebungszustände. Basiert die Zeitmodellierung auf Zügen im Spiel, so sind Zeitpunkte ebenfalls abzählbar. Ebenso abzählbar (und sogar endlich) sind die Menge der möglichen Beobachtungen (Schachbrettkonfigurationen) und die Menge möglicher Aktionen (Züge). Eine solche Umgebung wäre damit volldiskret.

2.1.5.1 *Proaktivität*

In einer *proaktiven Umgebung* ist es möglich, dass ein Umgebungszustand modifiziert wird, ohne dass Aktionen von Agenten dafür vorliegen müssen (z. B. zur Realisierung von Umwelteinflüssen). Anderenfalls handelt es sich um eine *reaktive Umgebung*, welche ihren Umgebungszustand nur als Reaktion auf die Aktionen von Agenten ändert.[2]

2 Es wird hier davon ausgegangen, dass das reine Fortschreiten von Zeit keine Änderung des Umgebungszustands darstellt.

Reaktiv/proaktiv und deterministisch/stochastisch (siehe Kapitel 2.1.2) sind bzgl. ihrer Definitionen ähnlich. Der Unterschied besteht darin, dass bei stochastischen Umgebungen die Aktionsanforderung eines Agenten auch dann nicht zwangsläufig zu der gewünschten Aktion führt, wenn die Vorbedingungen der Aktion erfüllt sind, sondern der Erfolg einer Aktionsanforderung zusätzlich von einer Wahrscheinlichkeitsfunktion abhängt. Dennoch kann in einer solchen Umgebung jede Zustandsänderung von einer Aktionsanforderung eines Agenten ausgelöst werden, die Umgebung wäre dann stochastisch, aber nicht proaktiv. Im Gegensatz dazu kann eine proaktive Umgebung auch dann Zustandsänderungen vornehmen, wenn dieser Zustandsänderung keine Aktionsanforderung eines Agenten zu Grunde liegt.

2.2 bestehende ansätze

Im Folgenden werden zwei bestehende Ansätze zur Realisierung von Umgebungen für Multiagentensysteme erläutert.

2.2.1 *ELMS*

Die *Environment Description Language for Multi-Agent Simulation (ELMS)* (vgl. [OBdRC04]) ist eine XML-Sprache zur Spezifikation von Umgebungen für Multiagentensysteme. Die Sprache ist Teil des Projektes *Multi-Agent Simulation for the Social Sciences (MAS-SOC)* (vgl. [BOdO+04]) zum Design und zur Implementierung von Multi-Agenten-Simulationen. Es existiert ein Prototyp eines ELMS-Interpreters, welcher basierend auf einer in ELMS gegebenen Umgebungsspezifikation dem Multiagentensystem die zugehörige Umgebung zur Verfügung stellt. ELMS bietet dabei Unterstützung für Gridwelten, die Verwendung eines Grids ist im Gegensatz zu GridWorldSim jedoch optional.

Die Nachfrage bei einem der Autoren, Fabio Y. Okuyama, ergab jedoch, dass dieser Prototyp sich nicht in einem zur Veröffentlichung geeigneten Stadium befindet und dass wegen größerer Änderungen an MAS-SOC eine Weiterentwicklung des Prototyps nicht geplant sei. Die komplette Neuentwicklung eines ELMS-Interpreters wurde für das Jahr 2011 in Aussicht gestellt, derzeit ist jedoch kein ELMS-Interpreter öffentlich verfügbar.

Das Ziel des MAS-SOC-Projekts besteht in der Bereitstellung eines Frameworks zur Erstellung von agentenbasierten sozialen Simulationen. Dabei soll das Multiagentensystem, einschließlich Agenten und Umgebung, vom Benutzer durch eine grafische Benutzerschnittstelle spezifiziert werden, ohne dass tiefergehende Programmierkenntnisse erforderlich wären. ELMS ist daher primär als Zwischensprache intendiert, da ELMS-Spezifikationen

i. d. R. durch ein grafisches Frontend automatisiert erzeugt werden; dennoch ist eine händische Spezifikation mittels XML- oder Text-Editors grundsätzlich möglich.

ELMS unterstützt partiell wahrnehmbare, stochastische, sequentielle, dynamische, diskrete[3] und reaktive Umgebungen.

2.2.1.1 *Sprachkonstrukte*

ELMS besteht aus den folgenden Sprachkonstrukten:

1. Spezifikation von Agenten
    - *Agentenrumpf (agent body)*: Ein Agentenrumpf spezifiziert einen Agententyp bestehend aus einem Namen, einer Liste von Attributen, einer Liste möglicher Aktionen und einer Liste möglicher Wahrnehmungen. Die Attribute charakterisieren dabei die wahrnehmbaren Eigenschaften des Agenten.
    - *Wahrnehmung (perception)*: Eine Wahrnehmung besteht aus einem Namen, einer optionalen Liste von Vorbedingungen und einer Liste von wahrnehmbaren Eigenschaften. Wenn ein Agententyp über eine bestimmte Wahrnehmung verfügt und alle Vorbedingungen der Wahrnehmung erfüllt sind, teilt die Umgebung dem Agenten den Zustand der wahrnehmbaren Eigenschaften mit.
    - *Aktion (action)*: Eine Aktion besteht aus einem Namen, einer optionalen Liste von Parametern, einer optionalen Liste von Vorbedingungen und einer Befehlssequenz, welche festlegt, wie sich die Umgebung bei Ausführung der Aktion ändert. Möglicherweise für eine Aktion vorhandene Parameter ermöglichen es dem Agenten, die von ihm angeforderte Aktion näher spezifizieren zu können. Die Befehlssequenz besteht aus beliebigen Zuweisungen von Attributwerten von bspw. Agenten oder Objekten, ebenso können Objekte erschaffen oder zerstört werden. Wenn ein Agent eine Aktion angefordert hat und alle Vorbedingungen erfüllt sind, wird die Befehlssequenz von der Umgebung atomar ausgeführt.
2. Spezifikation von Umgebungen
    - *Grid-Optionen*: ELMS unterstützt optional zwei- oder dreidimensionale Grids. Eine Gridspezifikation besteht

3 [OBdRC04] erwähnt zwar die Möglichkeit kontinuierlicher Umgebungen, erlaubt jedoch keine Datentypen beliebiger Präzision.

aus der Angabe der x-, y- und z-Dimension sowie Attributen und Reaktionen, die allen Gridzellen gemein sind.

- *Gegenstände (resources)*: Gegenstände sind bei ELMS rein reaktiv. Die Definition einer Objektklasse beinhaltet den Klassennamen, eine Liste von Attributen und eine Menge von Reaktionen.
- *Reaktionen (reactions)*: Analog zu Aktionen bestehen Reaktionen aus einem Namen, einer Liste von Vorbedingungen und einer Befehlssequenz. Der Unterschied zu Aktionen besteht darin, dass zum einen eine explizite Aktionsanforderung zur Auslösung nicht notwendig ist, zum anderen, dass nicht nur eine Aktion pro Agent ausgeführt wird, sondern alle Reaktionen mit erfüllten Vorbedingungen gleichzeitig aktiv werden.

3. Allgemeine Sprachkonstrukte
   - Es bestehen Konstrukte zur Spezifikation, welche Eigenschaften der Umgebung an die MAS-SOC-Schnittstelle übermittelt werden, zur Initalisierung des Umgebungszustands und zur Angabe, welche Attributwerte Teil der Simulation sind, um die Erstellung von Snapshots der Simulation zu vereinfachen.
   - *Attribut-Definitionen*: Für Attribute werden von ELMS die Datentypen Boolean, Integer, Float und String unterstützt. Jedes Attribut besitzt einen Typen und einen initialen Wert, wobei der initiale Wert im Fall von Boolean, Integer oder Float nicht nur eine Konstante, sondern auch ein Ausdruck sein kann.
   - *Ausdrücke (expressions)*: ELMS unterstützt einige algebraische, logische und relationale Operatoren, mit deren Hilfe Ausdrücke definiert werden können.
     - relationale Operatoren: $=, \neq, >, <$
     - algebraische Operatoren: $+, -, \cdot, \div, \text{mod}, \sum, \prod$
     - logische Operatoren: $\vee, \wedge, \neg$

     Ferner existieren zwei Funktionen zur Erzeugung von Pseudozufallszahlen, durch welche stochastische Umgebungen mit ELMS ermöglicht werden. Ausdrücke können in allen Sprachkonstrukten verwendet werden, in denen dies sinnvoll ist.
   - *Vorbedingungen (preconditions)*: Vorbedingungen für Aktionen, Reaktionen und Wahrnehmungen bestehen aus einer Sequenz logischer Operationen unter Verwendung der unterstützten logischen Operatoren.

- *Befehle (commands)*: Die in ELMS verfügbaren Befehle sind Zuweisung, Allokation eines Elements auf dem Grid, zufällige Allokation eines Elements auf dem Grid, Entfernung eines Elements vom Grid, Positionsänderung eines Elements auf dem Grid und Erzeugung sowie Entfernung von Instanzen.

2.2.1.2 *Zeitfortschritt und Ausführungsreihenfolge*

Der ELMS-Interpreter führt genau dann einen Zeitfortschritt durch, wenn er eine Aktionsnachricht von jedem Agenten erhalten hat. Wurde ein Zeitfortschritt ausgelöst, werden die vorliegenden Aktionen zur Bestimmung der Ausführungsreihenfolge zufällig in eine Warteschlange eingereiht. Die Warteschlange wird dann von vorne abgearbeitet: Sind die Vorbedingungen der jeweiligen Aktion erfüllt, wird die Aktion durchgeführt, anderenfalls wird eine Fehlermeldung an den Agenten versendet. Nach Abarbeitung aller Aktionen werden alle Reaktionen, deren Vorbedingungen erfüllt sind, durchgeführt.

Ferner unterstützt der ELMS-Interpreter einen asynchronen Simulationsmodus, dessen Semantik jedoch nicht genau bekannt ist. Es ist zu vermuten, dass im asynchronen Simulationsmodus jede Aktion eines Agenten einen Zeitfortschritt auslöst.

2.2.2 *MASSim*

MASSim ist eine Server-Software zur Realisierung von Umgebungen und wurde im Rahmen des *Multi-Agent Programming Contests*[4] entwickelt (vgl. [DDN06]). Der Multi-Agent Programming Contest ist ein Wettbewerb zur Evaluation verschiedener Konzepte zur Realisierung von Agenten und der Implementierungen dieser Konzepte. Im Zuge eines solchen Wettbewerbs stellen die Veranstalter eine Umgebung im Internet bereit, die über den MASSim-Server angeboten wird. Zu Beginn des Turniers verbinden sich die Agenten der teilnehmenden Teams über eine TCP-Verbindung mit dem MASSim-Server. Über diese Verbindung kommunizieren Agenten und Server durch den Versand von XML-Dokumenten (vgl. [BDHK]). Der Server startet im Laufe des Wettbewerbs eine Reihe von Matches, bei denen die Agenten von jeweils zwei Teams in einem Wettbewerbsszenario gegeneinander antreten. MASSim bewertet dabei

4 http://www.multiagentcontest.org/

die Leistung der Teams anhand szenariospezifischer Kriterien, um einen Gewinner des Matches zu ermitteln. Die im Moment nicht aktiven Agenten bleiben dennoch mit dem MASSim-Server verbunden und warten darauf, an einem Match teilnehmen zu können. In den vergangenen Wettbewerben konzentrierte sich der Multi-Agent Programming Contest dabei vor allem auf Szenarien, in denen Agenten in einer Gridwelt Ressourcen akquirieren müssen, wie z. B. durch Einsammeln von Gold oder das Treiben von Kühen in Gehege.

Der MASSim-Server implementiert vor allem technische Aspekte der Realisierung von Umgebungen, wie z. B. Mechanismen zur Authentifikation von Agenten, eine Schnittstelle zur Anbindung grafischer Komponenten, den Empfang und Versand von XML-Dokumenten über TCP-Verbindungen und eine Schnittstelle zur Einbindung von Szenarien. Nachrichtenaustausch zwischen Agenten wird hingegen nicht unterstützt. MASSim selbst realisiert keine Umgebungen, sondern stellt lediglich Schnittstellen und abstrakte Java-Klassen bereit, um in Java implementierte Gridwelt-Umgebungen über ein Netzwerk verfügbar zu machen.

Da die Semantik von Umgebungen nicht Teil von MASSim ist, sondern von der Java-Implementierung des jeweiligen Szenarios abhängt und es sich somit bei MASSim primär um ein technisches Werkzeug handelt, ist eine darüber hinausgehende konzeptionelle Beschreibung der von MASSim unterstützten Umgebungen nicht möglich.

3 ZIELSETZUNG

In diesem Kapitel sollen die konzeptionellen und implementatorischen Ziele des im Rahmen dieser Arbeit erstellten Frameworks vorgestellt werden.

3.1 KONZEPTIONELLE ZIELE

Zunächst wird festgelegt, welche der in Kapitel 2.1 vorgestellen Umgebungseigenschaften von GridWorldSim unterstützt werden. Danach werden weitere konzeptionelle Aspekte des Frameworks beschrieben.

3.1.1 *Unterstützte Eigenschaften*

Die Unterstützung bzgl. der Eigenschaften aus Kapitel 2.1 durch GridWorldSim stellt sich wie nachfolgend beschrieben dar.

3.1.1.1 *Zugänglichkeit*

Das Framework unterstützt die Einschränkung der Wahrnehmung durch sichtbehindernde Hindernisse und begrenzte Sichtweiten und somit *partiell wahrnehmbare Umgebungen*. Vollständig wahrnehmbare Umgebungen können dadurch erreicht werden, dass auf sichtbehindernde Hindernisse verzichtet und eine unbegrenzte Sichtweite für alle Agenten spezifiziert wird.

3.1.1.2 *Determinismus*

Es soll möglich sein, dass der Versuch eines Agenten, eine Aktion durchzuführen, gemäß einer Wahrscheinlichkeitsfunktion fehlschlägt, obwohl alle deterministischen Voraussetzungen für die Ausführung der Aktion erfüllt sind. Somit werden *stochastische Umgebungen* unterstützt.

3.1.1.3 *Sequenzialität*

Das im Rahmen dieser Arbeit entwickelte Framework unterstützt *sequentielle Umgebungen*, jedoch keine episodischen Umgebungen, da episodische Umgebungen vom Autor als seltener Spezialfall angesehen werden, während das Gros der Umgebungen sequentieller Natur ist.

3.1.1.4 *Dynamik*

Hinsichtlich der Dynamik von Umgebungen sind aus Agentensicht statische Umgebungen einfacher zu handhaben, da der Agent während des Überlegens der nächsten Aktion die Umgebung nicht beobachten muss und es insbesondere nicht möglich ist, dass eine Änderung des Umgebungszustands die aktuellen Überlegungen des Agenten irrelevant werden lässt.

Komplexe Umgebungen und insbesondere Umgebungen, in denen Agenten miteinander konkurrieren, sind jedoch als statische Umgebungen nur schwerlich zu realisieren. Eine statische Umgebung setzt voraus, dass die Umgebung keine Zustandsüberführung durchführt, bevor jeder Agent der Umgebung signalisiert hat, dass seine Überlegungen abgeschlossen sind und er nun eine Aktion durchführen möchte. In einer vollständig statischen Umgebung könnte ein Agent daher absichtlich den Fortschritt der Umgebung in einen neuen Zustand blockieren, indem er der Umgebung keine Rückmeldung gibt, z. B. weil er im aktuellen Zustand bereits eine für ihn optimale Situation erreicht hat und daher jeder Zustandsfortschritt nur eine Verschlechterung für ihn bedeuten könnte. Ebenso würde bei vollstatischen Umgebungen der Zustandsfortschritt blockiert, wenn ein Agent aus technischen Gründen der Umgebung keine Rückmeldungen mehr gibt.

GridWorldSim unterstützt sowohl *statische* als auch *dynamische Umgebungen*.

3.1.1.5 *Kontinuität*

Die reale Welt wird von Menschen zwar allgemein als kontinuierlich wahrgenommen, ob Zeit und Raum jedoch tatsächlich kontinuierliche Größen sind, stellt eine offene Frage der Physik dar. So postuliert die Theorie der Schleifenquantengravitation eine diskrete Natur von Zeit und Raum (vgl. [Sid05]).

Das im Rahmen dieser Arbeit entwickelte Framework unterstützt ausschließlich *volldiskrete Umgebungen*. Da eine Beantwortung der Frage, welche kontinuierlichen Größen in der realen Welt überhaupt existieren, nicht Gegenstand dieser Arbeit ist, wurde diese Entscheidung unter dem Aspekt getroffen, dass im Rahmen der Zustandsüberführung echte Gleichzeitigkeit gewünscht wird, wobei die Ausführungsreihenfolge in einem diskreten Zeitpunkt durch eine Ordnung bestimmt sein soll.

Sollte Kontinuität dennoch gewünscht sein, kann diese durch eine entsprechend feinstufige Modellierung zumindest approximiert werden.

3.1.1.6 *Proaktivität*

GridWorldSim unterstützt *proaktive Umgebungen*. Als proaktiver Umwelteinfluss wird dabei Nebel unterstützt. Nebel tritt mit einer spezifizierbaren Wahrscheinlichkeit auf. Wird diese Wahrscheinlichkeit mit 0 angegeben und liegt keine Spezifikation für anderes proaktives Verhalten vor, verhält sich eine von GridWorldSim realisierte Umgebung hingegen reaktiv.

3.1.2 *Weitere Funktionalität*

Neben Mechanismen zur Realisierung der gemäß Kapitel 3.1.1 unterstützten Eigenschaften soll GridWorldSim die im Folgenden genannte Funktionalität unterstützen.

3.1.2.1 *Aktionen*

Agenten können verschiedene Arten von Aktionen ausführen. Zu den grundlegenden unterstützten Aktionen zählt dabei die Bewegung des Agenten auf dem Grid in acht mögliche Richtungen: Norden, Nord-Osten, Osten, Süd-Osten, Süden, Süd-Westen, Westen und Nord-Westen. Ebenso ist es einem Agenten möglich, Objekte in sein Inventar aufzunehmen, aus seinem Inventar auf das Grid abzulegen, auf dem Grid zu verschieben oder aus seinem Inventar an einen anderen Agenten zu übergeben, sofern dieser bereit ist, das Objekt anzunehmen. Ferner kann ein Agent bei der Umgebung die Aktion anfordern, gar nichts zu tun. Darüberhinaus gibt es weitere Aktionen, z. B. zur Interaktion mit Gegenständen.

Jeder Agent darf zu jedem Zeitpunkt maximal eine Aktion ausführen. Dabei sind Aktionen i. d. R. an Vorbedingungen geknüpft: Zum Beispiel muss ein Agent über genügend Platz in seinem Inventar verfügen, um ein Objekt aufnehmen zu können. Auch im Falle einer Bewegungsaktion muss die Zielzelle der Bewegung genügend Platz für den Agenten bieten und darf zudem nicht von bestimmten Typen von Hindernissen blockiert werden. Da stochastische Umgebungen unterstützt werden, muss nicht jede Aktion, deren Vorbedingungen erfüllt sind, auch tatsächlich erfolgreich sein.

Die Gültigkeit von Vorbedingungen kann zudem davon abhängen, in welcher Reihenfolge zeitgleiche Aktionswünsche behandelt werden.

3.1.2.2 *Kommunikation*

Es ist eine Vielzahl von Umgebungen denkbar, in denen die Kommunikation zwischen Agenten nicht uneingeschränkt möglich ist.

Dies betrifft insbesondere Umgebungen, in denen Agenten eine begrenzte Sende- oder Empfangsreichweite besitzen oder Hindernisse Kommunikation stören können. Beide Fälle sind dem Menschen aus der realen Welt bekannt: Der Schall der menschlichen Stimme ist auf freiem Feld für einen Empfänger (abhängig von der Empfindlichkeit dessen Gehörs und des Schalldruckpegels der Nachricht) maximal einige Kilometer weit verständlich wahrnehmbar und aus einer schalldichten Zelle wird selbst der nur wenige Meter außerhalb der Zelle befindliche Empfänger nicht mehr erreicht.

Die Einschränkung von Kommunikation ist mit dem im Rahmen dieser Arbeit entwickelten Framework möglich. Dazu muss, ähnlich wie bei der Beschränkung der Wahrnehmung bei partiell wahrnehmbaren Umgebungen, ein Verfahren verwendet werden, um die tatsächlich per Kommunikation zu erreichenden Zellen zu bestimmen.

Es wird zwischen öffentlicher und privater Kommunikation unterschieden. Bei öffentlicher Kommunikation erhalten alle in Sende- bzw. Empfangsreichweite befindlichen Agenten die von einem Agenten versandte Nachricht, während bei privater Kommunikation nur die vom sendenden Agenten intendierten Empfänger die Nachricht erhalten („flüstern"). Private Kommunikation ist jedoch nur möglich, wenn sich Sender und intendierte Empfänger in der gleichen Gridzelle befinden; analog dazu kann auch ein Mensch keinem anderen Menschen etwas flüstern, wenn dieser mehrere Meter entfernt steht.

3.1.2.3 *Interaktion mit Objekten*

Ein Objekt in einer Umgebung ist ein Gegenstand (ein eigenständiges Objekt, welches nicht als Agent modelliert wird). Es soll Agenten in einer Umgebung möglich sein, mit Objekten auf bestimmte Art und Weise zu interagieren. Besitzt ein Objekt genügend Platz, um ein bestimmtes anderes Objekt aufzunehmen, so können bspw. Objekte einander hinzugefügt werden.

Darüber hinaus sollen *komplexe Objekte* unterstützt werden, mit denen Agenten gegenstandsspezifische Interaktionen durchführen können. GridWorldSim verfügt über komplexe Objekte vom Typ *Schließfach* und bietet die Unterstützung für eigene komplexe Objekttypen.

3.1.2.4 *Einschränkung von Bewegung, Wahrnehmung und Kommunikation*

Die Möglichkeiten von Agenten zur Bewegung, Wahrnehmung der Umgebung und Kommunikation sollen sich einschränken lassen. Bewegungen können dadurch eingeschränkt sein, dass

eine Gridzelle nicht über genügend Platz verfügt, um einen Agenten aufzunehmen. Die Wahrnehmung kann durch die maximale Sichtweite eines Agenten beschränkt werden und hinsichtlich Kommunikation hängt die Empfangbarkeit von Nachrichten von der Empfangs- und Sendeleistung der beteiligten Agenten ab. Darüber hinaus ist es möglich, Einschränkungen durch Hindernisse vorzunehmen.

Es gibt dabei fünf Arten von Hindernissen:

1. *Mauer*: verdeckt die Sicht auf dahinterliegende Gridzellen und verhindert das Betreten der zugemauerten Gridzelle
2. *Graben*: schränkt die Sicht nicht ein, verhindert aber das Betreten der Graben-Gridzelle
3. *Vorhang*: verdeckt die Sicht auf dahinterliegende Gridzellen, schränkt jedoch die Bewegungsmöglichkeiten von Agenten nicht ein
4. *Nebel*: wie Vorhang, wird aber proaktiv erzeugt und aufgehoben
5. *Interferenz*: blockiert Kommunikation

Mauern, Vorhänge und Nebel sind *sichtbehindernde Hindernisse*, Interferenz ist ein *kommunikationsblockierendes* Hindernis und Mauern und Gräben zählen zu den *bewegungsblockierenden Hindernissen.*

Mit Ausnahme von Interferenz können sich nicht mehrere Hindernisse in einer Zelle befinden, so kann z. B. eine Zelle nicht Mauer und Vorhang beinhalten, wohl aber Mauer und Interferenz.

3.1.2.5 *Zeitfortschritt und definierte Ausführungsreihenfolge*

Wie bereits in Kapitel 3.1.1.5 angesprochen, wird Zeit in Umgebungen durch das Durchlaufen diskreter Zeitpunkte modelliert. Ein diskreter Zeitpunkt entspricht dabei genau einem Umgebungszustand.

Es gibt verschiedene Möglichkeiten, wie die Überführung in einen neuen Umgebungszustand angestoßen werden kann:

1. *gesteuert durch Einzelagenten*: Jeder eintreffende Versuch eines Agenten, eine Aktion durchzuführen, erzeugt einen neuen Umgebungszustand.
2. *gesteuert durch alle Agenten*: Es wird genau dann in einen neuen Umgebungszustand überführt, wenn von jedem Agenten ein Versuch vorliegt, eine Aktion auszuführen. Dabei kann eine zusätzliche Zeitschranke sicherstellen, dass kein Agent den Fortschritt der Umgebung blockieren kann.

3. *taktgesteuert*: Die Überführung findet jeweils nach einer gewissen Spanne realweltlicher Zeit statt, wobei dennoch jeder Agent pro diskretem Zeitpunkt nur einen Versuch vornehmen kann, eine Aktion auszuführen.

Bei den Möglichkeiten 2 und 3 kann es vorkommen, dass mehrere Agenten zueinander konfligierende Aktionen anfordern, z. B. in dem mehrere Agenten zum selben Zeitpunkt versuchen, dasselbe Objekt in ihr Inventar aufzunehmen. Die Reihenfolge, in der Agenten bei Aktionen zum Zuge kommen, soll sich dabei nicht aus einer Eigenart der Implementierung ergeben, sondern konzeptionell eindeutig bestimmt und vom Benutzer spezifizierbar sein.

3.1.3 *Umgebungszustand und Zustandsüberführung*

Die Semantik, mit der das im Rahmen dieser Arbeit entwickelte Umgebungs-Framework Umgebungen realisiert, soll formal beschrieben werden. Dazu bedarf es geeigneter Formalismen zur Beschreibung von Umgebungszuständen und Zustandsüberführungen. Der initiale Umgebungszustand ergibt sich dabei aus der Spezifikation der Umgebung. Weitere Umgebungszustände sind das Resultat von Zustandsüberführungen, deren Ergebnis vom vorhergehenden Umgebungszustand, den anliegenden Versuchen von Agenten, eine Aktion auszuführen, und proaktivem Umgebungsverhalten abhängt.

3.2 IMPLEMENTIERUNG

Die Implementierung erfolgt unabhängig von einem bestimmten MAS-Framework in der Programmiersprache Java[1] und realisiert die in den vorherigen Abschnitten diskutierten Funktionen. Das Umgebungs-Framework stellt dabei den Agenten eine Umgebung in Form einer Serversoftware über ein TCP/IP-basiertes Netz bereit. Die Agenten können sich daher auf unterschiedlichen Rechnern befinden.

Darüber hinaus wird ein benutzergesteuerter Testagent in Form einer grafischen Client-Software implementiert, mit dessen Hilfe das Umgebungs-Framework getestet werden kann. Weiterhin existiert ein Beobachtungswerkzeug, mit dessen Hilfe ein Beobachter Statusinformationen der Agenten und den Zustand der Umgebung observieren kann. Ebenso steht zur Verwendung von GridWorldSim mit eigenen Agenten eine Referenzimplementierung der Kommunikationsprotokolle in Java zur Verfügung.

1 Java SE 6

3.2.1 *Kommunikation*

Obwohl Nachrichtenvermittlung üblicherweiser in den Aufgabenbereich des MAS-Frameworks fällt, muss sämtliche Kommunikation über GridWorldSim stattfinden, da Kommunikation zwischen Agenten durch begrenzte Sende- bzw. Empfangsreichweite und kommunikationsblockierende Hindernisse eingeschränkt sein kann. Denn nur GridWorldSim kann bestimmen, welche Kommunikation im aktuellen Umgebungszustand erlaubt ist und welche nicht. Kann davon ausgegangen werden, dass die Entwickler der Agenten eines Multiagentensystems sich an die vorgegebene Regel halten, dass die Agenten sämtliche Kommunikation über GridWorldSim abwickeln müssen, so sollten keine weiteren Maßnahmen erforderlich sein, um sicherzustellen, dass Agenten nicht an der Umgebung vorbei kommunizieren. In einer Wettbewerbssituation müssen jedoch geeignete technische Maßnahmen getroffen werden, die verhindern, dass Agenten die i. d. R. vorhandene Kommunikationsinfrastruktur des MAS-Frameworks direkt nutzen oder sogar über das Netzwerk direkt miteinander kommunizieren.

Da keine Vorgaben über das Format der Kommunikation zwischen Agenten gemacht werden sollen, muss die Implementierung beliebige Nachrichteninhalte eines Agenten kapseln können. Ebenso muss die Implementierung nach Erhalt einer für andere Agenten bestimmten Nachricht überprüfen können, welche anderen Agenten die Nachricht überhaupt empfangen können.

Darüber hinaus äußern Agenten ihre Wünsche zur Durchführung von Aktionen durch den Versand von Nachrichten an GridWorldSim, ebenso erhalten sie Informationen über den für sie wahrnehmbaren Umgebungszustand in Form von Nachrichten. Der Erfolg oder Misserfolg eines Aktionswunsches wird Agenten von der Umgebung dabei nicht explizit mitgeteilt, sondern die Agenten müssen durch die Beobachtung der Umgebung selbst feststellen, ob ihr Aktionswunsch erfolgreich war oder nicht. Die jeweils aktuelle Wahrnehmung des Agenten ist das einzige, was die Umgebung je einem Agenten mitteilt, andere Formen der Rückmeldung existieren nicht.

Es gelten die folgenden Anforderungen:

- Ein Agent soll über die Umgebung Nachrichten an andere Agenten verschicken können (ggf. unter Angabe der beabsichtigten Empfänger im Falle von privaten Nachrichten).

- Die Umgebung soll Agenten Informationen über das für diese sichtbare Umfeld mitteilen können (z. B. sichtbare Objekte, Art und Eigenschaften dieser Objekte, Hindernisse).

- Agenten sollen ihren Wunsch, Aktionen auszuführen, mitteilen können (z. B. Aufnahme, Abgabe, Übergabe, Verschiebung von Objekten, Interaktion mit Objekten, eigene Bewegung).

- Ein Beobachtungswerkzeug soll es ermöglichen, dass mit dessen Hilfe ein Beobachter Informationen über den vollständigen Zustand der Umgebung und über die internen Zustände der Agenten erhalten kann. Ebenso soll der Beobachter Zustandsüberführungen manuell anstoßen können. Daraus folgt, dass Agenten in der Lage sein müssen, der Umgebung Informationen über ihre internen Zustände mitzuteilen, ohne dass dies jedoch Einfluss auf die Semantik der Umgebung hätte. Das Kommunikationsprotokoll muss es daher erlauben, dass Agenten ihre internen Zustände übermitteln können.

- Vor der Teilnahme an einer Umgebung muss sich ein Agent initial bei der Umgebung anmelden können (mit Agentenname und Passwort).

3.2.2 *Spezifikation*

Für das Spezifikationsformat, welches den GridWorldSim-Server und die von ihm bereitgestellte Umgebung konfiguriert, gelten folgende Anforderungen:

- Innerhalb des Funktionsumfangs des Frameworks sollen sich beliebige Umgebungen beschreiben lassen.

- Der Aufwand bei der Spezifikation soll im Verhältnis zur Komplexität dessen stehen, was modelliert werden soll. Zum Beispiel sollte es für die Spezifikation einer geraden Wand der Länge 100 ausreichen, Start- und Endkoordinaten der Wand anzugeben, statt 100 Wandobjekte einzeln spezifizieren zu müssen.

- Neben der Spezifikation der funktionellen Eigenschaften soll auch die Spezifikation von technischen Notwendigkeiten (z. B. Agentennamen und Kennwörter der Agenten, denen eine Registrierung bei der Umgebung erlaubt ist) unterstützt werden.

Das Spezifikationsformat ist eine XML-Sprache. Durch das dazugehörige XML-Schema lässt sich die Gültigkeit einer Spezifikation bereits zum Erstellungszeitpunkt mittels Validierung durch XML-Standardsoftware überprüfen.

4 UMGEBUNGSZUSTAND UND ZUSTANDSÜBERFÜHRUNG

Das in diesem Kapitel beschriebene Vorgehen zur Spezifikation von Umgebungszuständen, Einflüssen und Zustandsüberführungsregeln ist inspiriert vom *Influences-and-Reaction-Modell* von Ferber und Müller (vgl. [FM96]), ohne dieses Modell jedoch zu übernehmen.

4.1 EINLEITUNG

Im Folgenden wird zum einen das prototypische Beispiel dargestellt, welches verdeutlicht, welche Arten von Szenarien im Rahmen der grundlegenden Funktionalität von Umgebungen unterstützt werden sollen, zum anderen wird das allgemeine Vorgehen zu Realisierung dieser Semantik skizziert und auf konzeptionelle Besonderheiten eingegangen.

4.1.1 *Prototypisches Beispiel*

Sei Tweety ein Agent, der in einer Gridwelt lebt. Seine Gridwelt ist zweidimensional und in ihr gibt es u. a. Agenten, Objekte, Hindernisse und Gridzellen. Tweety kann sich in dieser Gridwelt fortbewegen, wobei ihm eine Bewegung in alle benachbarten Gridzellen möglich ist, einschließlich derer, die diagonal zu seiner Gridzelle liegen. Damit diagonale Bewegungen ihm keinen Geschwindigkeitsvorteil verschaffen[1], lässt ihn die Umgebung bei diagonalen Bewegungen manchmal stolpern, so dass er für diesen Zeitpunkt in seiner Gridzelle bleiben muss. Wenn Tweety zum Rand der Gridwelt läuft, dann kann er dort nicht weitergehen. Ebenso gibt es Hindernisse, die ihm ein Betreten der Zelle unmöglich machen. Dabei handelt es sich um Mauern und Gräben.

Alle Agenten und Objekte in der Gridwelt, einschließlich Tweety, benötigen eine bestimmte Menge an Platz und verfügen über eine bestimmte Menge an Platz, um andere Objekte aufnehmen zu können. Gridzellen benötigen selbst keinen Platz, da sie ohnehin niemand aufnehmen kann, aber der Platz, den sie selbst

1 Der euklidische Abstand vom Mittelpunkt einer Zelle zum Mittelpunkt einer diagonal benachbarten Zelle beträgt $\sqrt{2}$ und ist somit größer als die Distanz zum Mittelpunkt einer horizontal oder vertikal benachbarten Zelle, welche 1 beträgt.

bieten, kann begrenzt sein. Wenn Tweety eine Zelle betreten möchte, die nicht über genug Platz für ihn verfügt, dann ist ihm dies nicht möglich. Betritt Tweety eine Zelle, dann reduziert sich der Platz in dieser Zelle um den Platzbedarf von Tweety.

Tweety kann Objekte in sein Inventar aufnehmen, falls er über genug Platz dazu verfügt. In diesem Fall reduziert sich sein freier Platz um den Platzbedarf des Objekts und sein eigener Platzbedarf steigt um den Platzbedarf des Objekts. Auch Objekte können über freien Platz verfügen. Wenn ein Objekt genug freien Platz besitzt, um ein anderes Objekt aufzunehmen, kann Tweety diesem Objekt das andere Objekt hinzufügen, vorausgesetzt, es ist entweder bereits in seinem Inventar oder er hat selbst genug Platz, um es aufzunehmen. Der Platzbedarf des aufnehmenden Objekts steigt dabei um den Platzbedarf des hinzugefügten Objekts und der freie Platz reduziert sich um denselben Wert. Analog dazu kann Tweety auch Objekte aus seinem Inventar oder aus dem Stauraum anderer Objekte wieder in die Gridzelle legen.

Tweety kann Objekte auch verschieben, ohne sie aufzunehmen. In seiner Gridwelt ist das Verschieben eines Objekts proportional schwer zum Platzbedarf des Objekts. Deshalb kann Tweety nur Objekte verschieben, für welche die Kraft reicht, mit der er drücken oder ziehen kann.

Objekte können in der Gridwelt beliebige von Tweety wahrnehmbare Eigenschaften haben, von Platzbedarf und freiem Platz abgesehen, sind diese Eigenschaften für die Umgebung jedoch nicht relevant, mit einer Ausnahme: Es gibt in seiner Gridwelt Schließfach-Objekte, in die Tweety Objekte ablegen kann, um darauffolgend das Schließfach mit einem Passwort zu verschließen. Ein Objekt, das in einem verschlossenen Schließfach liegt, kann von keinem Agenten wahrgenommen oder aufgenommen werden, solange, bis ein Agent das Schließfach mit dem richtigen Passwort wieder öffnet.

Tweety lebt in der Gridwelt nicht allein, sondern zusammen mit anderen Agenten. Diesen kann er Objekte aus seinem Inventar überreichen, wobei dies nur geschieht, wenn diese das Objekt auch annehmen wollen.

4.1.2 *Zustand und Zustandsüberführung*

Da das im Rahmen dieser Arbeit konzipierte Framework Zeit als diskrete Größe modelliert (siehe Kapitel 3.1.1.5), kann jeder Zeitpunkt durch eine natürliche Zahl beschrieben werden. Sei dabei ein Zeitpunkt, in dem sich eine Umgebung befindet, mit t bezeichnet, sei $t = 0$ der initiale Zeitpunkt einer Umgebung und sei $t + 1$ der Nachfolgezeitpunkt eines Zeitpunkts t.

Zu jedem Zeitpunkt t befindet sich die Umgebung in einem *Zustand* S_t, der alle in der Umgebung enthaltenen Elemente samt ihrer Eigenschaften vollständig beschreibt. Die drei grundlegenden Elemente einer Umgebung sind dabei Agenten, Objekte, Gridzellen und Nachrichten.

Beim Zeitfortschritt von t nach $t + 1$ wird die Umgebung vom Zustand S_t in den Zustand S_{t+1} überführt. Dazu werden *Zustandsüberführungsregeln* auf S_t angewandt, welche von *Einflüssen* ausgelöst werden. Ein Einfluss im Zeitpunkt t ist ein Versuch, den aktuellen Zustand S_t im Zuge der Überführung nach $t + 1$ auf eine bestimmte Art zu ändern. Eine Zustandsüberführungsregel legt abhängig von einem Zustand und der Ausführungswahrscheinlichkeit p der Regel fest, wann dieser Versuch erfolgreich ist und welche Änderungen sich für den Nachfolgezustand ergeben.

Beispiel 1: Der Agent Tweety versucht sich im Zustand S_t Richtung Osten zu bewegen und hofft deshalb, sich im Zustand S_{t+1} eine Gridzelle weiter östlich zu befinden. Er erzeugt dazu einen Einfluss, der seinem Versuch nach Osten zu gehen, Ausdruck verleiht. Dieser Einfluss löst die Regel aus, welche die Bewegung von Agenten realisiert. Sind die Vorbedingungen dieser Regel, welche festlegt, unter welchen Umständen sich Agenten bewegen dürfen und welche Auswirkungen eine erfolgreiche Bewegung für den Nachfolgezustand besitzt, erfüllt, so befindet sich Tweety mit Wahrscheinlichkeit p im Nachfolgezustand eine Zelle weiter östlich.

Beispiel 2: In einem Zustand S_t herrscht in einer Gridzelle (x, y) eine nebelbegünstigende Wetterlage. Sei f eine Regel, welche beschreibt, unter welchen Bedingungen in einer Zelle Nebel auftreten kann und welche Auswirkungen eine Nebelerzeugung für den Nachfolgezustand besitzt. Diese Regel wird von einem Einfluss ausgelöst, welchen die Umgebung selbst erzeugt. Liegt der Einfluss vor und sind die Vorbedingungen der nebelerzeugenden Regel erfüllt, wird mit der gegebenen Ausführungswahrscheinlichkeit der Regel in der Zelle (x, y) im Nachfolgezustand Nebel erzeugt.

Auch wenn es sich in beiden Beispielen um Einflüsse handelt, haben diese fundamental unterschiedliche Ursachen. Deshalb soll begrifflich wie folgt differenziert werden:

- Von Agenten erzeugte Einflüsse werden als *Aktionsanforderungen* bezeichnet.
- Von der Umgebung selbst erzeugte Einflüsse heißen *interne Einflüsse* und dienen der Realisierung proaktiven Verhaltens.

In der Literatur werden die als Aktionsanforderungen bezeichneten Versuche von Agenten, den Umgebungszustand zu ändern, häufig als *Aktionen* bezeichnet. Sprachlich bezeichnet das Wort „Aktion" das Vornehmen einer Handlung. Bei den im Rahmen dieser Arbeit konzipierten Umgebungen liegt es jedoch nicht in der Gewalt eines Agenten, ob er eine Handlung ausführen kann oder nicht. Deshalb findet der Begriff der „Aktionsanforderung" Verwendung. Stattdessen wird im Rahmen dieser Arbeit der Begriff „Aktion" für erfolgreiche Aktionsanforderungen verwendet, also für solche Aktionsanforderungen, die eine Änderung des Umgebungszustands bewirken.

Agenten werden über den Erfolg oder Nicht-Erfolg von Aktionsanforderungen nicht gesondert informiert, sondern müssen aufgrund ihrer Wahrnehmung der Umgebung im Folgezustand selbst feststellen, ob ihre Aktionsanforderung erfolgreich war oder nicht.

Angenommen zu einem Zeitpunkt t liegen mehrere regelauslösende Einflüsse vor. Bezögen sich die Vorbedingungen aller Regeln auf S_t und die Nachbedingungen aller Regeln auf S_{t+1}, könnte dies dazu führen, dass die Umgebung sich unerwünscht verhält.

Beispiel 3: In der Zelle (x, y) befindet sich ein Schokoladenriegel. Die Agenten Tweety und Geeko befinden sich ebenfalls in Zelle (x, y) und möchten beide im Zeitpunkt t diesen Schokoladenriegel an sich nehmen, weshalb beide entsprechende Aktionsanforderungen erzeugen. Besage die Vorbedingung der von diesen Aktionsanforderungen ausgelösten Regel, dass ein Objekt von einem Agenten genau dann aufgenommen werden kann, wenn es für den Agenten nicht zu schwer ist und wenn sich der Agent in der gleichen Zelle wie das Objekt befindet. Besage ferner die Nachbedingung der Regel, dass das Objekt, welches aufzunehmen versucht wurde, die Gridzelle verlässt und in das Inventar des Agenten übernommen wird.

Angenommen die Vorbedingungen sind sowohl für Tweety als auch für Geeko im Zustand S_t erfüllt. Die Regel würde nun für jede der beiden Aktionsanforderungen genau einmal ausgeführt und da in S_t die Vorbedingungen für beide Agenten erfüllt sind, würden für beide Regelausführungen die Nachbedingungen aktiv und der Schokoriegel würde dem Inventar beider Agenten hinzugefügt. Aus einem Schokoriegel wären zwei geworden.

Zur Vermeidung von Problemen bei sich gegenseitig ausschließenden Aktionsanforderungen, findet die Überführung von S_t nach S_{t+1} nicht direkt, sondern (im Falle von mindestens zwei ausgelösten Regeln) über Zwischenzustände statt. Die zuerst

ausgeführte Regel erzeugt ausgehend von S_t einen Zwischenzustand. Alle weiteren Regeln wenden ihre Vorbedingungen auf den Zwischenzustand an, der von der zuletzt ausgeführten Regel erzeugt wurde und erzeugen einen neuen Zwischenzustand, mit Ausnahme der zuletzt ausgeführten Regel, welche nach S_{t+1} überführt. Die Ausführungsreihenfolge wird dabei durch eine totale Ordnung der Einflüsse und eine totale Ordnung der Regeln bestimmt.

Auf diese Weise wird unerwünschtes Verhalten im Falle sich gegenseitig ausschließender Einflüsse vermieden: Wenn die Regel für Tweetys Aktionsanforderung zuerst ausgeführt wird, dann wird Geekos Regel bei der Ausführung auf einen Zwischenzustand angewendet, für den die Vorbedingungen der Regel nicht mehr erfüllt sind.

Es gilt: Der Zustand S_t in dem sich eine Umgebung zu einem Simulationszeitpunkt t befindet, wird durch eine Menge prädikatenlogischer Fakten beschrieben. Auf einen solchen Zustand wirkt eine Menge von Einflüssen I_t ein, die jedoch nicht Teil des Zustands ist. Für S_t gilt die *closed world assumption* und es gibt in S_t keine negativen Literale. Wenn $\neg P$ für S_t gültig ist, gilt daher $P \notin S_t$ und nicht $\neg P \in S_t$.

Der Zustand S_0 ergibt sich aus der initialen Spezifikation der Umgebung. Ein Umgebungszustand S_t zum Zeitpunkt $t \in \mathbb{N}^{>0}$ ergibt sich aus S_{t-1}, I_{t-1}, den Zustandsüberführungsregeln R, einer totalen Ordnung π und einer totalen Ordnung ρ. Die totale Ordnung π gibt an, in welcher Reihenfolge die Einflüsse zu berücksichtigen sind und die totale Ordnung ρ gibt an, in welcher Reihenfolge die zu einem Einfluss passenden Regeln berücksichtigt werden.

Es gilt also: Umgebungszustände beschreiben den *Ist-Zustand* einer Umgebung, Einflüsse stellen *Versuche dar, diesen Ist-Zustand zu ändern*, Zustandsüberführungsregeln beschreiben die *Regeln*, die festlegen, wie und unter welchen Bedingungen ein Einfluss zu einer Zustandsänderung führt und zwei totale Ordnungen legen fest, in welcher Reihenfolge Einflüsse und Regeln angewandt werden.

4.1.3 *Kapazitäten und Verschiebekraft*

In der realen Welt kann z. B. ein Ort nur eine gewisse Menge an Personen und Gegenständen aufnehmen, eine Tasche verfügt nur über begrenzt viel Stauraum und ein Mensch kann nicht unendlich große oder unendlich schwere Gegenstände tragen. Beschränkungen dieser Art sind von Eigenschaften wie Volumen oder Gewicht abhängig. Der Stauraum einer Tasche ist bspw. nicht über eine Anzahl von Objekten definiert, sondern über das

Volumen. Gleiches gilt auch für Menschen, denen das Tragen von zehn kleinen Erbsen in der Regel keine Schwierigkeiten bereitet, das Tragen eines einzelnen sehr großen oder sehr schweren Gegenstands hingegen schon.

Verallgemeinert gilt für die genannten Beispiele, dass eine bestimmte Kapazität der aufnehmenden Entität nicht kleiner sein darf als ein bestimmter Kapazitätsbedarf der aufzunehmenden Entität. Für das im Rahmen dieser Arbeit erstellte Konzept für Umgebungen soll gelten, dass von physikalischen Größen wie Masse oder Volumen abstrahiert werden soll. Als Verallgemeinerung von Begriffen wie „Gewicht" oder „Volumen" findet der Begriff *Kapazitätsbedarf* Verwendung. Begriffe wie „Stauraum" oder „Traglast" werden zum Begriff *Kapazität* verallgemeinert. In vielen Fällen ist die Gesamtkapazität dabei weniger relevant als die aktuell zur Verfügung stehende Kapazität. Diese ergibt sich aus der Kapazität eines Agenten, Objekts oder einer Zelle abzüglich der Kapazitätsbedarfe der von einem Agenten, Objekt oder einer Zelle aufgenommenen Agenten und Objekte.

Beispiel 4: Ein Objekt hat einen Kapazitätsbedarf von 20. Ein Agent mit einer freien Kapazität von 30 möchte dieses Objekt aufnehmen. Dies gelingt ihm und nach Aufnahme des Objekts beträgt seine freie Kapazität 10.

Beispiel 5: Eine Gridzelle besitzt eine freie Kapazität von 50 und ein Agent mit einem Kapazitätsbedarf von 60 möchte diese Zelle betreten. Das Betreten der Zelle ist ihm jedoch nicht möglich, da die Zelle nicht über genügend freie Kapazität verfügt.

Da Gridzellen nicht von Agenten oder Objekten aufgenommen werden können, haben diese nur eine Kapazität, jedoch keinen Kapazitätsbedarf.

Für handelnde Subjekte gilt in der realen Welt zudem, dass das Tragen von Gegenständen im Allgemeinen nicht genauso schwierig ist wie das Verschieben von Gegenständen. So können z. B. Menschen, die einen schweren Schrank schieben können, diesen nicht zwangsläufig tragen. Zur Abstrahierung dieses Umstandes verfügen Agenten neben Kapazität und Kapazitätsbedarf über eine *Verschiebekraft* genannte Größe. Diese gibt an, wie groß der Kapazitätsbedarf eines Objekts maximal sein darf, damit es vom Agenten in eine anderen Gridzelle verschoben werden kann.

4.1.4 *Übergabe von Gegenständen*

In der realen Welt kann ein Gegenstand von einem handelnden Subjekt in der Regel nur dann an ein anderes handelndes Subjekt übergeben werden, wenn der Empfänger bereit ist, den

Gegenstand auch anzunehmen. Dieses Prinzip muss auch für Umgebungen gelten, denn anderenfalls könnte ein Agent einem anderen Agenten dadurch schaden, dass er die freie Kapazität dieses Agenten durch Übergabe von Gegenständen, die dieser gar nicht annehmen möchte, erschöpft.

Die Bereitschaft einen Gegenstand zu akzeptieren, ist dabei Teil des internen Zustandes eines Agenten und der Umgebung nicht bekannt. Es muss daher eine Möglichkeit existieren, die Umgebung gewahr werden zu lassen, dass ein Agent einer bestimmten Übergabe zustimmt.

Dafür sind zwei Ansätze denkbar:

1. Ein Agent a_1, der einen Gegenstand übergeben möchte, stellt eine entsprechende Aktionsanforderung im Zustand S_t. Die Umgebung erzeugt daraufhin für den Agenten a_2, an den der Gegenstand übergeben werden soll, für den Zustand S_{t+1} eine spezielle Wahrnehmung, die ihm mitteilt, dass ihm a_1 einen Gegenstand übergeben möchte. Ähnlich wie in der realen Welt müssen sich die Agenten a_1 und a_2 dazu in der gleichen Zelle befinden. Nach Erhalt dieser speziellen Wahrnehmung kann a_2 im Zustand S_{t+1} der Umgebung antworten, ob er der Übergabe zustimmt. Ist dies der Fall und sind alle zur Übergabe notwendigen Vorbedingungen erfüllt, wird der Gegenstand im Zuge der Zustandsüberführung von S_{t+1} nach S_{t+2} von a_1 zu a_2 übergeben.

2. Ein Agent a_2 erklärt gegenüber der Umgebung, dass er bereit ist, ein bestimmtes Objekt o von einem Agenten a_1 zu empfangen, ohne dass a_1 eine entsprechende Aktionsanforderung zur Übergabe gestellt hat. Eine solche Erklärung wird Teil des Umgebungszustands. Soll eine Übergabe im Zeitpunkt t stattfinden, kann diese Erklärung in einem beliebigen Zeitpunkt k mit $k < t$ erfolgen. Stellt ein Agent die Aktionsanforderung zur Übergabe eines Gegenstandes, wird überprüft, ob vom Empfänger eine entsprechende Erklärung vorliegt. Ist dies der Fall, findet die Übergabe statt, anderenfalls nicht. Damit ein Agent überhaupt Kenntnis davon erlangt, dass ein anderer Agent ihm einen Gegenstand übergeben möchte, muss er bei dieser Lösung i. A. den beabsichtigten Empfänger vorher selbst mittels einer Nachricht über seine Übergabeabsicht informieren.

Die erste Möglichkeit besitzt den Nachteil, dass sich die beiden Agenten zum Zeitpunkt der Übergabeanfrage in der gleichen Zelle befinden müssen. Wenn sich a_1 und a_2 fortlaufend auf dem Grid bewegen, ist dies nicht ohne weiteres zu erreichen, da ein

Agent, der sich im Zustand S_t entscheiden muss, in welche Richtung er sich bewegen möchte, nicht weiß, in welche Richtung der andere Agent sich zu bewegen beabsichtigt. Selbst wenn sich beide Agenten im Zeitpunkt t in der gleichen Zelle befinden und a_1 eine Aktionsanforderung zur Übergabe gestellt hat, kann der Fall eintreten, dass a_2 im gleichen Zustand eine Aktionsanforderung gestellt hat, sich aus der Zelle fortzubewegen. Wird nun die Aktionsanforderung von a_2 vor der Aktionsanforderung von a_1 verarbeitet, würde die Aktionsanforderung von a_1 fehlschlagen und a_1 müsste erneut versuchen, im gleichen Zeitpunkt in die gleiche Zelle wie a_2 zu gelangen. Für den Agenten a_1 wäre es daher in vielen Fällen sinnvoller, a_2 per Nachricht über seine Übergabeabsicht zu informieren, damit er diesem nicht zeitintensiv folgen muss. Wird a_2 ohnehin per Nachricht von a_1 über dessen Übergabeabsicht in Kenntnis gesetzt, so ist die zweite Möglichkeit flexibler, da a_2 seine Zustimmung zur Übergabe auch zu Zeitpunkten erteilen kann, in denen er sich nicht in der gleichen Zelle wie a_1 befindet.

Das im Rahmen dieser Arbeit entwickelte Konzept realisiert daher die zweite Möglichkeit.

4.1.5 *Diagonalkorrektur*

Das Grid einer simulierten Welt dient üblicherweise als Abstraktion einer Fläche aus der realen Welt. Weist man jeder Gridzelle eine Höhe und eine Breite der Länge 1 zu, so handelt es sich bei der Entfernung zwischen dem Mittelpunkt einer Gridzelle und den Mittelpunkten der zu dieser Gridzelle benachbarten Gridzellen um den euklidischen Abstand. Dieser beträgt 1 für die Richtungen Norden, Osten, Süden und Westen und $\sqrt{2}$ für die Richtungen Nord-Osten, Süd-Osten, Süd-Westen und Nord-Westen.

Ein Agent kann zu jedem Zeitpunkt eine Aktionsanforderung stellen. Würde eine Aktionsanforderung zur Bewegung Richtung Norden, Westen, Süden und Osten genauso behandelt, wie eine Aktionsanforderung zur Bewegung Richtung Nord-Osten, Süd-Osten, Süd-Westen oder Nord-Westen, so entstünde eine große Diskrepanz zwischem dem Grid und der vom Grid abstrahierten realen Welt, da der Agent abhängig von der Richtung eine größere Aktionsreichweite besitzen würde.

Diese Diskrepanz kann dadurch minimiert werden, dass Aktionsanforderungen in diagonaler Richtung nur mit einer Wahrscheinlichkeit von $\frac{1}{\sqrt{2}}$ erfolgreich sind, obwohl die Vorbedingungen der zugehörigen Regel erfüllt sind. Im Erwartungswert läge damit auch bei die Vorbedingungen der zugehörigen Regel erfüllenden Aktionsanforderungen zur diagonalen Bewegung eine

Schrittlänge von 1 vor. Dieses Vorgehen wird im Weiteren als *Diagonalkorrektur* bezeichnet.

4.2 aufbau von umgebungszuständen

Im Folgenden werden die zur Kodierung von Umgebungszuständen verfügbaren grundlegenden Prädikate erläutert.

- **Grid**(x, y) gibt an, dass sich die Koordinate (x, y) auf dem Grid befindet. Für ein Grid mit Breite w und Höhe h sind die Fakten Grid(x,y) mit $0 \leq x \leq w-1$ und $0 \leq y \leq h-1$ Bestandteil eines jeden Zustands. Da die Größe des Grids sich zu keinem Zeitpunkt ändert, bleiben auch diese Fakten zu jedem Zeitpunkt unverändert.

- **Agent**(a): a ist ein Agent.

- **Object**(o): o ist ein Objekt, wie z. B. eine verschiebbare Kiste.

- **Contains**(z, o): Der Agent bzw. das Objekt z trägt bzw. beinhaltet das Objekt o.

- **Loc**(x, y, z): Der Agent oder das Objekt z befindet sich in Gridzelle (x, y). Befindet sich ein Objekt o im Inventar eines sich in Zelle (x, y) befindlichen Agenten oder Objekts, dann gilt Loc(x, y, o) dennoch nicht, da sich o nicht unmittelbar in der Gridzelle befindet.

- **CapNeed**(z, c): Der Agent oder das Objekt z besitzt einen Kapazitätsbedarf von c. Dieser Wert schließt von z getragene bzw. in z enthaltene Objekte mit ein.

- **FreeCellCap**(x, y, c): Die Zelle (x, y) verfügt über eine freie Kapazität von c.

- **FreeCap**(z, c): Der Agent oder das Objekt z verfügt über eine freie Kapazität von c.

- **MoveForce**(a, f): Der Agent a verfügt über eine Verschiebekraft von f.

- **Wall**(x, y): Auf Gridzelle (x, y) befindet sich eine Mauer (siehe Kapitel 3.1.2.4).

- **Trench**(x, y): Auf Gridzelle (x, y) befindet sich ein Graben (siehe Kapitel 3.1.2.4).

- **Curtain**(x, y): Auf Gridzelle (x, y) befindet sich ein Vorhang (siehe Kapitel 3.1.2.4).

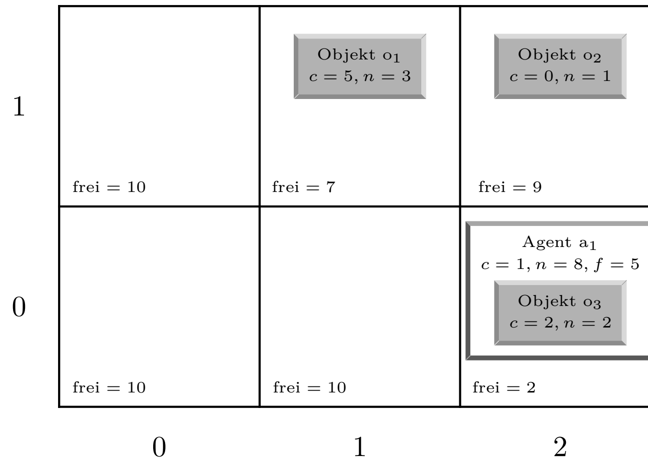

Abbildung 1: Visualisierung eines Umgebungszustands (c gibt die freie Kapazität an, n den aktuellen Kapazitätsbedarf und f die Verschiebekraft)

- **Interference**(x, y): Gridzelle (x, y) beinhaltet Interferenz (siehe Kapitel 3.1.2.4).
- **AcceptReceive**(a_1, a_2, o): Der Agent a_1 hat gegenüber der Umgebung die Bereitschaft erklärt, Objekt o von Agent a_2 zu akzeptieren.
- Darüber hinaus kann ein Umgebungszustand benutzerspezifizierte Fakten aufnehmen, die Eigenschaften eines Agenten oder Objekts beschreiben, z. B. Gold(o) um anzuzeigen, dass es sich bei o um Gold handelt. Solche Typangaben können Agenten bspw. dazu verwenden, um Objekten oder anderen Agenten eine Wertschätzung beizumessen. Man denke z. B. an ein Szenario, in welchem Agenten im Wettstreit miteinander möglichst viele Gegenstände mit möglichst hohem Wert einsammeln sollen. Die Bewertung von Objekten (wie „Gold ist wertvoll" oder „Blumen sind schön") obliegt jedoch ausschließlich der (subjektiven) Interpretation der Agenten und ist daher nicht Teil des Umgebungszustandes.

  Wenn die Umgebung den Agenten deren Beobachtungen mitteilt, so werden diese Typangaben den Agenten übergeben, für die Zustandsüberführung sind solche Fakten jedoch nicht relevant.

Beispiel 6: Abbildung 1 visualisiert folgenden Umgebungszustand: Es liegt ein Grid der Größe 3×2 vor und jede Gridzelle hat eine initiale Kapazität von 10. Es gibt drei Objekte: o_1 mit einer Kapazität von 5 und einem Kapazitätsbedarf von 3, o_2 mit einer Kapazität von 0 und einem Kapazitätsbedarf von 1 und o_3

$$
\begin{aligned}
&\{\mathrm{Grid}(x,y) \mid x \in \mathbb{N}^{\leq 2}, y \in \mathbb{N}^{\leq 1}\} \cup \\
&\{\mathrm{Agent}(a_1), \\
&\ \mathrm{Object}(o_1), \mathrm{Object}(o_2), \mathrm{Object}(o_3), \\
&\ \mathrm{Contains}(a_1, o_3), \\
&\ \mathrm{Loc}(2,0,a_1), \mathrm{Loc}(1,1,o_1), \mathrm{Loc}(2,1,o_2), \\
&\ \mathrm{CapNeed}(a_1, 8), \mathrm{CapNeed}(o_1, 3), \\
&\ \mathrm{CapNeed}(o_2, 1), \mathrm{CapNeed}(o_3, 2)\}, \\
&\ \mathrm{FreeCellCap}(0,0,10), \mathrm{FreeCellCap}(1,0,10), \\
&\ \mathrm{FreeCellCap}(2,0,2), \mathrm{FreeCellCap}(0,1,10), \\
&\ \mathrm{FreeCellCap}(1,1,7), \mathrm{FreeCellCap}(2,1,9), \\
&\ \mathrm{MoveForce}(a_1, 5), \\
&\ \mathrm{FreeCap}(a_1, 1), \mathrm{FreeCap}(o_1, 5), \\
&\ \mathrm{FreeCap}(o_2, 0), \mathrm{FreeCap}(o_3, 2)\}
\end{aligned}
$$

Spezifikation 1: Beispiel für einen Umgebungszustand

mit einer Kapazität von 2 und einem Kapazitätsbedarf von 2. Das Objekt o_1 befindet sich in Gridzelle $(1,1)$, o_2 in Zelle $(2,1)$ und o_3 im Inventar des Agenten a_1, der sich in Zelle $(2,0)$ befindet und über eine Kapazität von 3, einen Eigenkapazitätsbedarf von 6 und eine Verschiebekraft von 5 verfügt. Dabei ist zu beachten, dass beispielsweise für den Agenten a_1 der Wert von c gleich 1 und nicht etwa 3 ist, da der Agent bereits 2 Kapazitätseinheiten zum Tragen von Objekt o_3 benötigt. Ebenso ist n gleich 8 und nicht gleich 6, da der Kapazitätsbedarf enthaltener Objekte aufaddiert wird.

Der zugehörige Umgebungszustand ist in Spezifikation 1 angegeben. Die Objekte o_1 und o_2 sowie der Agent a_1 sind dabei konkrete Instanzen und keine Variablen.

4.3 Integritätsbedingungen

Für den Startzustand S_0 und alle Folgezustände müssen die folgenden Integritätsbedingungen gelten:

1. Es darf sich kein Agent oder Objekt in einer Zelle befinden, die eine Mauer oder einen Graben beinhaltet.

2. Jeder Agent befindet sich in genau einer Zelle.

3. Jedes Objekt befindet sich entweder in genauer einer Zelle oder in genau einem Inventar eines Agenten oder anderen Objekts.

4. Die freie Kapazität einer Zelle, eines Objekts oder eines Agenten wird nie negativ.

5. Der Kapazitätsbedarf eines Agenten oder Objekts wird nie negativ.

6. Bei der Aufnahme eines Agenten oder Objekts durch eine Zelle oder bei der Aufnahme eines Objekts durch einen Agenten oder ein anderes Objekt gilt, dass der Kapazitätsbedarf des Agenten, Objekts oder der Zelle um den Kapazitätsbedarf des aufgenommenen Elements steigt und die freie Kapazität des Agenten, Objekts oder der Zelle um diesen Kapazitätsbedarf verringert wird.

7. Verlässt ein Agent oder Objekt eine Zelle oder wird ein Objekt aus dem Inventar eines anderen Objekts oder Agenten entfernt, so steigt die freie Kapazität der Zelle, des Objekts oder des Agenten um den Kapazitätsbedarf des Elements, welches die Zelle, das Objekt oder den Agenten verlassen hat und der Kapazitätsbedarf verringert sich um den Kapazitätsbedarf dieses Elements.

8. Nur Interferenzen sind mit anderen Hindernissen kombinierbar, für alle anderen Hindernisse gilt, dass sich nicht zwei oder mehr von ihnen in einer Zelle befinden dürfen.

Ein Zustand S_t, in welchem diese Bedingungen erfüllt sind, heißt *konsistent*. Es gilt, dass bei der Ausführung von Umgebungen kein inkonsistenter Zustand auftreten darf.

4.4 AKTIONSANFORDERUNGEN

Aktionsanforderungen (vgl. Kapitel 4.1.2) werden genau wie die Fakten von Umgebungszuständen durch Prädikate repräsentiert. Ein Agent kann dabei nur Aktionsanforderungen stellen, in denen er selbst der Handelnde ist. Den Agenten stehen folgende grundlegende Typen von Aktionsanforderungen zur Verwendung der Basisfunktionalität zur Verfügung:

- **Move**(a, d): Der Agent a möchte sich in Richtung d bewegen, wobei $d \in \{\text{n, ne, e, se, s, sw, w, nw}\}$.

- **Take**(a, o): Der Agent a möchte das Objekt o aus seiner aktuellen Gridzelle in sein Inventar aufnehmen.

- **Release**(a, o): Der Agent a möchte das Objekt o aus seinem Inventar in seine aktuelle Gridzelle ablegen.

- **MoveObject**(a, o, d): Der Agent a möchte das Objekt o in Richtung d bewegen. Befindet sich das Objekt o in der

gleichen Gridzelle wie a, so kann der Agent das Objekt in eine benachbarte Gridzelle verschieben. Befindet sich das Objekt o in einer benachbarten Gridzelle, so kann der Agent das Objekt in seine Gridzelle ziehen.

- **HandOver**(a, a_1, o): Der Agent a möchte das Objekt o aus seinem Inventar an ein anderen Agenten a_1 übergeben, der sich entweder in der gleichen oder in einer benachbarten Gridzelle befindet.

- **DeclareAccept**(a, a_1, o): Der Agent a erklärt gegenüber der Umgebung die Bereitschaft, Objekt o von Agent a_1 empfangen zu wollen.

- **RetractAccept**(a, a_1, o): Der Agent a zieht seine zuvor erklärte Bereitschaft, Objekt o von Agent a_1 empfangen zu wollen, zurück.

- **Load**(a, o_1, o_2): Der Agent a möchte das Objekt o_1 in den Laderaum von Objekt o_2 legen. Dazu muss o_2 über genügend freie Kapazität verfügen, um den Kapazitätsbedarf von o_1 erfüllen zu können.

- **UnloadToGrid**(a, o_1, o_2): Der Agent a möchte Objekt o_1 aus dem Laderaum von Objekt o_2 entfernen, wobei Objekt o_1 in die aktuelle Gridzelle gelegt werden soll.

- **UnloadToInventory**(a, o_1, o_2): Der Agent a möchte das Objekt o_1 von Objekt o_2 entfernen und dabei in sein Inventar aufnehmen.

- **NoOp**(a): Der Agent a möchte zum aktuellen Zeitpunkt keine Aktion ausführen. Das Vorhandensein einer Aktionsanforderung kann eine Rolle bei der Entscheidung spielen, wann in den nächsten Zeitpunkt überführt wird. Dies ist der Grund für das Vorhandensein einer Aktionsanforderung, welche nie eine Aktion verursacht.

Wie in Kapitel 4.1.2 erläutert, stellen nicht nur Aktionsanforderungen (externe) Einflüsse dar, sondern es gibt darüber hinaus interne Einflüsse zur Realisierung proaktiven Verhaltens. Die Menge der Einflüsse I_t zum Zeitpunkt t ergibt sich nicht aus einer Überführung von I_{t-1}, sondern ausschließlich aus der Menge A_t der von Agenten zum Zeitpunkt t erhaltenen Aktionsanforderungen und der Menge B_t der gemäß Einflusserzeugungsregeln erzeugten internen Einflüsse. Es gilt also

$$I_t = A_t \cup B_t.$$

Die Möglichkeit, mit einem Einfluss mehrere Regeln auszulösen, ist hilfreich bei der Behandlung unterschiedlicher Ausgangssituationen und ermöglicht darüber hinaus die Verkettung von Regeln.

4.5 FUNKTIONEN

In den folgenden Abschnitten finden die im Folgenden spezifizierten Funktionen Verwendung. Die Aufgabe von Funktionen besteht zum einen darin, komplexe Berechnungen auszulagern, zum anderen dienen sie der kompakteren Darstellung. Folgende Funktionen sind gegeben:

- **xDir** erzeugt abhängig von einer Richtung (*n* wie Norden, *ne* wie Nord-Osten etc.) den zugehörigen Offset auf der x-Achse und ist wie folgt definiert:

$$\text{xDir} : \{\text{n, ne, e, se, s, sw, w, nw}\} \rightarrow \{0, 1, -1\}$$

$$\text{xDir}(d) = \begin{cases} -1 & \text{falls } d \in \{\text{sw, w, nw}\} \\ 0 & \text{falls } d \in \{\text{n, s}\} \\ 1 & \text{falls } d \in \{\text{ne, e, se}\} \end{cases}$$

- **yDir**(d) erzeugt analog dazu den zugehörigen Offset auf der y-Achse und ist wie folgt definiert:

$$\text{yDir} : \{\text{n, ne, e, se, s, sw, w, nw}\} \rightarrow \{0, 1, -1\}$$

$$\text{yDir}(d) = \begin{cases} -1 & \text{falls } d \in \{\text{se, s, sw}\} \\ 0 & \text{falls } d \in \{\text{e, w}\} \\ 1 & \text{falls } d \in \{\text{n, ne, nw}\} \end{cases}$$

- **dirProb** gibt abhängig von einer Richtung die Wahrscheinlichkeit an, mit der eine Aktion in dieser Richtung erfolgreich ist. Dies dient der Realisierung der Diagonalkorrektur (siehe Kapitel 4.1.5).

  Die Funktion ist daher wie folgt definiert:

$$\text{dirProb} : \{\text{n, ne, e, se, s, sw, w, nw}\} \rightarrow \{\tfrac{1}{\sqrt{2}}, 1\}$$

$$\text{dirProb}(d) = \begin{cases} \frac{1}{\sqrt{2}} & \text{falls } d \in \{\text{ne, se, sw, nw}\} \\ 1 & \text{falls } d \in \{\text{n, e, s, w}\} \end{cases}$$

- **euklid** ergibt (für im Falle des Grids natürliche Zahlen) den euklidischen Abstand im Zweidimensionalen für zwei Punkte (x_1, y_1) und (x_2, y_2) und ist wie folgt definiert:

$$\begin{aligned} &\text{euklid} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R} \\ &\text{euklid}(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \end{aligned}$$

4.6 vorgehen zur zustandsüberführung

Die Zustandsüberführung erfolgt gemäß Zustandsüberführungsregeln, wobei jede dieser Regeln von genau einem Typ von Einfluss ausgelöst wird.

Gegeben sei ein Umgebungszustand S_t zum Zeitpunkt t, eine Menge I_t von Einflüssen zum Zeitpunkt t, die jeweils mindestens eine Regel auslösen, eine Menge von Zustandsüberführungsregeln R, eine totale Ordnung π auf I_t, wobei $\pi_i(I_t)$ das i-te Element von I_t gemäß dieser Ordnung angibt und eine totale Ordnung ρ auf $R^* \subseteq R$, wobei $\rho_j(R^*)$ das j-te Element gemäß dieser Ordnung angibt. Notiere $R_k \subseteq R$ die Menge der von einem Einfluss $k \in I_t$ ausgelösten Regeln.

Die Umgebung durchläuft zur Überführung von S_t nach S_{t+1} interne Zwischenzustände der Form $S_{t,i,j}$. Sei $r \in R$, $l \in I_t$ und sei S ein Zustand. Für die Ausführungsfunktion **exec** gilt dann, dass $\text{exec}(r, l, S)$ den Zustand ergibt, der durch Anwendung der Regel r für den Einfluss l auf den Zustand S entsteht. Ein neuer Zustand S_{new} wird dann aus einem alten Zustand $S_{t,i,j}$ mittels

$$S_{\text{new}} = \text{exec}(\rho_j(R_{\pi_i(I_t)}), \pi_i(I_t), S_{t,i,j})$$

erzeugt. Für die Regelanwendung auf den Ausgangszustand S_t gilt

$$S_{\text{new}} = \text{exec}(\rho_j(R_{\pi_i(I_t)}), \pi_i(I_t), S_t) \text{ mit } i = 0, j = 0.$$

Für S_{new} gilt:

- Wurden noch nicht alle Regeln für den aktuell betrachteten Einfluss $\pi_i(I_t)$ verarbeitet, gilt also $j < |R_{\pi_i(I_t)}| - 1$, so ist $S_{\text{new}} = S_{t,i,j+1}$.

- Wurde die letzte Regel für den aktuell betrachteten Einfluss $\pi_i(I_t)$ verarbeitet, war aber $\pi_i(I_t)$ nicht der letzte unverarbeitete Einfluss, gilt also $i < |I_t| - 1$ und $j = |R_{\pi_i(I_t)}| - 1$, so ist $S_{\text{new}} = S_{t,i+1,0}$.

- Wurde die letzte Regel für den aktuell betrachteten Einfluss $\pi_i(I_t)$ verarbeitet und ist $\pi_i(I_t)$ der letzte Einfluss, gilt also $i = |I_t| - 1$ und $j = |R_{\pi_i(I_t)}| - 1$, so ist $S_{\text{new}} = S_{t+1}$.

Sofern $I_t \neq \emptyset$ ergibt sich damit ingesamt für die Zustandsüberführung von S_t nach S_{t+1}:

$$
\begin{aligned}
&S_t \xrightarrow{\text{exec}(\rho_0(R_{\pi_0(I_t)}),\pi_0(I_t),S_t)} S_{t,0,1} \dots \\
&\dots \xrightarrow{\text{exec}(\rho_{|R_{\pi_0(I_t)}|-1}(R_{\pi_0(I_t)}),\pi_0(I_t),S_{t,0,|R_{\pi_0(I_t)}|-1})} S_{t,1,0} \dots \\
&\dots \xrightarrow{\text{exec}(\rho_{|R_{\pi_{|I_t|-1}(I_t)}|-1}(R_{\pi_{|I_t|-1}(I_t)}),\pi_{|I_t|-1}(I_t),S_{t,|I_t|-1,|R_{\pi_{|I_t|-1}(I_t)}|-1})} S_{t+1}
\end{aligned}
$$

Bei der Zustandsüberführung von S_t nach S_{t+1} wird also zunächst die Menge aller Einflüsse total geordnet, damit für den Fall, dass mehrere Einflüsse konfligierende Aktionen nach sich ziehen würden, bestimmt ist, welcher Einfluss Vorrang besitzt. Dazu besitzen Agenten eindeutige Prioritäten, wobei die Aktionsanforderung eines höher priorisierten Agenten Vorrang vor der Aktionsanforderung eines Agenten mit niedrigerer Priorität besitzt. Eine höhere Priorität eines Agenten soll realweltlich der höheren Durchsetzungsfähigkeit eines Agenten im Konfliktfall entsprechen.

Eine mögliche Alternative bestünde darin, zuerst die Regeln gemäß ihrer totalen Ordnung zu verarbeiten und nur für den Fall, dass die gleiche Regel von verschiedenen Einflüssen ausgelöst wird, die totale Ordnung der Einflüsse zu berücksichtigen. Dies würde jedoch zu ungewolltem Verhalten führen.

Beispiel 7: Ein Agent a_1 ist groß und stark und dementsprechend durchsetzungsfähig im Konfliktfall und verfügt deshalb über eine hohe Priorität. Ein Agent a_2 ist hingegen klein und schwach und besitzt eine niedrige Priorität. Angenommen der Agent a_1 wolle ein Objekt o im gleichen Zeitpunkt verschieben, zu dem der Agent a_2 das Objekt in sein Inventar aufnehmen möchte. Sei ferner angenommen, bei der Regel für das Verschieben und bei der Regel zum Aufnehmen von Objekten in Inventare handele es sich um unterschiedliche Regeln. Abhängig davon, welche der beiden Regeln eine höhere Priorität besitzt, würde der Agent a_2 das Objekt aufnehmen können, obwohl er dem Agenten a_1 hinsichtlich seiner Durchsetzungsfähigkeit unterlegen ist.

Interne Einflüsse werden vorrangig zu Aktionsanforderungen behandelt. Dies entspricht der Vorstellung, dass von der Umgebung selbst erzeugte Einflüsse der Auslösung von Regeln dienen, die Naturgesetze (z. B. die Erzeugung von Nebel) repräsentieren und dass das Verhalten der Natur Vorrang vor den Aktionsanforderungen von Agenten besitzt. Für die Ordnung der internen Einflüsse untereinander kann für jeden Typ von Einfluss eine eindeutige Priorität angegeben werden, welche diese Einflüsse

$$
\begin{aligned}
I_t &= \{\text{MoveObject}(\text{tweety}, \text{box}, \text{n}), \\
&\quad \text{Move}(\text{geeko}, \text{w})\} \\
\pi_0(I_t) &= \text{Move}(\text{geeko}, \text{w}) \\
\pi_1(I_t) &= \text{MoveObject}(\text{tweety}, \text{box}, \text{n}) \\
R &= \{\text{Regel 1 mit Auslöser MoveObject}(a, o, d), \\
&\quad \text{Regel 2 mit Auslöser MoveObject}(a, o, d), \\
&\quad \text{Regel 3 mit Auslöser Move}(a, d)\} \\
\rho_0(\pi_0(I_t)) &= \text{Regel 3} \\
\rho_0(\pi_1(I_t)) &= \text{Regel 2} \\
\rho_1(\pi_1(I_t)) &= \text{Regel 1}
\end{aligned}
$$

Spezifikation 2: Beispiel-Ausgangssituation (ohne Angabe des Zustands)

total ordnet. Die Typen von internen Einflüssen, denen keine Priorität zugeordnet wird, haben eine eindeutige randomisierte Priorität, wobei ihre Priorität niedriger ist als die Priorität der Einflusstypen, für welche eine Priorität angegeben wird.

Die Zustandsüberführung beginnt dann mit dem Einfluss niedrigster Ordnung. Die von diesem Einfluss ausgelösten Zustandsüberführungsregeln werden gemäß ρ nacheinander ausgeführt. Bei der Ausführung einer Regel wird diese mit den Parametern des zugehörigen Einflusses instanziiert. Sind die Ausführungsbedingungen der Regel erfüllt, so ist der neue Zustand der um die Nachbedingungen der Regel geänderte alte Zustand, anderenfalls sind neuer und alter Zustand identisch. Wurden alle passenden Regeln für einen Zustand angewandt, fährt die Überführung mit dem nächsten Einfluss gemäß π gleichermaßen fort, solange bis alle Einflüsse und die durch sie ausgelösten Regeln verarbeitet wurden und damit der Zustand S_{t+1} erreicht ist.

Beispiel 8: Es soll von S_t nach S_{t+1} überführt werden. Gegeben sei das Beispiel in Spezifikation 2.

Zuerst wird überprüft, ob die Ausführungsbedingungen von Regel 3 für a = geeko und d = w im Zustand S_t erfüllt sind. Angenommen dies ist der Fall, so entsteht durch Anwendung der Nachbedingungen von Regel 3 auf den Zustand S_t der Zwischenzustand $S_{t,1,0}$, da keine weiteren passenden Regeln für den Einfluss vorliegen, es aber noch einen weiteren Einfluss gibt.

Nun wird überprüft, ob die Ausführungsbedingungen von Regel 2 für a = tweety, o = box und d = n im Zustand $S_{t,1,0}$ erfüllt sind. Angenommen dies ist nicht der Fall, so entsteht der neue Zustand $S_{t,1,1}$, der mit dem Zustand $S_{t,1,0}$ identisch ist.

Für den Zustand $S_{t,1,1}$ wird überprüft, ob die Ausführungsbedingungen von Regel 1 für $a =$ tweety, $o =$ box und $d =$ n erfüllt sind. Angenommen dies ist der Fall, dann entsteht durch Anwendung der Nachbedingungen von Regel 1 der Zustand S_{t+1}, da nun die letzte Regel für den letzten Einfluss verarbeitet wurde.

Für den Fall, dass es mehr als eine Belegung der freien Variablen gibt, welche die Vorbedingungen einer ausgelösten Regel im Zustand $S_{t,i,j}$ bzw. S_t erfüllt, so wird dabei die Regel für jede dieser erfüllenden Variablenbelegungen auf $S_{t,i,j}$ bzw. S_t (d. h. ohne Erzeugung weiterer Zwischenzustände) in zufälliger Reihenfolge angewandt.

Beispiel 9: Angenommen sei eine Regel r mit Auslöser Firefront(x), die der Realisierung einer Feuerwand für alle Zellen dient, welche sich auf der im Auslöser angegebenen x-Koordinate befinden. Habe diese Regel Grid(x, y) als einzige Vorbedingung. Die Variable x wird durch den auslösenden Einfluss Firefront(x) belegt. Soll eine Feuerwand für die x-Koordinate 5 erzeugt werden, so würde der Einfluss Firefront(5) vorliegen. Für die Vorbedingung der Regel würde dann Grid$(5, y)$ gelten. Diese Vorbedingung wäre für alle Grid(x, y) mit $x = 5$ im vorliegenden Zustand S erfüllt und die Regel würde für alle Grid$(5, y) \in S$ ausgeführt.

4.7 ZUSTANDSÜBERFÜHRUNGSREGELN

In diesem Abschnitt wird zunächst der Aufbau von Zustandsüberführungsregeln vorgestellt. Darauf folgt eine Angabe der Zustandsüberführungsregeln zur Festlegung der grundlegenden Funktionalität einer von GridWorldSim realisierten Umgebung.

4.7.1 *Aufbau*

Eine Zustandsüberführungsregel $r \in R$ besitzt die folgende Form:

Name der Regel	Auslöser
Pre:	Vorbedingungen
Prob:	Ausführungswahrscheinlichkeit
Post:	Nachbedingungen

Der Name der Regel dient ausschließlich informellen Zwecken und ist für die Zustandsüberführung nicht relevant.

Der Auslöser ist eine offene Formel, die einen Einflusstypen beschreibt. Ist der aktuell betrachtete Einfluss $\pi_i(I_t)$ eine mögliche Instanz dieser Formel, so gilt die zugehörige Regel als *ausgelöst*.

Zum Beispiel löst der Einfluss Move(tweety, w) eine Regel mit dem Auslöser $\text{Move}(a, d)$ aus.

Die Menge *Pre* beinhaltet die Formeln, von denen jede im aktuell betrachteten Zustand erfüllt sein muss, damit die Regel eine Zustandsänderung herbeiführen kann. Die im Auslöser vorkommenden Variablen werden dabei durch den auslösenden Einfluss instanziiert. Des Weiteren stehen der Kleiner-Gleich-Operator $\leq$ und der auf Gleichheit testende Operator $=$ mit Rückgabewert *true* oder *false*, der Additions-Operator $+$, der Subtraktions-Operator $-$ und der Multiplikations-Operator $\cdot$ in Infix-Notation zur Verfügung. Wurde z. B. die Regel mit Auslöser $\text{Move}(a, d)$ durch Move(Tweety, w) ausgelöst, so würde

$$Pre = \{\text{Loc}(x, y, a), \text{Grid}(x + \text{xDir}(d), y + \text{yDir}(d))\}$$

bedeuten, dass es sich bei der Position einen Schritt westlich von Tweetys aktuellem Ort um eine Gridzelle handeln muss. Sind die durch *Pre* spezifizierten Vorbedingungen nicht erfüllt, verändert die Regel den Zustand nicht.

Prob gibt die Wahrscheinlichkeit an, mit der eine Regel zu einer Zustandsänderung führt, wenn die Vorbedingungen erfüllt sind. $Prob < 1$ findet bei der Diagonalkorrektur (siehe Kapitel 4.5) Verwendung.

Die Menge der Nachbedingungen wird durch *Post* angegeben, wobei $+, -$ und $\cdot$ in Infix-Notation auch hier möglich sind. Positive Literale aus *Post* werden dem Zustand hinzugefügt, für negative Literale aus *Post* werden die zugehörigen positiven Literale aus dem Zustand entfernt (analog zur *add list* und *delete list* bei STRIPS, vgl. [FN71]). Die zu den freien Variablen der Formeln aus *Pre* gehörenden Instanzen instanziieren dabei die freien Variablen aus *Post*. Das Zeichen $\neg$ hat in *Pre* und *Post* unterschiedliche Bedeutung. In *Pre* bedeutet $\neg P$, dass im aktuellen Zustand S gelten muss: $P \notin S$. In *Post* bedeutet $\neg P$, dass P im Rahmen der Zustandsüberführung nicht in den neuen Zustand übernommen wird.

Die Anlehung an STRIPS bietet gegenüber einer logischen Formalisierung (wie z. B. in Form des Situationskalküls) den Vorteil, dass eine explizite Behandlung des Rahmenproblems nicht erforderlich ist, da der Operatordefinition und -ausführung implizit die STRIPS-Annahme zu Grunde liegt, welche besagt, dass alle Elemente einer STRIPS-Datenbasis, die nicht in der Nachbedingung eines Operators aufgeführt sind, bei der Anwendung des Operators unverändert bleiben (vgl. [BKI03]). Im Interesse einer – im Vergleich zum Situationskalkül – kompakteren und verständlicheren Darstellung wird daher einer an STRIPS angelehnten Notation der Vorzug gegeben. Für die STRIPS-Notation existiert eine Formulierung im Situationskalkül (vgl. [SS98] und [McC85]),

$$\begin{aligned} S_t \;=\; \{&\mathrm{Grid}(0,0), \mathrm{Grid}(0,1), \mathrm{Grid}(0,2), \mathrm{Grid}(1,0), \\ &\mathrm{Grid}(1,1), \mathrm{Grid}(1,2), \mathrm{Grid}(2,0), \mathrm{Grid}(2,1), \\ &\mathrm{Grid}(2,2), \mathrm{Agent}(\mathrm{tweety}), \mathrm{Loc}(1,1,\mathrm{Tweety})\} \end{aligned}$$

Spezifikation 3: Beispiel-Zustand

welche STRIPS-Regeln unter Zuhilfenahme von Reifikation in das Situationskalkül überführt. Inwieweit der im Rahmen dieser Arbeit verwendete (probabilistische) Formalismus auf ähnliche Weise überführt werden kann, liegt nicht im Rahmen der Aufgabenstellung und stellt eine mögliche Erweiterung dieser Arbeit dar.

Beispiel 10: Gegeben sei eine aus neun Gridzellen bestehende Umgebung zum Zeitpunkt t mit einer Länge und Breite von 3 sowie ein Agent Tweety, der sich in Zelle $(1,1)$ befindet. Der zugehörige Umgebungszustand ist in Spezifikation 3 gegeben. Tweety möchte immer noch nach Westen, von daher liegt der Einfluss Move(tweety, w) vor. Die Menge

$$Pre = \{\mathrm{Loc}(x,y,a), \mathrm{Grid}(x + \mathrm{xDir}(d), y + \mathrm{yDir}(d))\}$$

ergäbe dann instanziiert

$$Pre = \{\mathrm{Loc}(1,1,\mathrm{tweety}), \mathrm{Grid}(0,1)\}.$$

Da diese Vorbedingungen erfüllt sind, wird

$$Post = \{\neg\mathrm{Loc}(x,y,a), \mathrm{Loc}(x + \mathrm{xDir}(d), y + \mathrm{yDir}(d), a)\}$$

instanziiert zu

$$Post = \{\neg\mathrm{Loc}(1,1,\mathrm{tweety}), \mathrm{Loc}(0,1,\mathrm{tweety})\}.$$

Bei erfolgreicher Ausführung der Regel wird $\mathrm{Loc}(1,1,\mathrm{tweety})$ nicht in den neuen Zustand übernommen und $\mathrm{Loc}(0,1,\mathrm{tweety})$ dem neuen Zustand hinzugefügt. Tweety befände sich im Nachfolgezustand einen Schritt weiter westlich.

4.7.2 *Regeln*

Im Folgenden werden die grundlegenden Zustandsüberführungsregeln von GridWorldSim spezifiziert und erläutert.

4.7.2.1 *Bewegung von Agenten*

Die Regel **AgentMove** dient der Bewegung von Agenten auf dem Grid. Sie wird ausgelöst durch die Aktionsanforderung eines Agenten a, der sich in Richtung d bewegen möchte. Damit dies möglich ist, muss gelten, dass das Ziel der Bewegung eine existierende Gridzelle ist, die weder einen Graben, noch eine Mauer beinhaltet und über genügend freie Kapazität verfügt, um den Agenten aufzunehmen. Sind die Vorbedingungen erfüllt, kann der Agent mit einer Wahrscheinlichkeit von $\text{dirProb}(d)$ seine alte Zelle verlassen und die neue Zelle betreten. Die freien Kapazitäten der betroffenen Zellen werden in diesem Falle aktualisiert.

Die Regel lautet:

AgentMove		$\text{Move}(a,d)$
Pre:	$\{\text{Loc}(x,y,a),$	(1)
	$\text{Grid}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(2)
	$\neg\text{Wall}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(3)
	$\neg\text{Trench}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(4)
	$\text{FreeCellCap}(x,y,c_1),$	(5)
	$\text{FreeCellCap}(x+\text{xDir}(d), y+\text{yDir}(d), c_2),$	(6)
	$\text{CapNeed}(a,n),$	(7)
	$n \leq c_2\}$	(8)
Prob:	$\text{dirProb}(d)$	(9)
Post:	$\{\neg\text{Loc}(x,y,a),$	(10)
	$\neg\text{FreeCellCap}(x,y,c_1),$	(11)
	$\neg\text{FreeCellCap}(x+\text{xDir}(d), y+\text{yDir}(d), c_2),$	(12)
	$\text{Loc}(x+\text{xDir}(d), y+\text{yDir}(d), a),$	(13)
	$\text{FreeCellCap}(x,y,c_1+n),$	(14)
	$\text{FreeCellCap}(x+\text{xDir}(d), y+\text{yDir}(d), c_2-n)\}$	(15)

Da jeder Agent genau eine Position auf dem Grid besitzt, werden x und y in Zeile 1 eindeutig der Position von a zugeordnet. In Zeile 2 wird überprüft, ob es sich bei der in Richtung d gelegenen Zelle um eine Gridzelle handelt, also das Ziel der Bewegung nicht außerhalb des Grids liegt. Die Zeilen 3 und 4 legen fest, dass es sich bei der Zielzelle der Bewegung nicht um eine Zelle handeln darf, die eine Mauer oder einen Graben beinhaltet. Da jede Zelle (x,y) genau eine freie Kapazität besitzt, wird der Wert von c_1 in Zeile 5 der Kapazität der Zelle (x,y) eindeutig zugeordnet. Gleiches gilt für die Zuordnung der freien Kapaziät der Zielzelle zu c_2 in Zeile 6 und die Zuordnung des Kapazitätsbedarfs des Agenten zu n in Zeile 7. Zeile 8 beinhaltet die Bedingung,

dass der Kapazitätsbedarf des Agenten nicht größer sein darf als die freie Kapazität der Zielzelle. In Zeile 9 wird abhängig von der Richtung der Bewegung durch Verwendung der Funktion dirProb die Ausführungswahrscheinlichkeit der Regel festgelegt.

Ist die Aktionsanforderung erfolgreich, werden also die Nachbedingungen der Regel ab Zeile 10 angewendet, so werden der alte Ort, die alte freie Kapazität der Zelle, in der sich der Agent zum Zeitpunkt seiner Aktionsanforderung befindet und die alte freie Kapazität der Zelle, in die er sich bewegen möchte, nicht Teil des neuen Zustands (Zeile 10–12). In Zeile 13 wird dem neuen Zustand ein Faktum hinzugefügt, welches den neuen Ort des Agenten als die Zielzelle seiner Bewegung festlegt. Für die freie Kapazität der alten Zelle gilt, dass diese der vorherigen freien Kapazität entspricht zuzüglich des Kapazitätsbedarfs des Agenten a, der sich nun nicht mehr in dieser Zelle befindet. Die neue freie Kapazität wird in Zeile 14 berechnet und dem Zustand hinzugefügt. Auf die gleiche Weise wird in Zeile 15 die neue freie Kapazität der neuen Zelle, die sich aus der vorherigen freien Kapazität der Zelle abzüglich des Kapazitätsbedarfs des Agenten ergibt, Teil des neuen Zustands.

4.7.2.2 *Aufnahme und Ablage von Objekten*

Die Regel **TakeObjectFromGrid** dient der Aufnahme eines Objekts vom Grid in das Inventar eines Agenten. Sie wird ausgelöst durch die Aktionsanforderung eines Agenten a, der ein Objekt o von seiner Gridzelle aufnehmen möchte. Dies gelingt mit einer Wahrscheinlichkeit von 1, wenn sich das Objekt in der gleichen Gridzelle wie der Agent befindet und wenn der Kapazitätsbedarf des Objekts die freie Kapazität des Agenten nicht überschreitet. Ist die Aktion erfolgreich, so wird das Objekt vom Grid entfernt und dem Inventar hinzugefügt. Dabei steigt der Kapazitätsbedarf des Agenten und seine freie Kapazität sinkt um den Kapazitätsbedarf des Objekts.

Die Regel lautet:

TakeObjectFromGrid		Take(a, o)
Pre:	$\{\text{Object}(o),$	(1)
	$\text{Loc}(x, y, a),$	(2)
	$\text{Loc}(x, y, o),$	(3)
	$\text{FreeCap}(a, c),$	(4)
	$\text{CapNeed}(a, n_1),$	(5)
	$\text{CapNeed}(o, n_2),$	(6)
	$n_2 \leq c\}$	(7)
Prob:	1	(8)

Post: $\{\neg \text{FreeCap}(a, c),$ (9)
$\neg \text{CapNeed}(a, n_1),$ (10)
$\neg \text{Loc}(x, y, o),$ (11)
$\text{Contains}(a, o),$ (12)
$\text{FreeCap}(a, c - n_2),$ (13)
$\text{CapNeed}(a, n_1 + n_2)\}$ (14)

Zeile 1 stellt sicher, dass es sich bei o um ein Objekt handelt. Zeile 2 und 3 legen fest, dass die x- und y-Koordinate des Agenten und des Objekts identisch sind, sich beide also in der gleichen Zelle befinden. Zeile 4 ordnet c die freie Kapazität des Agenten zu, Zeile 5 ordnet n_1 den Kapazitätsbedarf des Agenten zu und Zeile 6 ordnet n_2 den Kapazitätsbedarf des Objekts zu. Gemäß Zeile 7 muss gelten, dass der Kapazitätsbedarf des Objekts nicht größer sein darf als die freie Kapazität des Agenten. Sind die Vorbedingungen erfüllt, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 8).

Ist die Aktionsanforderung erfolgreich, so werden die alte freie Kapazität des Agenten (Zeile 9), der alte Kapazitätsbedarf des Agenten (Zeile 10) und die alte Gridposition des Objekts (Zeile 11) nicht Teil des neuen Zustands. In Zeile 12 wird dem neuen Zustand hinzugefügt, dass der neue Ort des Objekts das Inventar des Agenten ist. Zeile 13 und 14 aktualisieren die freie Kapazität und den Kapazitätsbedarf von a, wobei die freie Kapazität um den Kapazitätsbedarf des Objekts verringert wird und sich der Kapazitätsbedarf des Agenten um den Kapazitätsbedarf des Objekts erhöht.

Die Regel **ReleaseToGrid** dient der Ablage eines Objekts aus dem Inventar eines Agenten in die aktuelle Gridzelle des Agenten. Sie wird ausgelöst durch die Aktionsanforderung eines Agenten a, der ein Objekt o aus seinem Inventar in seine aktuelle Gridzelle legen möchte. Die einzige nicht triviale Vorbedingung besteht darin, dass der Agent das Objekt in seinem Inventar besitzen muss. Die freie Kapazität der Zelle ist für die Regel nicht relevant, da der Kapazitätsbedarf von „Agent a mit Objekt o im Inventar“ genauso groß ist, wie die Summe der Kapazitätsbedarfe von „Agent a ohne Objekt o im Inventar“ und Objekt o. Ist die Aktion erfolgreich, so wird das Objekt aus dem Inventar entfernt und in die Gridzelle des Agenten gelegt. Dabei sinkt der Kapazitätsbedarf des Agenten und seine freie Kapazität steigt um den Kapazitätsbedarf des Objekts. Dass es sich bei o um ein Objekt handelt, muss im Gegensatz zur Regel TakeObjectFromGrid nicht überprüft werden, da sich o im Inventar des Agenten befinden muss und bereits bei der Aufnahme von Objekten sichergestellt wird, dass es sich tatsächlich um Objekte handelt.

Die Regel lautet:

ReleaseObjectToGrid		Release(a, o)
Pre:	$\{\text{Loc}(x, y, a),$	(1)
	$\text{Contains}(a, o),$	(2)
	$\text{FreeCap}(a, c),$	(3)
	$\text{CapNeed}(a, n_1),$	(4)
	$\text{CapNeed}(o, n_2)\}$	(5)
Prob:	1	(6)
Post:	$\{\neg\text{FreeCap}(a, c),$	(7)
	$\neg\text{CapNeed}(a, n_1),$	(8)
	$\neg\text{Contains}(a, o),$	(9)
	$\text{Loc}(x, y, o),$	(10)
	$\text{FreeCap}(a, c + n_2),$	(11)
	$\text{CapNeed}(a, n_1 - n_2)\}$	(12)

Zeile 1 ordnet x und y die x- und y-Koordinate der Positon von a zu. In Zeile 2 wird überprüft, ob der Agent a das abzulegende Objekt tatsächlich in seinem Inventar vorhält. Die Zuordnung der freien Kapazität des Agenten zu c, des Kapazitätsbedarfs des Agenten zu n_1 und des Kapazitätsbedarfs des Objekts zu n_2 findet in den Zeilen 3 bis 5 statt. Sind die Vorbedingungen erfüllt, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 6).

In den Nachbedingungen werden die alte freie Kapazität des Agenten, der alte Kapazitätsbedarf des Agenten und die Enthaltenseinsbeziehung zwischen Agent und Objekt nicht Teil des neuen Zustands (Zeile 7–9). Dass es sich bei dem neuen Ort des Objekts um die Position des Agenten handelt, wird in Zeile 10 dem Zustand hinzugefügt. Die Zeilen 11 und 12 aktualisieren die freie Kapazität und den Kapazitätsbedarf des Agenten.

4.7.2.3 *Bewegung von Objekten*

Für den Fall, dass ein Einfluss vom Typ MoveObject(a, o, d) vorliegt, wird zwischen zwei Fällen unterschieden:

1. Das Objekt o befindet sich in der gleichen Gridzelle wie der Agent a. In diesem Fall wird im Erfolgsfalle o aus der Zelle des Agenten in die benachbarte Zelle in Richtung d verschoben.

   Innerhalb der Grid-Abstraktion gibt es zwar keine unterscheidbaren Positionen innerhalb einer Zelle, in der realen Welt entspricht eine Gridzelle jedoch einer unterteilbaren Fläche. Das Verhalten der Umgebung im 1. Fall soll dem

realweltlichen Vorgang entsprechen, dass ein Agent das Objekt o zuerst in Richtung d bis zum Rand der Zelle und von dort über die Zellenmarkierung in die Nachbarzelle schiebt, wobei er selbst seine Zelle nicht verlässt.

2. Das Objekt o befindet sich in einer Nachbarzelle des Agenten a. Der Agent kann das Objekt dann in seine Zelle verschieben, wenn von der Zelle des Objekts aus gesehen die Zelle des Agenten in Richtung d liegt.

   Realweltlich entspricht dies dem Vorgang, dass das Objekt o sich an der Zellenmarkierung befindet, welche die Zellen von a und o trennt. Der Agent kann nun das Objekt in seine Zelle hinüberziehen, ohne dabei selbst seine Zelle zu verlassen.

Die beiden Fälle werden durch zwei verschiedene Regeln umgesetzt, wobei beide Regeln über sich gegenseitig ausschließende Vorbedingungen verfügen (ein Objekt kann sich nicht gleichzeitig in der gleichen Zelle wie ein Agent und in der Nachbarzelle dieses Agenten befinden).

Die Regel **MoveObjectOutOfCurrentCell** deckt den 1. Fall ab. Sie setzt voraus, dass sich das Objekt o in der gleichen Zelle wie Agent a befindet und dass die in Richtung d liegende Nachbarzelle existiert und keine Mauer und keinen Graben beinhaltet sowie über genug freie Kapazität verfügt. Darüber hinaus muss der Agent über eine Verschiebekraft verfügen, die für den Kapazitätsbedarf des Objekts ausreicht. Hinsichtlich der Ausführungswahrscheinlichkeit greift auch hier die Diagonalkorrektur bei Aktionsanforderungen in diagonaler Richtung. Findet die Aktion statt, so wird der Ort des Objekts aktualisiert und die Kapazitäten der betroffenen Gridzellen werden angepasst.

Die Regel lautet:

MoveObjectOutOfCurrentCell		MoveObject(a,o,d)
Pre:	$\{\text{Object}(o),$	(1)
	$\text{Loc}(x,y,a),$	(2)
	$\text{Loc}(x,y,o),$	(3)
	$\text{Grid}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(4)
	$\neg\text{Wall}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(5)
	$\neg\text{Trench}(x+\text{xDir}(d), y+\text{yDir}(d)),$	(6)
	$\text{FreeCellCap}(x,y,c_1),$	(7)
	$\text{FreeCellCap}(x+\text{xDir}(d), y+\text{yDir}(d), c_2),$	(8)
	$\text{CapNeed}(o,n),$	(9)
	$\text{MoveForce}(a,f),$	(10)
	$n \leq c_2,$	(11)

$$n \leq f\} \quad (12)$$

Prob: $\text{dirProb}(d)$ (13)

Post: $\{\neg\text{Loc}(x, y, o),$ (14)

$\neg\text{FreeCellCap}(x, y, c_1),$ (15)

$\neg\text{FreeCellCap}(x + \text{xDir}(d), y + \text{yDir}(d), c_2),$ (16)

$\text{Loc}(x + \text{xDir}(d), y + \text{yDir}(d), o),$ (17)

$\text{FreeCellCap}(x, y, c_1 + n),$ (18)

$\text{FreeCellCap}(x + \text{xDir}(d), y + \text{yDir}(d), c_2 - n)\}$ (19)

In Zeile 1 wird festgelegt, dass es sich bei o um ein Objekt handeln muss. Die Zeilen 2 und 3 stellen sicher, dass die Position von a und o identisch sind. Die Festlegung, dass das Ziel des Verschiebens eine Gridzelle sein muss, findet sich in Zeile 4. Die Zeilen 5 und 6 stellen sicher, dass die Zielzelle der Objektverschiebung keine Mauer und keinen Graben beinhaltet. Die Zuordnung der freien Kapazität der Zelle des Agenten zu c_1, der freien Kapazität der Zielzelle zu c_2, des Kapazitätsbedarfs von o zu n und der Verschiebekraft von a zu f wird in den Zeilen 7 bis 10 vorgenommen. Es gilt, dass die Zielzelle über genügend freie Kapazität zur Aufnahme von o verfügen muss (Zeile 11) und dass die Kapazität des Objekts nicht größer sein darf als die Verschiebekraft des Agenten (Zeile 12). Sind die Vorbedingungen erfüllt, werden die Nachbedingungen mit einer von der Richtung abhängenden Wahrscheinlichkeit aktiv (Zeile 13).

In den Nachbedingungen werden die alte Position des Objekts o (Zeile 14), die alte freie Kapazität der Zelle, die Ausgangspunkt der Verschiebung ist (Zeile 15) und die freie Kapaziät der Zelle, die Ziel der Verschiebung ist (Zeile 16), nicht Teil des neuen Zustands. Zeile 17 fügt den neuen Ort des Objekts o dem neuen Zustand hinzu und die Zeilen 18 und 19 erhöhen die freie Kapazität der alten Zelle und verringern die freie Kapazität der neuen Zelle um den Kapazitätsbedarf des Objekts.

Der 2. Fall, bei dem sich das Objekt o in einer zu a benachbarten Zelle befindet, wird von der Regel **MoveObjectFromNeighborCell** behandelt. Die Vorbedingungen der beiden Regeln unterscheiden sich darin, dass die Zelle von a nun in Richtung d zur Zelle von o benachbart sein muss. Zudem entfällt die Überprüfung, dass die Zielzelle der Verschiebung existieren muss und weder Mauer noch einen Graben beinhalten darf, da dies implizit der Fall ist, weil sich ansonsten in der Zielzelle kein Agent befinden könnte. Die Diagonalkorrektur findet auch hier Verwendung und die Nachbedingungen der Regel entsprechen den Nachbedingungen von MoveObjectOutOfCurrentCell (wenngleich die Variablen durch die unterschiedlichen Vorbedingungen in den Nachbedingungen eine andere Bedeutung haben).

Die Regel lautet:

MoveObjectFromNeighborCell — $\text{MoveObject}(a, o, d)$

Pre: $\{\text{Object}(o),$
$\text{Loc}(x, y, o),$
$\text{Loc}(x + \text{xDir}(d), y + \text{yDir}(d), a),$
$\text{FreeCellCap}(x, y, c_1),$
$\text{FreeCellCap}(x + \text{xDir}(d), y + \text{yDir}(d), c_2),$
$\text{CapNeed}(o, n),$
$\text{MoveForce}(a, f),$
$n \leq c_2,$
$n \leq f\}$

Prob: $\text{dirProb}(d)$

Post: $\{\neg\text{Loc}(x, y, o),$
$\neg\text{FreeCellCap}(x, y, c_1),$
$\neg\text{FreeCellCap}(x + \text{xDir}(d), y + \text{yDir}(d), c_2),$
$\text{Loc}(x + \text{xDir}(d), y + \text{yDir}(d), o),$
$\text{FreeCellCap}(x, y, c_1 + n),$
$\text{FreeCellCap}(x + \text{xDir}(d), y + \text{yDir}(d), c_2 - n)\}$

Aufgrund der Ähnlichkeit zur Regel MoveObjectOutOfCurrentCell wird auf eine Beschreibung der einzelnen Zeilen verzichtet.

4.7.2.4 *Übergabe von Objekten*

Die Regel **CreateAccept** behandelt DeclareAccept-Aktionsanforderungen, indem sie das zugehörige AcceptReceive-Faktum erzeugt. Eine von einem Agenten a abgegebene Erklärung gegenüber der Umgebung, von einem Agenten a_1 das Objekt o entgegenzunehmen, falls dieser in einem späteren Zeitpunkt versucht, das Objekt an a zu übergeben, ist immer erfolgreich, sofern a_1 ein Agent und o ein Objekt ist und nicht schon eine identische Erklärung vorliegt.

Die Regel lautet:

CreateAccept — $\text{DeclareAccept}(a, a_1, o)$

Pre:	$\{\text{Agent}(a_1),$	(1)
	$\text{Object}(o),$	(2)
	$\neg\text{AcceptReceive}(a, a_1, o)\}$	(3)
Prob:	1	(4)
Post:	$\{\text{AcceptReceive}(a, a_1, o)\}$	(5)

Zeile 1 legt fest, dass es sich bei a_1 um einen Agenten handeln muss und Zeile 2 bestimmt, dass o ein Objekt sein muss. Ist die angeforderte Bereitschaftserklärung bereits Teil des Zustandes, sollen die Nachbedingungen nicht aktiv werden (Zeile 3). Sind die Vorbedingungen erfüllt, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 4).

Die Nachbedingungen bestehen einzig daraus, dass dem neuen Zustand die Bereitschaftserklärung hinzugefügt wird (Zeile 5).

Wird eine abgegebene Erklärung von einem Agenten zurückgenommen, so wird diese RetractAccept-Aktionsanforderung von der Regel **RemoveAccept** behandelt. Die einzige Vorbedingung besteht darin, dass die zurückzurufende Erklärung existieren muss:

RemoveAccept	RetractAccept(a, a_1, o)	
Pre:	$\{\text{AcceptReceive}(a, a_1, o)\}$	(1)
Prob:	1	(2)
Post:	$\{\neg\text{AcceptReceive}(a, a_1, o)\}$	(3)

Zeile 1 setzt voraus, dass die zu entfernende Bereitschaftserklärung existieren muss. Ist dies der Fall, wird mit einer Wahrscheinlichkeit von 1 (Zeile 2) die Bereitschaftserklärung entfernt (Zeile 3).

Die Übergabe eines Objekts o von einem Agenten a zu einem Agenten a_1 wird durch die Regeln **TransferObjectCurrentCell** und **TransferObjectNeigborCell** realisiert. Diese Regeln werden von einem Einfluss vom Typ HandOver(a, a_1, o) ausgelöst. Erstere Regel realisiert die Übergabe eines Objekts an einen Agenten, der sich in der gleichen Zelle befindet, zweitere Regel die Übergabe an einen Agenten, der sich in einer benachbarten Zelle befindet.

Damit diese Aktionsanforderung Erfolg hat, muss der empfangende Agent a_1 zuvor seine Bereitschaft erklärt haben, das Objekt o vom Agenten a empfangen zu wollen (siehe Kapitel 4.1.4). Ferner muss sich o im Inventar von a befindet und die freie Kapazität von a_1 muss für den Kapazitätsbedarf von o ausreichen. Ebenso muss sich der Agent a_1 entweder in derselben Zelle oder in einer Nachbarzelle von a befinden. Für den Fall, dass das Objekt an einen Agenten a_1 in einer benachbarten Zelle übergeben wird, muss zusätzlich überprüft werden, ob die freie Kapazität dieser Nachbarzelle ausreicht. Bei einer Übergabe in eine Nachbarzelle in diagonaler Richtung greift zudem die Diagonalkorrektur. Ist die Aktionsanforderung erfolgt, geht o in den Besitz von a_1 über und die freien Kapazitäten beider Agenten sowie ggf. die freien Zellkapazitäten werden aktualisiert.

Eine Überprüfung, ob a_1 tatsächlich ein Agent ist und o tatsächlich ein Objekt, findet nicht statt, denn dies gilt implizit, da

ein AcceptReceive(a_1, a, o)-Faktum für a_1 und o vorliegen muss und bereits bei der Erzeugung dieses AcceptReceive-Faktums überprüft wurde, dass es sich bei a_1 um einen Agenten und bei o um ein Objekt handelt.

In Analogie zu TakeObjectFromGrid und ReleaseObjectToGrid muss ferner auch die Zellenkapazität für den Fall, dass an einen Agenten in der gleichen Zelle übergeben wird, nicht berücksichtigt werden, da das Objekt die Zelle nicht verlässt. Die Bereitschaft, das Objekt o vom Agenten a zu erhalten (z. B. wenn o von a_1 wieder abgegeben wurde und später von a erneut angeboten wird), bleibt auch im Erfolgsfalle bestehen und muss explizit mit RetractAccept widerrufen werden, so a_1 dies wünscht.

Die realweltliche Entsprechung bei der Übergabe in eine benachbarte Zelle besteht darin, dass sich beide Agenten (analog zu MoveObjectFromNeighborCell) an der zwischen ihren beiden Zellen liegenden Zellbegrenzung treffen und das Objekt übergeben, ohne dabei jeweils ihre Zelle zu verlassen. Würde GridWorldSim die Übergabe an einen Agenten in einer Nachbarzelle nicht unterstützen, so wäre es in der Umgebung nicht möglich, ein Objekt zu übergeben, wenn die Summe des Kapazitätsbedarfs von a (der den Kapazitätsbedarf von o beinhaltet) und a_1 größer ist als die Kapazität der Zelle, obwohl die Zelle ansonsten genug Platz haben könnte, einen der beiden Agenten samt dem Objekt aufzunehmen.

Beispiel 11: Angenommen in einer Gridwelt haben alle Zellen eine Kapazität von 10, alle Agenten eine Kapazität von 5 und alle Objekte eine Kapazität von 1. Ein Agent Tweety möchte einem Agenten Geeko ein Objekt o innerhalb der gleichen Zelle übergeben. Wenn sich außer Geeko sonst kein Objekt in Geekos Zelle befindet, hat diese Zelle eine freie Kapazität von 5. Tweety hat jedoch – weil sich Objekt o in seinem Inventar befindet – mindestens einen Kapazitätsbedarf von 6 und kann daher Geekos Zelle nicht betreten. Tweety kann Geeko das Objekt nur übergeben, wenn auch die zellübergreifende Übergabe von Objekten möglich ist.

Die Regel für die Übergabe innerhalb einer Zelle lautet:

TransferObjectCurrentCell	HandOver(a, a_1, o)	
Pre:	{Loc(x, y, a),	(1)
	Loc(x, y, a_1),	(2)
	AcceptReceive(a_1, a, o),	(3)
	Contains(a, o),	(4)
	FreeCap(a, c_1),	(5)
	FreeCap(a_1, c_2),	(6)

	$\text{CapNeed}(o, n),$	(7)
	$n \leq c_2\}$	(8)
Prob:	1	(9)
Post:	$\{\neg\text{Contains}(a, o),$	(10)
	$\neg\text{FreeCap}(a, c_1),$	(11)
	$\neg\text{FreeCap}(a_1, c_2),$	(12)
	$\text{Contains}(a_1, o),$	(13)
	$\text{FreeCap}(a, c_1 + n),$	(14)
	$\text{FreeCap}(a_1, c_2 - n)\}$	(15)

Die Zeilen 1 und 2 stellen sicher, dass sich a und a_1 in derselben Gridzelle befinden. In Zeile 3 wird festgelegt, dass eine Bereitschaftserklärung von a_1 vorliegt, das Objekt o vom Agenten a zu empfangen. Zeile 4 stellt sicher, dass der Agent a über das Objekt o in seinem Inventar verfügt. Die Zuordnung der freien Kapazität von a zu c_1, der freien Kapazität von a_1 zu c_2 und des Kapazitätsbedarfs von o zu n findet in den Zeilen 5 bis 7 statt. In Zeile 8 wird sichergestellt, dass a_1 über genügend freie Kapazität verfügt, um o aufnehmen zu können. Sind die Vorbedingungen erfüllt, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 9).

Für die Nachbedingungen gilt, dass die Enthaltenseinsbeziehung zwischen a und o sowie die alten freien Kapazitäten von a und a_1 nicht Teil des neuen Zustands werden (Zeile 10–12). Dem Nachfolgezustand wird eine Enthaltenseinsbeziehung zwischen a_1 und o hinzugefügt (Zeile 13). Gleiches gilt für die neuen freien Kapazitäten, wobei die freie Kapazität von a um den Kapazitätsbedarf von o erhöht (Zeile 14) und die freie Kapazität von a_1 um den Wert von o verringert wird (Zeile 15).

Die Übergabe eines Objekts an einen Agenten in einer benachbarten Zelle wird durch folgende Regel realisiert:

TransferObjectNeighborCell		$\text{HandOver}(a, a_1, o)$
Pre:	$\{\text{Loc}(x_1, y_1, a),$	(1)
	$\text{Loc}(x_2, y_2, a_1),$	(2)
	$\text{AcceptReceive}(a_1, a, o),$	(3)
	$\text{Contains}(a, o),$	(4)
	$\text{FreeCap}(a, c_1),$	(5)
	$\text{FreeCap}(a_1, c_2),$	(6)
	$\text{CapNeed}(o, n),$	(7)
	$\text{FreeCellCap}(x_1, y_1, d_1),$	(8)
	$\text{FreeCellCap}(x_2, y_2, d_2),$	(9)
	$n \leq c_2,$	(10)

$$n \leq d_2, \quad (11)$$
$$((\text{euklid}(x_1, y_1, x_2, y_2) = 1) \vee \quad (12)$$
$$(\text{euklid}(x_1, y_1, x_2, y_2) = \sqrt{2}))\} \quad (13)$$

Prob: $$\frac{1}{\text{euklid}(x_1,y_1,x_2,y_2)} \quad (14)$$

Post: $$\{\neg\text{Contains}(a, o), \quad (15)$$
$$\neg\text{FreeCap}(a, c_1), \quad (16)$$
$$\neg\text{FreeCap}(a, c_2), \quad (17)$$
$$\neg\text{FreeCellCap}(x_1, y_1, d_1), \quad (18)$$
$$\neg\text{FreeCellCap}(x_2, y_2, d_2), \quad (19)$$
$$\text{Contains}(a_1, o), \quad (20)$$
$$\text{FreeCap}(a, c_1 + n), \quad (21)$$
$$\text{FreeCap}(a_1, c_2 - n), \quad (22)$$
$$\text{FreeCellCap}(x_1, y_1, d_1 + n), \quad (24)$$
$$\text{FreeCellCap}(x_2, y_2, d_2 - n)\} \quad (24)$$

Zeile 1 ordnet x_1 und y_1 der (x, y)-Position von a zu, gleiches gilt für x_2 und y_2 für die Position von a_1 in Zeile 2. Zeile 3 stellt sicher, dass eine Bereitschaftserklärung von a_1 vorliegt, das Objekt o von a empfangen zu wollen. In Zeile 4 wird festgelegt, dass a das Objekt o in seinem Inventar beinhalten muss. In den folgenden Zeilen findet die Zuordnung der freien Kapazität von a zu c_1 (Zeile 5), der freien Kapazität von a_1 zu c_2 (Zeile 6), des Kapazitätsbedarfs von o zu n (Zeile 7), der freien Kapazität der Zelle (x_1, y_1) zu d_1 (Zeile 8) und der freien Kapazität der Zelle (x_2, y_2) zu d_2 statt (Zeile 9). Es muss nun gelten, dass die freie Kapazität von a_1 und die freie Kapazität der Zelle von a_1 nicht kleiner sein dürfen als der Kapazitätsbedarf von o (Zeile 10 und 11) und dass der euklidische Abstand der Zellen beider Agenten entweder 1 oder $\sqrt{2}$ beträgt, die Zellen also benachbart sind (Zeile 12 und 13). Ist dies der Fall, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1, falls die Zellen nicht diagonal benachbart sind, oder mit einer Wahrscheinlichkeit von $\frac{1}{\sqrt{2}}$, falls die Zellen diagonal benachbart sind, aktiv (Zeile 14).

In den Nachbedingungen werden die Enthaltenseinsbeziehung zwischen a und o (Zeile 15) und die alten freien Kapazitäten von a, a_1, der Zelle von a und der Zelle von a_1 nicht Teil des neuen Zustands (Zeile 16-19). Eine neue Enthaltenseinsbeziehung zwischen a_1 und o wird in Zeile 20 dem Zustand hinzugefügt. In den Zeilen 21 bis 24 werden die um den Kapazitätsbedarf von o erhöhten freien Kapazitäten von a und der Zelle von a dem Zustand hinzugefügt, ebenso wie die um den Kapazitätsbedarf von o verminderten freien Kapazitäten von a_1 und der Zelle von a_1.

4.7.2.5 *Be- und Entladen von Objekten*

Die Regeln zur Umsetzung der verbleibenden grundlegenden Einflüsse behandeln das Hinzufügen eines Objekts zu und das Entfernen eines Objekts aus dem Laderaum eines anderen Objekts.

Liegt ein Einfluss vom Typ Load(a, o_1, o_2) vor, werden die Regeln **LoadObjectFromGrid** und **LoadObjectFromInventory** ausgelöst. Beide Regeln behandeln unterschiedliche Ausgangssituationen: Erstere Regel ist für den Fall zuständig, dass sich o_1 auf dem Grid befindet. Dennoch darf die freie Kapazität von a nicht kleiner als der Kapazitätsbedarf von o_1 sein, damit der Agent das Objekt kurzfristig aufheben kann. Ebenso müssen sich beide Objekte und der Agent in der gleichen Gridzelle befinden und die freie Kapazität von o_2 darf nicht kleiner sein als der Kapazitätsbedarf von o_1.

Die Regel lautet wie folgt:

LoadObjectFromGrid		Load(a, o_1, o_2)
Pre:	$\{\text{Object}(o_1),$	(1)
	$\text{Object}(o_2),$	(2)
	$\text{Loc}(x, y, a),$	(3)
	$\text{Loc}(x, y, o_1),$	(4)
	$\text{Loc}(x, y, o_2),$	(5)
	$\text{CapNeed}(o_1, n),$	(6)
	$\text{FreeCap}(a, c_1),$	(7)
	$\text{FreeCap}(o_2, c_2),$	(8)
	$n \leq c_1,$	(9)
	$n \leq c_2\}$	(10)
Prob:	1	(11)
Post:	$\{\neg\text{Loc}(x, y, o_1),$	(12)
	$\neg\text{FreeCap}(o_2, c_2),$	(13)
	$\text{Contains}(o_2, o_1),$	(14)
	$\text{FreeCap}(o_2, c_2 - n)\}$	(15)

Die Zeilen 1 und 2 stellen sicher, dass es sich bei o_1 und o_2 um Objekte handelt. Beide Objekte und der Agent a müssen sich auf der gleichen Gridposition befinden (Zeile 3–5). Der Kapazitätsbedarf von o_1 wird n (Zeile 6), die freie Kapazität von a wird c_1 (Zeile 7) und die freie Kapazität von o_2 wird c_2 zugeordnet (Zeile 8). Zeile 9 legt fest, dass der Kapazitätsbedarf von o_1 nicht größer als die freie Kapazität von a sein darf und Zeile 10 legt fest, dass der Kapazitätsbedarf von o_1 nicht größer als die freie Kapazität von o_2 sein darf. Sind die Vorbedingungen erfüllt, werden die

Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 11).

Für den Nachfolgezustand gilt, dass der alte Ort von o_1 (Zeile 12) und die alte freie Kapazität von o_2 (Zeile 13) nicht mehr enthalten sind. Dafür besteht im Nachfolgezustand eine Enthaltenseinsbeziehung zwischen o_1 und o_2 (Zeile 14) und die freie Kapazität von o_2 ist um den Kapazitätsbedarf von o_1 vermindert (Zeile 15).

Für den Fall, dass das in den Laderaum von o_2 zu legende Objekt o_1 sich im Inventar des Agenten a befindet, muss die freie Kapazität von a im Gegensatz zur letzten Regel nicht mehr überprüft werden und o_1 muss sich nun im Inventar des Agenten und nicht mehr in der Gridzelle des Agenten befinden. Die Überprüfung auf Object(o_1) entfällt ebenfalls, da sie implizit gelten muss, denn wäre o_1 kein Objekt, hätte es nicht in das Inventar von a aufgenommen werden können. Ist die Aktion erfolgreich, gewinnt a im Gegensatz zur letzten Regel an freier Kapazität.

Damit ergibt sich die zweite Regel wie folgt:

LoadObjectFromInventory		Load(a, o_1, o_2)
Pre:	$\{$Object(o_2),	(1)
	Loc(x, y, a),	(2)
	Loc(x, y, o_2),	(3)
	Contains(a, o_1),	(4)
	CapNeed(o_1, n),	(5)
	FreeCap(a, c_1),	(6)
	FreeCap(o_2, c_2),	(7)
	$n \leq c_2\}$	(8)
Prob:	1	(9)
Post:	$\{\neg$Contains(a, o_1),	(10)
	$\neg$FreeCap(a, c_1),	(11)
	$\neg$FreeCap(o_2, c_2),	(12)
	Contains(o_2, o_1),	(13)
	FreeCap$(a, c_1 + n)$,	(14)
	FreeCap$(o_2, c_2 - n)\}$	(15)

Zeile 1 stellt sicher, dass es sich bei o_2 um ein Objekt handeln muss. Die Festlegung, dass sich a und o_2 auf der gleichen Gridposition befinden, findet in den Zeilen 2 und 3 statt. Zeile 4 gibt an, dass a das Objekt o_1 in seinem Inventar beinhalten muss. Der Kapazitätsbedarf von o_1 wird n (Zeile 5), die freie Kapazität von a wird c_1 (Zeile 6) und die freie Kapazität von o_2 wird c_2 zugeordnet (Zeile 7). Es muss gelten, dass der Kapazitätsbedarf von o_1 nicht größer ist als die freie Kapazität von o_2 (Zeile 8).

Gelten die Vorbedingungen, so werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 9).

Die Nachbedingungen legen fest, dass im Nachfolgezustand der Agent a das Objekt o_1 (Zeile 10) nicht mehr enthält. Ebenso werden die alten freien Kapazitäten von a und o_2 (Zeile 11 und 12) nicht Teil des Nachfolgezustands. Im neuen Zustand enthält o_2 das Objekt o_1 (Zeile 13), die freie Kapazität von a entspricht der um den Kapazitätsbedarf von o_1 erhöhten alten Kapazität (Zeile 14) und die freie Kapazität von o_2 entspricht der um den Kapazitätsbedarf von o_1 verminderten alten Kapazität (Zeile 15).

Für das Entfernen eines Objekts o_1 aus dem Laderaum eines Objekts o_2 stehen zwei verschiedene Aktionsanforderungen UnloadToGrid(a, o_1, o_2) und UnloadToInventory(a, o_1, o_2) zur Verfügung, je nachdem, ob der Agent o_2 in die Gridzelle legen oder in sein Inventar aufnehmen möchte. Die Regel **UnloadObjectToGrid** behandelt den ersten Fall. Beide Objekte müssen sich an der richtigen Position befinden (o_1 im Laderaum von o_2 und o_2 in der gleichen Zelle wie a) und a benötigt genügend freie Kapazität, um o_1 (zumindest kurzfristig) tragen zu können. Ist die Aktionsanforderung erfolgreich, gewinnt o_2 an freier Kapazität und der Ort von o_1 wird aktualisiert.

Damit lautet die Regel:

UnloadObjectToGrid		UnloadToGrid(a, o_1, o_2)
Pre:	$\{$Object(o_2),	(1)
	Loc(x, y, a),	(2)
	Loc(x, y, o_2),	(3)
	Contains(o_2, o_1),	(4)
	CapNeed(o_1, n),	(5)
	FreeCap(a, c_1),	(6)
	FreeCap(o_2, c_2),	(7)
	$n \leq c_1\}$	(8)
Prob:	1	(9)
Post:	$\{\neg$Contains(o_2, o_1),	(10)
	$\neg$FreeCap(o_2, c_2),	(11)
	Loc(x, y, o_1),	(12)
	FreeCap$(o_2, c_2 + n)\}$	(13)

Zeile 1 stellt sicher, dass es sich bei o_2 um ein Objekt handelt. In den Zeilen 2 und 3 wird festgelegt, dass sich a und o_2 auf der gleichen Gridposition befinden müssen. Ferner muss o_2 das Objekt o_1 beinhalten (Zeile 4). Der Kapazitätsbedarf von o_1 wird n (Zeile 5), die freie Kapazität von a wird c_1 (Zeile 6) und die freie Kapazität von o_2 wird c_2 zugeordnet (Zeile 7). Es muss

gelten, dass der Kapazitätsbedarf von o_1 nicht größer ist als die freie Kapazität von a (Zeile 8). Sind diese Vorbedingungen erfüllt, werden die Nachbedingungen mit einer Wahrscheinlichkeit von 1 aktiv (Zeile 9).

Die Nachbedingungen legen fest, dass im Nachfolgezustand das Objekt o_2 nicht mehr das Objekt o_1 enthält (Zeile 10) und dass die bisherige freie Kapazität von o_2 nicht mehr gültig ist (Zeile 11). Im Nachfolgezustand handelt es sich beim Ort von o_1 um die Gridzelle, in der sich auch a und o_2 befinden (Zeile 12) und die freie Kapazität von o_2 ist um den Kapazitätsbedarf von o_1 erhöht (Zeile 13).

Die Regel **UnloadObjectToInventory** zur Behandlung eines Einflusses vom Typ UnloadToInventory ist mit der letzten Regel hinsichtlich der Vorbedingungen identisch, in den Nachbedingungen wird jedoch das Inventar von a der neue Ort von o_1 und a verliert an Kapazität, da der Kapazitätsbedarf von o_1 nun dauerhaft von a zu erfüllen ist.

Die Regel lautet wie folgt:

UnloadObjectToInventory	UnloadToInventory(a, o_1, o_2)	
Pre:	$\{\text{Object}(o_2),$	(1)
	$\text{Loc}(x, y, a),$	(2)
	$\text{Loc}(x, y, o_2),$	(3)
	$\text{Contains}(o_2, o_1),$	(4)
	$\text{CapNeed}(o_1, n),$	(5)
	$\text{FreeCap}(a, c_1),$	(6)
	$\text{FreeCap}(o_2, c_2),$	(7)
	$n \leq c_1\}$	(8)
Prob:	1	(9)
Post:	$\{\neg\text{Contains}(o_2, o_1),$	(10)
	$\neg\text{FreeCap}(a, c_1),$	(11)
	$\neg\text{FreeCap}(o_2, c_2),$	(12)
	$\text{Contains}(a, o_1),$	(13)
	$\text{FreeCap}(a, c_1 - n),$	(14)
	$\text{FreeCap}(o_2, c_2 + n)\}$	(15)

Die Beschreibung der Zeilen 1 bis 10 ist identisch mit der Beschreibung dieser Zeilen für die Regel UnloadObjectToGrid.

Die Zeilen 11 und 12 legen fast, dass im Nachfolgezustand die vorherigen freien Kapazitäten von a und o_2 nicht mehr gelten. Zeile 13 fügt dem neuen Zustand das Faktum hinzu, dass a das Objekt o_1 in seinem Inventar besitzt. Die freie Kapazität von a vermindert sich deshalb um den Kapazitätsbedarf von o_1 (Zeile

14). Gemäß Zeile 15 ist die vorherige freie Kapazität von o_2 im neuen Zustand um den Kapazitätsbedarf von o_1 erhöht.

4.8 INTERNE EINFLÜSSE

In den vorhergehenden Abschnitten werden Einflüsse und Zustandsüberführungsregeln vorgestellt, bei denen ein Agent Einflüsse in Form von Aktionsanforderungen auf die Umgebung ausübt und diese darauf reagiert. In Kapitel 2.1.5.1 wird dieses Verhalten als *reaktiv* definiert, während unter *proaktivem Verhalten* die Veränderung eines Umgebungszustands verstanden wird, die nicht als direkte Konsequenz einer Aktionsanforderung erfolgt.

Zur Realisierung von proaktivem Verhalten bedarf es daher einer Möglichkeit, interne Einflüsse zu erzeugen, also Einflüsse, bei denen es sich nicht um Aktionsanforderungen von Agenten handelt.

Unterschieden werden soll dabei zwischen *deterministischen Einflüssen*, deren Erzeugung ausschließlich von S_t abhängt, *probabilistischen Einflüssen*, deren Erzeugung ausschließlich von S_t und einer Wahrscheinlichkeitsfunktion abhängt, und *indeterministisch-unprobabilistischen* Einflüssen, deren Erzeugung nicht ausschließlich von S_t und einer Wahrscheinlichkeitsfunktion abhängt. Regeln zur Erzeugung deterministischer Einflüsse und probabilistischer Einflüsse lassen sich ähnlich dem Aufbau von Zustandsüberführungsregeln als Regeln folgender Form angeben:

Name der Regel	
Pre:	Vorbedingungen in S_t
Prob:	Ausführungswahrscheinlichkeit
Post:	erzeugte Einflüsse

Bei indeterministischen-unprobabilistischen Regeln liegt eine schwierigere Ausgangssituation vor. Betrachtet man die reale Welt genau, so haben Vorgänge anscheinend immer eine Ursache und es ist anzunehmen, dass dort, wo die genaue Ursache nicht festgestellt oder nachvollzogen werden kann, das Kausalitätsprinzip dennoch gültig ist. So ist bspw. die Ursache eines konkreten Sturms für die meisten menschlichen Beobachter nicht unmittelbar erklärbar, dennoch steht es außer Zweifel, dass der Grund für den Sturm in lokal auftretenden hohen Luftdruckunterschieden liegt, welche gemäß physikalischen Gesetzen den Sturm verursachen. Selbst für den Willen des Menschen stellt sich die Frage, ob die von den meisten Menschen empfundene Freiheit des Willens nicht nur eine Illusion ist, da es zur Willensfreiheit eines Prinzips bedürfte, welches das menschliche Gehirn zu mehr befähigen würde, als nur seinen Dienst als komplexer

$$S_t = \{\text{Grid}(0,0), \text{Grid}(0,1), \text{Object}(o), \text{CapNeed}(o, 1000), \text{Loc}(0,0,c), \dots\}$$

CollapseWhenHeavy

Pre:	$\{\text{Object}(o), \text{Loc}(x,y,o), \text{CapNeed}(o,c), 500 < c\}$
Prob:	0,2
Post:	$\text{CollapseCell}(x,y)$

Spezifikation 4: Beispiel zur Erzeugung eines internen Einflusses

biochemischer Prozessor zu verrichten. Auch die Frage, ob indeterministische Messergebnisse im Bereich der Quantenmechanik tatsächlich durch objektiven Zufall zu erklären sind oder ob ihre Erklärung in verborgenen deterministischen Mechanismen liegt, ist offen.

Während Mechanismen für indeterministische Einflüsse in der realen Welt ein Problem darstellen, besitzen die von der Umgebung realisierten virtuellen Welten einen entscheidenden Vorteil: Die Umgebung wird auf einem System in der realen Welt ausgeführt, ohne dass die reale Welt dabei Teil der Umgebung wäre. Damit sind alle Einflüsse von außen aus Sicht der Umgebung indeterministisch.

Die Erzeugung der Menge der deterministischen und probabilistischen Einflüsse erfolgt durch Anwendung einer Methode **prodInfluence**(E, S_t), wobei E eine Menge von Einflusserzeugungsregeln der eben vorgestellten Form ist. Für jede Variablenbelegung, welche die Vorbedingungen einer Regel aus E in S_t erfüllt, wird dabei der in den Nachbedingungen spezifizierte Einfluss der von prodInfluence erzeugten Menge hinzugefügt.

Bei den indeterministisch-unprobabilistischen Einflüssen handelt es sich hingegen um Einflüsse, deren Ursprung in der realen Welt liegt, wie z. B. manuelle Eingriffe des Benutzers. Es ist daher unmöglich, Regeln für die Erzeugung indeterministisch-unprobabilistischer Einflüsse anzugeben.

Sei X_t die Menge der indeterministisch-unprobabilistischen Einflüsse zum Zeitpunkt t. Für die Menge B_t der internen Einflüsse gilt dann:

$$B_t = \text{prodInfluence}(E, S_t) \cup X_t$$

Beispiel 12: Gegeben sei das Beispiel in Spezifikation 4. Ein Einfluss vom Typ $\text{CollapseCell}(x,y)$ wird mit einer Wahrscheinlichkeit von 0,2 für die Zelle eines jeden Objekts mit einem Kapazitätsbedarf von mehr als 500 Kapazitätseinheiten erzeugt. Im Beispiel gilt dies in S_t für das Objekt o, welches sich in Zelle

(0,0) befindet, weshalb mit einer Wahrscheinlichkeit von 0,2 der Einfluss CollapseCell$(0, 0)$ ausgelöst wird.

4.9 TERMINIERUNG

Analog zur realen Welt terminiert eine von der Umgebung realisierte virtuelle Welt im Sinne der in diesem Kapitel vorgestellten Zustandsüberführung nicht. Es gibt also keine Menge von Terminierungsbedingungen, deren Erfüllung in S_t verhindert, dass die Umgebung in den Zustand S_{t+1} überführt wird.

4.10 ÜBERLEGUNGEN ZUR MODELLIERUNG

Bevor Überlegungen vorgenommen werden, wie praktikabel im Allgemeinen die Spezifizierung weiterer Objekte und Verhaltensweisen unter Verwendung des vorgestellen Formalismus ist, stellt sich die Frage, welches Verhalten der Umgebung überhaupt mit diesem Formalismus spezifiziert werden kann.

4.10.1 *Turingvollständigkeit*

Wie in Kapitel 4.5 erläutert, ist es möglich, innerhalb von Zustandsüberführungsregeln auf beliebige Funktionen zurückzugreifen. Sei angenommen, die einzige Regel $e \in E$ für prodInfluence(E, S_t) ist wie folgt gegeben:

StateInfluenceRule	
Pre:	$\top$
Prob:	1
Post:	StateInfluence(S_t)

Dieser in jedem Zeitpunkt t einmal erzeugte Einfluss StateInfluence(S_t) werde dabei von folgender Zustandsüberführungsregel behandelt:

StateTransitionRule	StateInfluence(S_t)
Pre:	$\top$
Prob:	1
Post:	turingFunction(S_t)

Die Funktion **turingFunction** erzeuge dabei eine Menge von positiven Literalen, die durch Anwendung der Zustandsüberführungsregel dem Folgezustand hinzugefügt werden und eine Menge von negierten Literalen, deren positive Formen aus dem Folgezustand entfernt werden.

Eine Turingmaschine M kann nun wie folgt simuliert werden: Sei B_t die Kodierung des Bandinhalts einer Turingmaschine zum Zeitpunkt t in Form logischer Fakten und sei Z_t die Kodierung des Zustands einer Turingmaschine zum Zeitpunkt t in Form logischer Fakten. Sei nun $B_t \subset S_t$ und $Z_t \subset S_t$. Ferner kodiere $\mathbf{BandPos}(x)$ die aktuelle Bandposition.

Die Funktion turingFunction(S_t) kann nun unter Berücksichtigung von Z_t und unter Selbstbeschränkung auf die Fakten aus B_t, die der aktuellen Bandposition zugeordnet sind, die Zustandsüberführung gemäß der Zustandsüberführungsfunktion einer Turingmaschine M vornehmen.

Im Gegensatz zu einer Umgebung hält eine Turingmaschine, wenn der aktuelle Zustand Teil einer Menge von akzeptierenden Zuständen ist. Die Menge der akzeptierenden Zustände kann als $F \subset S_t$ kodiert werden. Die Funktion turingFunction(S_t) werde nun so erweitert, dass sie Z_t daraufhin überprüft, ob der in Z_t kodierte Zustand einem in F kodierten Zustand entspricht. Ist dies der Fall, so wird ein Hold-Fakt erzeugt. Die Einflusserzeugungsregel StateInfluenceRule wird nun wie folgt modifiziert:

StateInfluenceRule*	
Pre:	$\{\neg\text{Hold}\}$
Prob:	1
Post:	StateInfluence(S_t)

Sobald Hold gilt, werden damit keine weiteren Einflüsse mehr erzeugt, welche die Zustandsüberführungsregel StateTransitionRule auslösen und damit zur Ausführung der Funktion turingFunction(S_t) führen würde.

Bei GridWorldSim wird Zeitfortschritt entweder durch das Vorliegen von Aktionsanforderungen oder durch einen realweltlichen Takt ausgelöst. Dies ist zumindest scheinbar ein Problem für die hier vorgestellte Simulation einer Turingmaschine, als dass entweder Zeitfortschritt überhaupt stattfindet oder auch dann stattfindet, wenn ein Zielzustand erreicht wurde. Für die Bewertung der Mächtigkeit des vorgestellten Formalismus ist dies jedoch zu vernachlässigen, ähnlich wie es für einen in einer Programmiersprache geschriebenen Turingmaschinensimulator nicht relevant ist, ob bei Erreichen eines akzeptierenden Zustands der zugehörige Betriebssystemprozess oder gar die verwendete Hardware terminiert. Es gilt daher:

Satz 1 *Der vorgestellte Formalismus zur Spezifikation des Verhaltens von Umgebungen ist turingvollständig.*

4.10.2 *Funktionen und Regeln*

Wie Kapitel 4.10.1 nahelegt, ist es möglich, die Semantik der Zustandsüberführungsregeln vollständig in Funktionen auszulagern. Dazu bedürfte es lediglich einer generischen Regel für jeden Einflusstypen, welche auch den Einfluss an eine Funktion übergibt. Alternativ könnten auch gleich die Zustandsüberführungsregeln aus dem Formalismus entfernt und stattdessen eine Zustandsüberführungsfunktion direkt auf dem aktuellen Zustand und der Menge der Einflüsse aufgerufen werden. Analog dazu bestünde auch die Möglichkeit, Einflusserzeugungsregeln durch Funktionen zu ersetzen. Dennoch sieht die Modellierung in Kapitel 4.7.2 und 4.8 davon ab, die Funktionalität von Zustandsüberführungs- und Einflusserzeugungsregeln vollständig in Funktionen auszulagern, sondern verwendet Regeln und Funktionen komplementär. Dies liegt darin begründet, dass die Aufgabe des verwendeten Formalismus nicht nur darin besteht, zur Spezifikation der gewünschten Semantik hinreichend mächtig zu sein, damit diese überhaupt mit Hilfe des Formalismus spezifiziert werden kann, sondern auch darin, die Semantik präzise und verständlich einem menschlichen Leser zu vermitteln.

Allgemeine mathematische Formalismen (wie sie zur Definition von Funktionen Verwendung finden) sind genauso wenig auf die Spezifikation des Verhaltens von Umgebungen spezialisiert wie allgemeine Programmiersprachen. Es ist deshalb sinnvoll, das grundlegende Verhalten von Umgebungen mit dem vorgestellten spezialisierten Formalismus zu beschreiben, da dieser für diesen Zweck kompakter und verständlicher ist und nur dort, wo allgemeinere Funktionalität erforderlich ist, auf Funktionen zurückzugreifen. Funktionen und Regeln sollten daher komplementär derart verwendet werden, dass nicht nur korrekte, sondern auch verständliche Spezifikationen entstehen.

4.11 ERWEITERTE FUNKTIONALITÄT

In den Kapiteln 4.2, 4.4 und 4.7.2 werden die zur Realisierung von grundlegendem reaktiven Verhalten notwendigen Prädikate zur Beschreibung von Umgebungszuständen, Typen von Aktionsanforderungen und Zustandsüberführungsregeln erläutert. Im Folgenden soll eine Erweiterung der dort gegebenen Spezifikationen stattfinden, um proaktives Verhalten und komplexe Gegenstände zu realisieren.

4.11.1 *Nebel*

Zunächst soll es möglich sein, dass in einer Zelle *Nebel* auftreten kann. Ist in einer Zelle Nebel vorhanden, kann ein Agent (genauso wie bei einem Vorhang) die Zelle zwar betreten, aber nicht sehen, was sich hinter der Zelle befindet. Nebel soll zufällig in jeder Zelle entstehen können, die weder Mauer noch Vorhang noch Graben beinhaltet. Ebenso soll existierender Nebel auch zufällig wieder verschwinden. Vom Benutzer sind dazu die Wahrscheinlichkeiten $p_{\text{createfog}}$ und $p_{\text{removefog}}$ anzugeben. Ist $p_{\text{createfog}} = 0$, so ist die Erzeugung von Nebel deaktiviert. Das Auftreten von Nebel stellt proaktives Verhalten dar.

Das Vorhandensein von Nebel in einer Zelle (x, y) wird im Umgebungszustand durch **Fog**(x, y) und der Einfluss zur Erzeugung von Nebel durch **MakeFog**(x, y) kodiert. Die zugehörige Einflusserzeugungsregel ist wie folgt gegeben:

FogInfluence	
Pre:	$\{\text{Grid}(x, y)\}$
Prob:	$p_{\text{createfog}}$
Post:	$\text{MakeFog}(x, y)$

Es wird also für jede Gridzelle mit einer Wahrscheinlichkeit von $p_{\text{createfog}}$ ein MakeFog-Einfluss erzeugt. Dieser Einfluss soll folgende Zustandsüberführungsregel auslösen:

CreateFog	$\text{MakeFog}(x, y)$
Pre:	$\{\neg\text{Fog}(x, y),$ $\neg\text{Curtain}(x, y),$ $\neg\text{Wall}(x, y),$ $\neg\text{Trench}(x, y)\}$
Prob:	1
Post:	$\{\text{Fog}(x, y)\}$

Um den Einfluss nur für gültige Gridpositionen zu erzeugen, benötigt die Einflusserzeugungsregel $\text{Grid}(x, y)$ als Vorbedingung. Es wäre in diesem Falle auch möglich, in den Vorbedingungen der Einflusserzeugungsregel auf Vorhang, Mauer und Graben zu testen und die Vorbedingungen der Zustandsüberführungsregel sogar gänzlich leer zu lassen. Da sich $\text{Grid}(x, y), \text{Curtain}(x, y)$, $\text{Wall}(x, y)$ und $\text{Trench}(x, y)$ nie ändern können, ist es in diesem Fall unerheblich, ob die Vorbedingungen Teil der Einflusserzeugungsregel oder der Zustandsüberführungsregel oder Teil von beiden Regeln sind.

Im allgemeinen Fall kann es jedoch entscheidend sein, welche Vorbedingungen Teil welcher Regel werden, da sich die Vorbedingungen der Einflusserzeugungsregel immer auf den Zustand S_t beziehen, während dies im Zuge der Zustandsüberführung nur für die erste Zustandsüberführungsregel gemäß ρ des ersten Einflusses gemäß π der Fall ist. Die Vorbedingungen aller anderen angewandten Zustandsüberführungsregeln beziehen sich jedoch auf einen Zwischenzustand $S_{t,i,j}$. Es ist daher sinnvoll in Fällen wie diesem, in denen die Vorbedingungen „Naturgesetze“ beschreiben, die Vorbedingungen Teil der Zustandsüberführungsregel werden zu lassen. Dies entspricht auch dem intuitiven Verständnis, welche Funktionen Einflüsse und Zustandsüberführungsregeln erfüllen.

Auch hinsichtlich der Frage, welche der beiden Regeln $p_{\text{createfog}}$ als Ausführungswahrscheinlichkeit erhalten soll, sind beide Varianten denkbar. Hier entspricht es jedoch der Intuition, dass wenn der Umwelteinfluss zur Erzeugung von Nebel vorliegt, die Umgebung immer Nebel erzeugt und dass es der Umwelteinfluss ist, der nicht ständig vorliegt. Deshalb erhält die Einflusserzeugungsregel $p_{\text{createfog}}$ als Ausführungswahrscheinlichkeit.

Der Einfluss zur Entfernung von Nebel wird durch **RemoveFog**(x, y) kodiert und von folgender Einflusserzeugungsregel erzeugt:

UnfogInfluence	
Pre:	$\{\text{Grid}(x, y)\}$
Prob:	$p_{\text{removefog}}$
Post:	$\text{RemoveFog}(x, y)$

Man mag versucht sein, dieser Regel $\text{Fog}(x, y)$ als Vorbedingung hinzuzufügen, da der erzeugte Einfluss ohnehin nur für Zellen mit Nebel eine Wirkung zeigt. In diesem Falle bestünde aber ein Unterschied im Verhalten der Umgebung: Wenn der Einfluss zur Erzeugung von Nebel gemäß π eine niedrigere Priorität besitzt als der Einfluss zur Entfernung von Nebel, dann kann in einem Zwischenzustand $S_{t,i,j}$ der Nebel entfernt, aber in einem späteren Zwischenzustand (oder S_{t+1}) neu erzeugt werden. Besitzt die Einflusserzeugungsregel hingegen $\text{Fog}(x, y)$ als Vorbedingung, so kann dies nicht geschehen. Für den Fall, dass innerhalb einer Zustandsüberführung von S_t nach S_{t+1} zuerst Nebel erzeugt, dieser aber vor Erreichen von S_{t+1} wieder entfernt wird, gilt das gleiche.

Intuitiv ist der Umwelteinfluss, der zur Auflösung des Nebels führt (z. B. starke Sonneneinstrahlung) unabhängig davon, ob sich in der Zelle Nebel befindet oder nicht. Schließlich scheint die Sonne nicht nur, wenn es vorher neblig war. Es wurde daher

darauf verzichtet, $\text{Fog}(x, y)$ zur Vorbedingung der Einflusserzeugungsregel zu machen.

Die zugehörige Zustandsüberführungsregel zur Auflösung von Nebel ist wie folgt gegeben:

UncreateFog	RemoveFog(x, y)
Pre:	$\{\text{Fog}(x, y)\}$
Prob:	1
Post:	$\{\neg\text{Fog}(x, y)\}$

Wird $\text{MakeFog}(x, y)$ in π höher priorisiert als $\text{RemoveFog}(x, y)$, so gilt für das Entstehen von Nebel in einer Zelle (x, y), in der sich in einem Zustand S_t kein Nebel befindet bzw. das Verschwinden von Nebel in einer Zelle (x, y), in der sich in einem Zustand S_t Nebel befindet:

$$\begin{aligned} P(\text{„Nebel entsteht“}) &= p_{\text{createfog}} \cdot (1 - p_{\text{removefog}}) \\ P(\text{„Nebel verschwindet“}) &= p_{\text{removefog}} \end{aligned}$$

Im umgekehrten Fall, dass $\text{RemoveFog}(x, y)$ höher priorisiert ist als $\text{MakeFog}(x, y)$, gilt:

$$\begin{aligned} P(\text{„Nebel entsteht“}) &= p_{\text{createfog}} \\ P(\text{„Nebel verschwindet“}) &= p_{\text{removefog}} \cdot (1 - p_{\text{createfog}}) \end{aligned}$$

Wie zu erkennen ist, bezieht sich „höher priorisiert“ im Sinne von π nur auf die Ausführungsreihenfolge und lässt ohne Kenntnis der Zustandsüberführungsregeln keine Rückschlüsse darauf zu, welcher Einfluss letzlich in S_{t+1} den gewünschten Effekt erzielt. Für Einflüsse vom Typ $\text{MakeFog}(x, y)$ und $\text{RemoveFog}(x, y)$ gilt daher: Der gemäß π zuletzt ausgeführte Einfluss besitzt Vorrang. Intuitiv lässt sich bspw. $\text{MakeFog}(x, y) < \text{RemoveFog}(x, y)$ so interpretieren, dass bei Vorliegen von Umwelteinflüssen, die normalerweise Nebel nach sich ziehen würden, kein Nebel erzeugt wird, wenn die Sonne scheint.

Wurden für $\text{MakeFog}(x, y)$ und $\text{RemoveFog}(x, y)$ die Prioritäten nicht spezifiziert, wird gemäß der Gleichverteilung randomisiert entschieden, welcher Einfluss Vorrang besitzt. In diesem Falle ergibt sich:

$$\begin{aligned} P(\text{„N. entsteht“}) &= p_{\text{createfog}} \cdot (1 - p_{\text{removefog}} \cdot 0{,}5) \\ P(\text{„N. verschwindet“}) &= p_{\text{removefog}} \cdot (1 - p_{\text{createfog}} \cdot 0{,}5) \end{aligned}$$

In diesem Fall wird also zufällig entschieden, welcher der beiden Umwelteinflüsse Vorrang besitzt.

4.11.2 *Schließfächer*

Ein *Schließfach* ist ein komplexes Objekt, in welches Agenten Gegenstände ablegen können, um sie später wieder herauszuholen. In einem Schließfach können mehrere Objekte abgelegt werden, solange die Summe der Kapazitätsbedarfe dieser Objekte die Schließfachkapazität nicht überschreiten. Jeder Agent kann für ein von ihm benutztes Schließfach ein Passwort festlegen, welches zum Herausnehmen von Objekten erforderlich ist, damit andere Agenten seine Objekte nicht entwenden können. Würde nur der Agent Objekte aus dem Schließfach entnehmen können, der sie selbst hineingelegt hat, könnte er anderen Agenten nicht durch Mitteilung des Passworts erlauben, ein Objekt aus seinem Schließfach herauszunehmen. Aus Gründen größerer Flexibilität wurde daher ein passwortbasierter Ansatz gewählt.

Im Umgebungszustand werden Eigenschaften von Schließfächern wie folgt kodiert:

- **SafeDepositBox**(b): b ist ein Schließfach.
- **Password**(b, p): Das Schließfach b ist durch das Passwort p geschützt.
- **Locked**(b): Das Schließfach b ist verschlossen. Ohne vorheriges Entsperren mit dem korrekten Passwort kann kein Agent Objekte aus dem Schließfach herausnehmen oder Objekte in das Schließfach hineinlegen.

Darüber hinaus finden die Prädikate aus Kapitel 4.2 Verwendung.

Beispiel 13: Es liegen zwei Schließfächer mit einer Kapazität von 10 vor. Ein Schließfach ist geöffnet und leer, das andere Schließfach ist mit einem Goldbarren mit Kapazitätsbedarf 3 belegt und durch das Passwort „secret“ geschützt. Die Schließfächer (ohne Inhalt) haben einen Kapazitätsbedarf von 2 Kapazitätseinheiten. Eine Menge von Fakten, welche die beiden Schließfächer beschreibt, ist in Spezifikation 5 gegeben. Hat ein Agent genügend freie Kapazität, um den Kapazitätsbedarf eines Schließfachs samt Inhalt aufzunehmen, so ist ihm das Aufnehmen des Schließfachs selbstverständlich möglich. Soll das Entwenden ganzer Schließfächer verhindert werden, so sind diese mit einem hohen Eigenkapazitätsbedarf zu spezifizieren.

Um mit Schließfächern zu interagieren, stehen den Agenten folgende neue Typen von Aktionsanforderungen zur Verfügung:

- **Lock**(a, b, p): Der Agent a möchte das Schließfach b schließen und dabei mit Passwort p schützen.

$$\begin{aligned}\{&\text{Object}(\text{fach1}), \text{Object}(\text{fach2}), \text{Object}(\text{gold}), \text{Gold}(\text{gold}),\\ &\text{SafeDepositBox}(\text{fach1}), \text{SafeDepositBox}(\text{fach2}),\\ &\text{Contains}(\text{fach2}, \text{gold}),\\ &\text{CapNeed}(\text{fach1}, 2), \text{CapNeed}(\text{fach2}, 5),\\ &\text{CapNeed}(\text{gold}, 3),\\ &\text{FreeCap}(\text{fach1}, 10), \text{FreeCap}(\text{fach2}, 7),\\ &\text{Locked}(\text{fach2}),\\ &\text{Password}(\text{fach2}, \text{„secret“})\}\end{aligned}$$

Spezifikation 5: Beispiel für zwei Schließfächer

- **Unlock**(a, b, p): Der Agent a möchte das Schließfach b unter Verwendung von Passwort p öffnen.

Für Lade- und Entlade-Aktionen sollen die Aktionsanforderungen Load(a, o_1, o_2), UnloadToGrid(a, o_1, o_2) und UnloadToInventory(a, o_1, o_2) aus Kapitel 4.4 Verwendung finden. Die zugehörigen Regeln LoadObjectFromGrid, LoadObjectFromInventory, UnloadObjectToGrid und UnloadObjectToInventory aus Kapitel 4.7.2 kennen jedoch bis auf die Kapazität keine Beschränkungen zur Ablage eines Objekts in und Aufnahme eines Objekts aus einem anderen Objekt. Dies ist nun aber erforderlich, da Objekte aus einem geschlossenen Schließfach nicht entwendet werden können. Daher werden die Vorbedingungen der Regeln LoadObjectFromGrid, LoadObjectFromInventory, UnloadObjectToGrid und UnloadObjectToInventory um $\neg$Locked(o_2) erweitert. Dies ist ausreichend, da für jedes Schließfach b auch Object(b) gilt und aufgrund der *closed world assumption* $\neg$Locked(o) für jedes Nichtschließfach-Objekt o gültig ist.

Nach dieser Erweiterung der Lade- und Entladeregeln aus Kapitel 4.7.2, müssen noch Regeln zum Öffnen und Schließen von Schließfächern spezifiziert werden. Die Regel **LockDepositBox** behandelt dabei Einflüsse vom Typ Lock(a, b, p) und verschließt ein Schließfach b dann mit einem Passwort p, wenn sich der Agent in der gleichen Zelle wie das Schließfach befindet und das Schließfach nicht bereits verschlossen ist.

Die Regel ist wie folgt gegeben:

LockDepositBox	Lock(a, b, p)
Pre:	$\{\text{SafeDepositBox}(b),$ $\text{Loc}(x, y, a),$ $\text{Loc}(x, y, b),$ $\neg\text{Locked}(b)\}$

Prob: 1
Post: {Locked(b),
Password(b, p)}

Für das Entsperren behandelt die Regel **UnlockDepositBox** Einflüsse vom Typ Unlock(a, b, p). Es gilt ebenso, dass sich Agent und Schließfach in der gleichen Zelle befinden müssen. Ist das Schließfach gesperrt und das richtige Passwort gegeben, dann wird das Schließfach geöffnet und das Passwort zurückgesetzt.

Die Regel lautet wie folgt:

UnlockDepositBox	Unlock(a, b, p)

Pre: {SafeDepositBox(b),
Loc(x, y, a),
Loc(x, y, b),
Locked(b),
Password(b, p)}
Prob: 1
Post: {¬Locked(b),
¬Password(b, p)}

4.11.3 *Funktionalitätsumfang und Erweiterungen*

Die Menge der für eine Umgebung denkbaren komplexen Gegenstände und Verhaltensweisen ist unüberschaubar groß. Selbst wenn es gelänge, an dieser Stelle auch nur ansatzweise alle wichtigen Objekten und Regeln der realen Welt zu spezifizieren, so wäre auch dies nicht ausreichend für einen Benutzer, welcher Fantasiewelten ohne realweltliche Entsprechung modellieren möchte. Es erscheint deshalb nicht sinnvoll, zu versuchen, im Rahmen dieser Arbeit eine möglichst große Anzahl an Objekten und Verhaltensweisen zu entwickeln, da selbst in diesem Falle die Menge der bereitgestellen Funktionalität verschwindend gering im Vergleich zur denkbaren Funktionalität wäre und der Benutzer deshalb mit großer Wahrscheinlichkeit für seine virtuellen Welten trotzdem eigene Objeke und Verhaltensweisen spezifizieren müsste.

Durch die Spezifikation eigener oder Modifikation bestehender Zustandsüberführungs- und Einflusserzeugungsregeln und Funktionen sowie die Einführung neuer Prädikate zur Verwendung für Umgebungszustände und Einflüsse lässt sich die vorgestellte Semantik von Umgebungen jedoch flexibel erweitern.

4.12 AUSLÖSUNG VON ZEITFORTSCHRITT

In Kapitel 3.1.2.5 werden die Möglichkeiten einen Zeitfortschritt auszulösen, also eine Überführung von S_t nach S_{t+1}, bereits kurz angesprochen. GridWorldSim bietet dabei drei verschiedene Optionen:

1. Das Vorhandensein der Aktionsanforderung eines einzigen Agenten löst einen Zeitfortschritt aus.

2. Ein Zeitfortschritt wird ausgelöst, wenn Aktionsanforderungen von allen in der Umgebung vorhandenen Agenten vorliegen (oder durch Überschreiten einer Zeitschranke, wenn von einem Agenten nach einer gewissen Zeitspanne keine Rückmeldung erhalten wird).

3. Der Zeitfortschritt erfolgt gemäß eines bestimmten Takts nach realweltlicher Zeit.

Möglichkeit 1 bietet den Vorteil, dass jeder Zustandsüberführung genau eine Aktionsanforderung zugeordnet ist; auf eine Ordnung der Aktionsanforderungen in π könnte daher verzichtet werden. Durch Einzelagenten gesteuerter Zeitfortschritt bietet jedoch eine Reihe von Nachteilen. So könnte z. B. ein Agent, der in der Lage ist, sehr viele Aktionsanforderungen in einer Spanne realweltlicher Zeit zu stellen, viel mehr Aktionen durchführen als ein Agent, der dies nicht kann. Dadurch hätte er erhebliche Vorteile. Ebenso kann ein Agent nicht abschätzen, wie viel Zeit ihm für Überlegungen zur Verfügung steht, um sich für die Aktionsanforderung zu entscheiden, die er im aktuellen Umgebungszustand durchführen möchte, da er i. A. nicht wissen kann, ob und wann ein anderer Agent die Überführung in einen neuen Umgebungszustand veranlasst. Wenn bspw. Agent a_1 fünf Sekunden benötigt, um sich für die im aktuellen Zustand zu stellende Aktionsanforderung zu entscheiden und bei einer zwischenzeitlichen Änderung des Umgebungszustands für den neuen Zustand erneut mit dem Überlegen beginnt, könnte ein feindlich gesonnener Agent a_2 den Agenten a_1 effektiv von der Teilnahme an der Umgebung ausschließen, indem er alle drei Sekunden eine beliebig sinnlose Aktionsanforderung stellt. Zudem ist das Verhalten hinsichtlich der internen Einflüsse unintuitiv. Da für jeden Zeitfortschritt interne Einflüsse von prodInfluence(E, S_t) erzeugt werden, könnte ein einzelner Agent durch ein Fluten der Umgebung z. B. mit NoOp-Aktionsanforderungen dafür sorgen, dass sich z. B. mit hoher Wahrscheinlichkeit Nebel in einer Zelle bildet. Möglichkeit 2 und 3 weisen dieses Problem nicht auf.

Möglichkeit 2 weist die Nachteile von Möglichkeit 1 nicht auf, eine Ordnung auf den Aktionsanforderungen durch π ist jedoch

zwingend erforderlich, da zwei Agenten konfligierende Aktionen anfordern könnten. Ein Problem dieser Möglichkeit besteht darin, dass ein Agent den Fortschritt der Umgebung (bewusst oder durch technische Probleme verursacht) blockieren könnte. Im Regelfall wird daher zusätzlich eine Zeitschranke verwendet, welche nach Überschreitung auch dann die Überführung in einen neuen Umgebungszustand veranlasst, wenn noch nicht von jedem Agenten eine Aktionsanforderung vorliegt.

Möglichkeit 3 unterscheidet sich von Möglichkeit 2 mit Timeout dadurch, dass auch bei Vorliegen einer Aktionsanforderung von jedem Agenten der Zeitfortschritt erst zum nächsten Takt erfolgt. Dies hat den Vorteil, dass ein Agent sich seine Rechenzeit sehr genau einteilen kann. Während bei Möglichkeit 2 zwar die maximale Bedenkzeit für die aktuell zu stellende Aktionsanforderung bekannt ist, ist dennoch nicht genau bekannt, wann die Überführung in den nächsten Zustand stattfindet.[2] Dies kann z. B. dann eine Rolle spielen, wenn ein Agent die Zeit, in der er gerade über keine Aktionsanforderung nachdenken muss, für strategische Überlegungen nutzt.

Für Möglichkeiten 2 mit Timeout-Timer und 3 können Agenten zudem ihre bereits gestellten Aktionsanforderungen revidieren, indem sie vor Auslösung der Zustandsüberführung eine neue Aktionsanforderung stellen. Ein Agent kann so zuerst *greedy* eine naheliegende Aktionsanforderung wählen und danach versuchen, eine bessere Aktionsanforderung zu berechnen, sofern ihm die Zeit dafür noch reicht. Möglichkeit 3 bietet auch hier den Vorteil, dass der genaue Zeitpunkt der nächsten Zustandsüberführung im Vorhinein bekannt ist.

2 Ein Agent könnte natürlich erzwingen, dass Möglichkeit 2 effektiv taktgesteuert wird, indem er seine Aktionsanforderungen so stellt, dass sie infinitesimal kurz vor Übertreten der Zeitschranke eintreffen.

5 WAHRNEHMUNG

In diesem Kapitel wird erläutert, welche Elemente der Umgebung von Agenten unter welchen Umständen wahrgenommen werden können.

5.1 EINLEITUNG

Agenten können die Umgebung nicht immer vollständig wahrnehmen. Im Folgenden wird dazu erläutert, welche Grundidee bei der Entwicklung von Wahrnehmung für Umgebungen im Vordergrund stand und von welchen Gegebenheiten es abhängen kann, welche Elemente einer Umgebung ein Agent wahrnimmt.

5.1.1 *Prototypisches Beispiel*

Im prototypischen Beispiel aus Kapitel 4.1.1 kann Tweety zwar schon auf vielfältige Weise in einer Umgebung agieren, aber das ist schwierig, weil er die Welt bislang noch nicht wahrnehmen kann und so gar nicht weiß, welche Dinge es überhaupt in der Welt gibt und ob seine Aktionsanforderungen erfolgreich sind oder nicht.

Tweety würde am liebsten gleich die ganze Welt vollständig wahrnehmen, dann wüsste er immer über alles Bescheid. Aber in der Welt gibt es Hindernisse, die ihm die Sicht versperren können, so dass Tweety nicht sieht, was sich hinter einer Mauer, einem Vorhang oder Nebel befindet. Immerhin ist der Nebel nie so dicht, dass er seine eigene Zelle nicht mehr sehen kann und wenn es nur Nebel in seiner Zelle gibt, dann kann er einfach zum Rand der Zelle gehen und in die nebelfreie Welt hinausblicken. Für Vorhänge gilt das genauso und Zellen mit Mauern kann er eh nicht betreten. Es gibt für Tweety Zellen, die er gar nicht wahrnehmen kann, weil die Sichtlinie zwischen der Zelle und ihm von einem anderen Hindernis unterbrochen wird, Zellen, bei denen er nur sieht, welches Hindernis sich dort befindet, nicht aber, welche Agenten und Objekte sich in der Zelle befinden und Zellen, die er vollständig wahrnehmen kann.

Aber selbst wenn ihm gerade keine Hindernisse die Sicht versperren, so ist Tweety leider etwas kurzsichtig und kann deshalb weniger weit sehen als die anderen Agenten in der Umgebung; von denen können zwar einige unbegrenzt weit sehen, wenn kein Hindernis im Weg steht, aber das sind die wenigsten.

5.1.2 *Hindernisse und Sichtweite*

Ein Agent kann in einer Umgebung Dinge wahrnehmen. Dazu erhält ein Agent a von der Umgebung zu jedem Zeitpunkt t eine Menge $O_{S_t,a}$ von Beobachtungen. Die Menge $O_{S_t,a}$ hängt dabei ausschließlich von a und vom Zustand S_t ab.

Die Teilmenge von S_t, welche die Positionen der Agenten und der sichtbehindernden Hindernisse beschreibt, ist dabei besonders relevant, da Agenten nicht wahrnehmen können, was sich von ihrer Position ausgehend hinter einem sichtbehindernden Hindernis befindet und sie die Agenten und Objekte in einer Zelle mit einem sichtbehindernden Hindernis nur dann wahrnehmen können, wenn sie sich selbst in dieser Zelle befinden.

Zudem kann ein Agent über eine eingeschränkte Sichtweite verfügen. Dies bedeutet, dass er auch dann den Inhalt einer Gridzelle nicht erkennen kann, wenn sich kein sichtbehinderndes Hindernis zwischen ihm und der Gridzelle befindet. In der realen Welt entspräche dies z. B. der eingeschränkten Sensorauflösung eines Agenten oder der Erdkrümmung. Die Sichtweite eines Agenten ist dabei im Umgebungszustand als Faktum folgenden Typs enthalten:

ViewRange(a, v): Die Sichtweite des Agenten a beträgt v.

Die folgenden Abschnitte diskutieren die Einschränkung der Wahrnehmungsmöglichkeiten durch sichtbehindernde Hindernisse und begrenzte Sichtweite sowie die Erzeugung der Beobachtungsmengen $O_{S_t,a}$.

5.2 SICHTBEHINDERNDE HINDERNISSE

In diesem Abschnitt wird zunächst der Ansatz diskutiert, zur Bestimmung der Sicht in Hinblick auf Hindernisse so vorzugehen, wie es für die reale Welt angemessen wäre. Darauffolgend wird ein diskreter Ansatz untersucht. Der Abschnitt wird komplettiert von Erläuterungen zur Bestimmung der Sichtbarkeit bzgl. Hindernissen.

5.2.1 *Realweltliche Sichtlinien*

In der realen Welt ist ohne Berücksichtigung der Sichtweite von einem Standort aus ein anderer Ort genau dann sichtbar, wenn die Strecke zwischen Standort und Ziel von keinem sichtbehindernden Hindernis geschnitten wird.

In Abbildung 2 kann der Agent a den markierten Zielpunkt sehen, weil die Strecke zwischen a und dem Zielpunkt von keinem

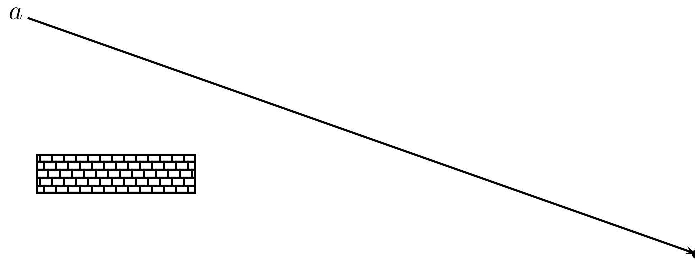

Abbildung 2: Sichtline von a zu einem Zielpunkt in der realen Welt ohne schneidendes Hindernis

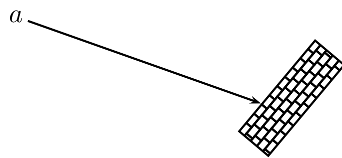

Abbildung 3: durch ein Hindernis unterbrochene Sichtlinie von a zu einem Zielpunkt in der realen Welt

Hindernis geschnitten wird. Wird wie in Abbildung 3 diese Strecke jedoch von einem Hindernis geschnitten, so ist der Zielpunkt für den Agenten a nicht sichtbar.

Im Unterschied zur realen Welt besteht eine von GridWorldSim simulierte Welt aus punktförmigen Gridzellen. Eine Gridzelle und die in ihr enthaltenen Objekte sind hinsichtlich ihrer Position nicht weiter aufteilbar: Entweder eine Gridzelle ist mitsamt ihren Objekten vollständig sichtbar oder gar nicht sichtbar. Ebenso kann eine Zelle entweder ein Hindernis vollständig beinhalten oder gar kein Hindernis beinhalten, eine feinere Abstufung als Gridzellenpositionen zur Positionierung von Hindernissen gibt es nicht.

Dennoch abstrahiert eine Gridzelle üblicherweise eine bestimmte Fläche der realen Welt. Ein naiver Ansatz bestünde deshalb darin, jeder Gridzelle eine zweidimensionale Fläche zuzuweisen und die Sichtbarkeit wie in der realen Welt zu berechnen (siehe Abbildung 4). Dieses Vorgehen ist jedoch mit Problemen behaftet:

- Im Zweidimensionalen hängt die Sichtbarkeit eines Objekts vom Standort des Objekts und des Agenten innerhalb der jeweiligen Zelle ab (siehe Abbildungen 5 und 6). Für die Gridwelt muss jedoch gelten, dass die Sichtbarkeit eines Objekts für einen Agenten bzgl. einer gegebenen Menge von Hindernissen ausschließlich davon abhängt, in welcher Zelle sich Agent und Objekt befinden.

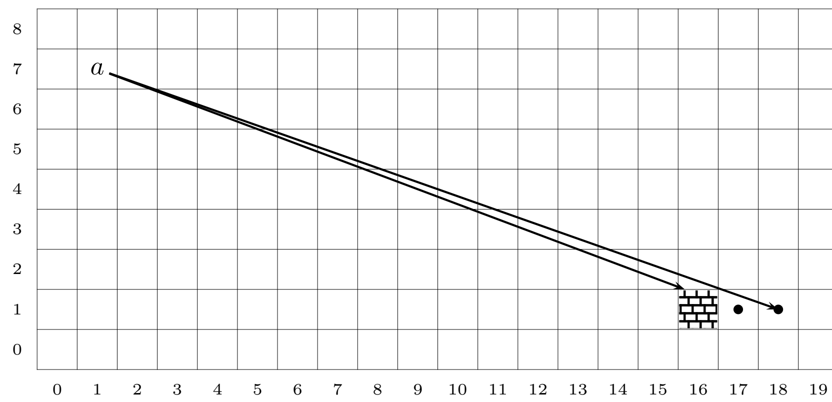

Abbildung 4: Sichtlinien von *a* zu den Mittelpunkten der Zellen $(17,1)$ und $(18,1)$

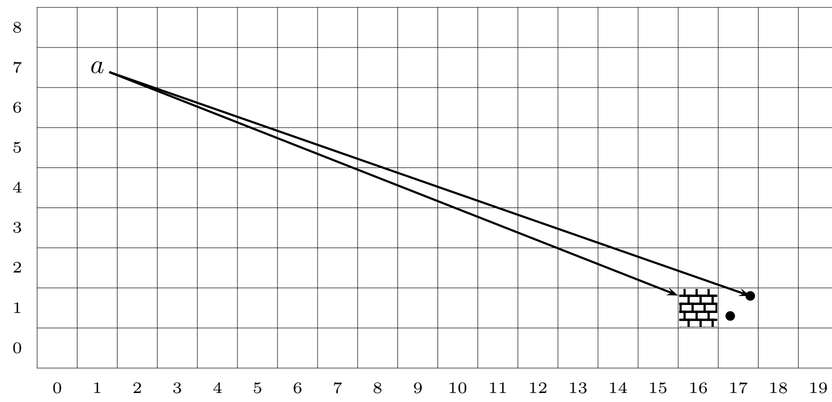

Abbildung 5: Sichtlinien von *a* zu zwei Zielpunkten in Zelle $(17,1)$

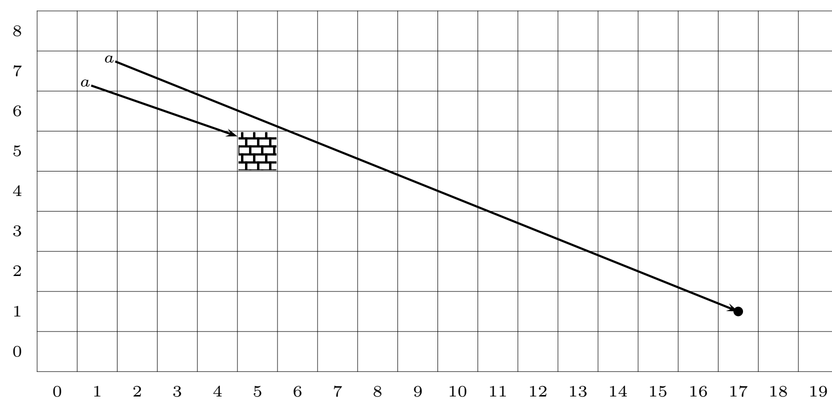

Abbildung 6: Sichtlinien von zwei verschiedenen Standorten in Zelle $(1,7)$ zu einem Zielpunkt

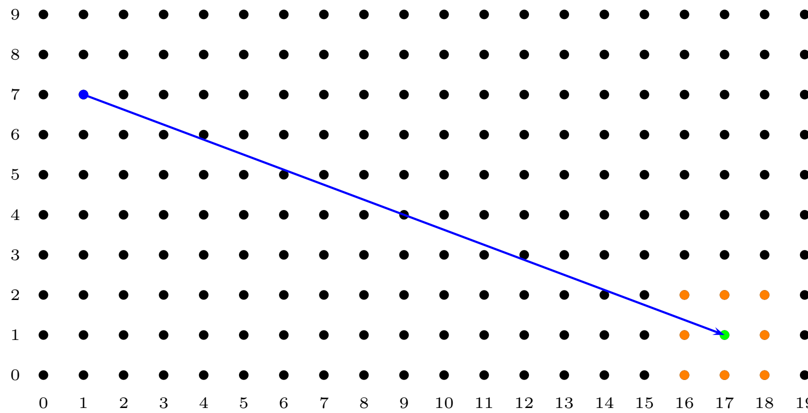

Abbildung 7: Sichtlinie von Zelle $(1,7)$ zu der von sichtbehindernden Hindernissen umgebenen Zelle $(17,1)$ (orangene Punkte markieren Hindernisse)

- Wenn Gridzellen und Hindernisse zweidimensional interpretiert werden, dann sollte dies in einer vollständigen zweidimensionalen Interpretation der Gridwelt auch für Agenten und Objekte gelten. Die zweidimensionale Fläche, die ein Agent oder Objekt einnimmt, ergibt sich jedoch nicht aus dem Zustand einer Gridwelt und selbst wenn dies der Fall wäre, so dürfte sie für die Sichtbarkeitsberechnung keine Rolle spielen. Zudem würde man bei zweidimensionalen Agenten und Objekten erwarten, dass diese je nach Standort nur teilweise sichtbar sein können; das Konzept partieller Sichtbarkeit existiert jedoch in Gridwelten ebenfalls nicht.

Es wäre möglich, diese Probleme dadurch zu umgehen, dass zunächst ein wünschenswertes Verhalten hinsichtlich der Sichtbarkeit bzgl. Hindernissen spezifiziert wird und darauf basierend Festlegungen getroffen werden (z. B. „zweidimensionales Grid, zweidimensionale Hindernisse, punktförmige Agenten und Objekte verortet im Mittelpunkt ihrer Zelle“), um dieses Verhalten zu erzwingen. Es erscheint jedoch wenig sinnvoll, eine Abstraktion zur Bestimmung der Sichtbarkeit bzgl. Hindernissen zu wählen, die derart von einer Menge von Ausnahmen geprägt wäre.

Stattdessen sollen Gridzellen als Punkte aufgefasst werden. In diesem Fall führt jedoch die Verwendung realweltlicher Sichtlinien zu keinen wünschenswerten Ergebnissen. Abbildung 7 zeigt eine Sichtlinie zu der vollständig von Hindernissen umgebenen Zelle $(17,1)$. Die Zelle wäre dennoch sichtbar, da kein Hindernis die Geradengleichung der Sichtlinie erfüllt.

Aufgrund der genannten Probleme wird von dem Ansatz, die Wahrnehmungsbeschränkung in der Gridwelt durch Verwendung realweltlicher Sichtlinien zu realisieren, Abstand genommen.

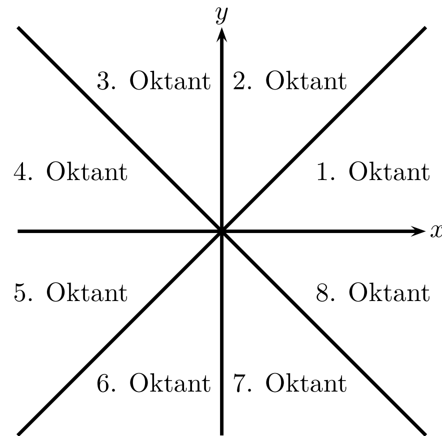

Abbildung 8: Oktanten eines zweidimensionalen kartesischen Koordinatensystems

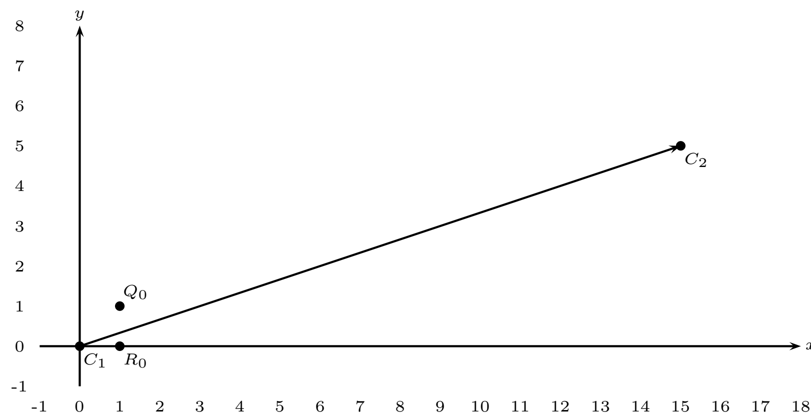

Abbildung 9: Koordinatensystem mit C_1 als Ursprung und C_2 im ersten Oktanten

5.2.2 *Diskrete Sichtlinien*

Statt realweltliche Sichtlinien zur Bestimmung der Sichtbarkeit zu verwenden, sollen *diskrete Sichtlinien* Verwendung finden. Eine diskrete Sichtlinie ist dabei eine diskrete Approximation der zugehörigen realweltlichen Sichtlinie. Es sollen also die Punkte (d. h. Zellen) bestimmt werden, durch welche die (nunmehr diskrete) Sicht verläuft. Im Beispiel von Abbildung 7 sind dies die Punkte, welche die eingezeichnete realtweltliche Sichtlinie approximieren.

Das Problem der Diskretisierung von Geraden bzw. Strecken stellt sich in vielfältigen Einsatzszenarien, zum Beispiel bei der diskreten Ansteuerung eines Druckers oder Monitors mit Vektorgrafiken. Das vorgestellte Vorgehen basiert auf der Arbeit von Bresenham zur Ansteuerung eines digitalen Plotters (vgl. [Bre65]).

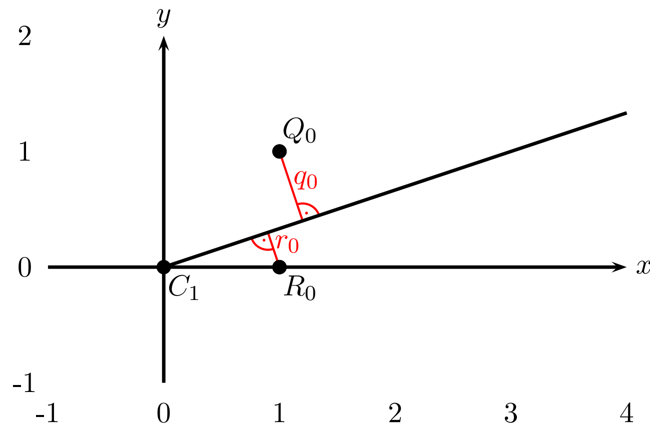

Abbildung 10: Ausschnitt des Koordinatensystems aus Abbildung 9 mit q_0 und r_0

Befinde sich der Agent in Gridzelle $C_1 = (x_a, y_a)$ und sei es die Aufgabe, eine diskrete Sichtlinie zur Gridzelle $C_2 = (x_g, y_g)$ zu bestimmen. Gebe es ferner ein kartesisches Koordinatensystem mit Ursprungspunkt (x_a, y_a). In diesem Koordinatensystem gilt nun $C_1 = (0, 0)$ und $C_2 = (x_g - x_a, y_g - y_a)$. Befinde sich ferner C_2 o. B. d. A. im ersten Oktanten (siehe Abbildung 8) dieses Koordinatensystems. Da sich C_2 im ersten Oktanten des Koordinatensystems befindet, muss zur Diskretisierung der Sichtlinie jeweils nur die östliche und nord-östliche Zelle in Erwägung gezogen werden. Für den ersten Diskretisierungsschritt ausgehend von C_1 sind dies die Zellen $R_0 = (1, 0)$ in östlicher und $Q_0 = (1, 1)$ in nord-östlicher Richtung. Ein Beispiel dazu ist in Abbildung 9 gegeben.

Sei r_i der Abstand zwischen der aktuell betrachteten östlichen Zelle R_i und der realweltlichen Sichtlinie und sei q_i der Abstand zwischen der aktuell betrachteten nord-östlichen Zelle Q_i und der realweltlichen Sichtlinie (Beispiel siehe Abbildung 10). Ausgehend von C_1 werde P_0 mit $P_0 = R_0$ falls $r_0 < q_0$ und $P_0 = Q_0$ falls $r_0 \geq q_0$ die erste Zelle, die Bestandteil der diskreten Sichtlinie wird. Im Beispiel der Abbildungen 9 und 10 gilt $r_0 < q_0$ und damit $P_0 = R_0$.

Ausgehend von allen weiteren Punkten P_{i-1} gilt, dass $P_i = R_i$ falls $r_i < q_i$ und $P_i = Q_i$ falls $r_i \geq q_i$ solange, bis P_i dem Zielpunkt C_2 entspricht. Abbildung 11 zeigt den zweiten Schritt im Beispiel. Im nächsten Schritt würde nun $P_1 = Q_1$, da $r_1 \geq q_1$.

Bisher wurde vorausgesetzt, dass sich C_2 im ersten Oktanten des Koordinatensystems befindet. Befindet sich C_2 in einem anderen Oktanten, so bildet C_1 ebenfalls den Ursprungspunkt des Koordinatensystems, aber die Achsen des Koordinatensystems werden anders ausgerichtet und den Fällen $r_i < q_i$ und $r_i \geq q_i$ werden andere Richtungen zugeordnet. Sei d_1 die Richtung, die

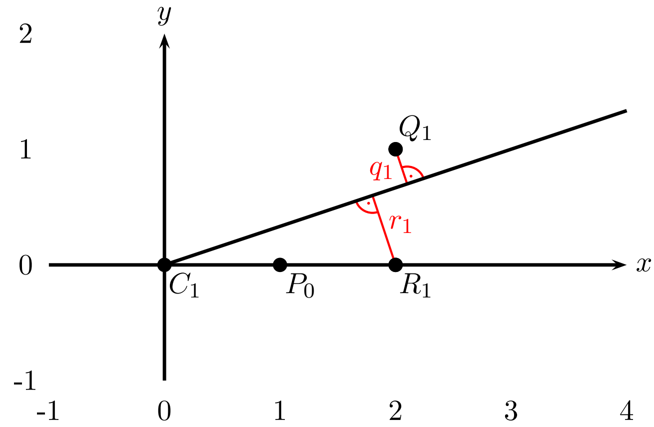

Abbildung 11: Folgezustand zur Situation in Abbildung 10

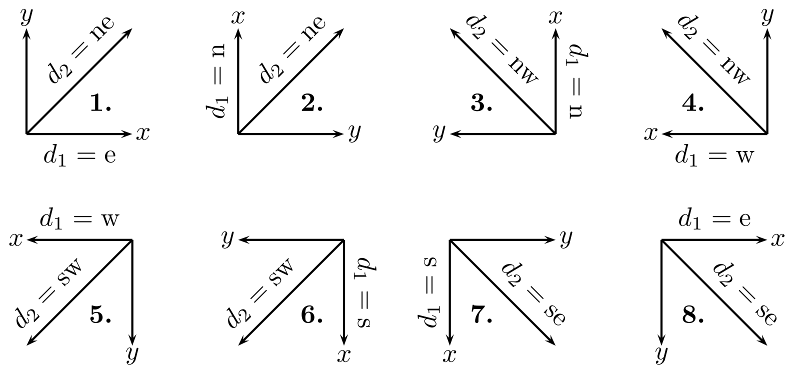

Abbildung 12: Ausrichtung des Koordinatensystems und Belegung der Richtungen für die verschiedenen Oktanten

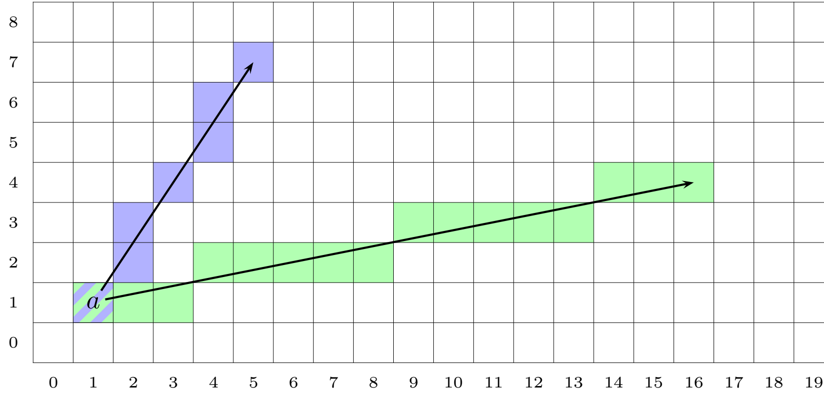

Abbildung 13: Diskrete Sichtlinien eines Agenten a zu zwei Zellen

im Falle $r_i < q_i$ und d_2 die Richtung, die im Falle $r_i \geq q_i$ beschritten wird. Abbildung 12 zeigt dann die Ausrichtung der Koordinatenachsen abhängig davon, in welchem Oktanten sich C_2 befindet sowie die zugehörige Belegung von d_1 und d_2.

Die diskreten Sichtlinien eines Agenten a zu zwei Zellen $(5,7)$ und $(16,4)$ sowie die zugehörigen realweltlichen Sichtlinien sind in Abbildung 13 gegeben.

Die Menge aller Zellen (x_i, y_i), die sich auf einer diskreten Sichtlinie von (x_a, y_a) nach (x_g, y_g) befinden, werde im Folgenden mit $\mathbf{dsl}(x_a, y_a, x_g, y_g)$ notiert.

5.2.3 *Wahrnehmung bzgl. Hindernissen*

Sei $\text{Loc}(x_a, y_a, a) \in S_t$. Im Folgenden soll die Menge $O^*_{S_t,a}$ der Beobachtungen eines Agenten a bzgl. Hindernissen im Zustand S_t bestimmt werden. Für jede Gridzelle $(x_g, y_g) \neq (x_a, y_a)$ der Gridwelt wird dabei wie folgt vorgegangen:

Es gilt

$$S_t \cap (\{\text{Grid}(x_g, y_g)\} \cup \{\text{Wall}(x_g, y_g)\} \cup \{\text{Fog}(x_g, y_g)\} \\ \cup \{\text{Curtain}(x_g, y_g)\}) \subseteq O^*_{S_t,a}$$

genau dann, wenn in S_t gilt

$$\forall (x_i, y_i) \in \text{dsl}(x_a, y_a, x_g, y_g) \setminus \{(x_a, y_a), (x_g, y_g)\} : \\ \neg\text{Wall}(x_i, y_i) \wedge \neg\text{Fog}(x_i, y_i) \wedge \neg\text{Curtain}(x_i, y_i).$$

Damit gilt, dass ein Agent das Vorhandensein (Grid(x, y)) einer Zelle und die Typen der sichtbehindernden Hindernisse genau dann wahrnehmen kann, wenn sich auf der diskreten Sichtlinie zwischen Agent und Zelle kein sichtbehinderndes Hindernis befindet. Dabei wird die erste und letzte Zelle der diskreten

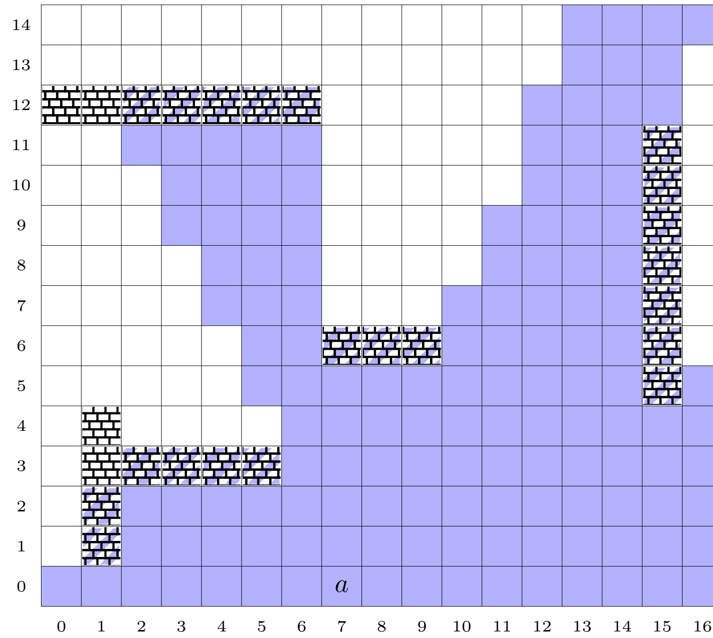

Abbildung 14: Sichtbarkeit bzgl. Hindernissen im Beispiel

Sichtlinie nicht berücksichtigt: Ein sichtbehinderndes Hindernis in der Zelle des Agenten beeinträchtigt seine Sicht daher nicht, ebenso kann er den Typ des sichtbehindernden Hindernisses in der betrachteten Zelle erkennen, sofern kein anderes Hindernis davor die Sicht versperrt.

Für die Sichtbarkeit von (anderen) Agenten und Objekten gilt für alle Zellen $(x_g, y_g) \neq (x_a, y_a)$

$$S_t \cap (\{\text{Trench}(x_g, y_g)\} \cup \{\text{Object}(o) \mid \text{Loc}(x_g, y_g, o) \in S_t\} \\ \cup \{\text{Agent}(z) \mid \text{Loc}(x_g, y_g, z) \in S_t\}) \subseteq O^*_{S_t,a}$$

genau dann, wenn in S_t gilt

$$\forall (x_i, y_i) \in \text{dsl}(x_a, y_a, x_g, y_g) \backslash \{(x_a, y_a)\} : \\ \neg\text{Wall}(x_i, y_i) \wedge \neg\text{Fog}(x_i, y_i) \wedge \neg\text{Curtain}(x_i, y_i).$$

Im Unterschied zur Wahrnehmung der Existenz einer Gridzelle und des Typs eines sichtbehindernden Hindernisses in dieser Zelle können Objekte, (andere) Agenten und Gräben nicht wahrgenommen werden, wenn die betrachtete Zelle ein sichtbehinderndes Hindernis beinhaltet. Ansonsten sind die Bedingungen identisch.

Abbildung 14 zeigt die Sichtbarkeit bzgl. Hindernissen im Beispiel. Für gestreifte Zellen sind die Hindernisse in dieser Zelle für den Agenten wahrnehmbar, nicht jedoch eventuell dort vorhandene Objekte. Die Abbildung berücksichtigt nicht, dass die immer sichtbare Zelle, in der sich der Agent aufhält, von der vorherigen Definition noch nicht erfasst ist.

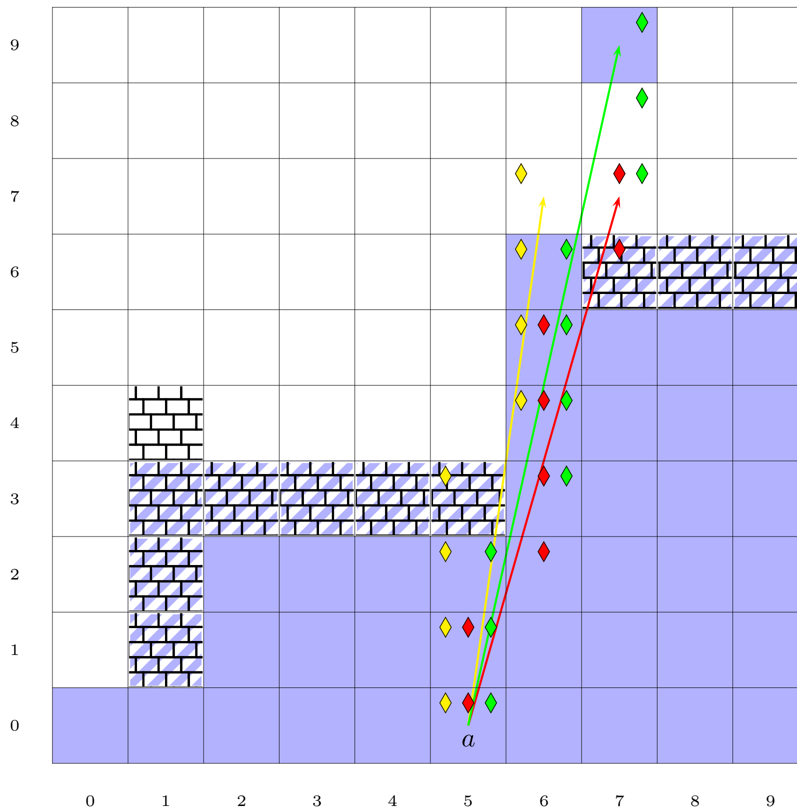

Abbildung 15: Sichtbarkeit bzgl. Hindernissen im Beispiel

In Abbildung 15 symbolisieren die verschiedenfarbigen Rauten die diskreten Sichtlinien zu verschiedenen Zellen. Die farbigen Linien stellen zum Vergleich die zugehörigen realweltlichen Sichtlinien zu den Zellmittelpunkten dar. Die Sichtbarkeit von Zelle $(7,9)$ erscheint bei dieser Darstellung der Gridzellen als Flächen unintuitiv zu sein, sind doch die aus Sicht des Agenten davorliegenden Zellen $(6,7)$ und $(7,7)$ (sowie $(6,8)$ und $(7,8)$) nicht sichtbar.

Dieser Eindruck entsteht durch die zweidimensionale Darstellung der Gridzellen als Flächen, obwohl es sich eigentlich nur um Punkte handelt. Interpretiert man die sichtbehindernden Hindernisse als Flächen und legt die realweltlichen Sichtlinien zu den Zellmittelpunkten zu Grunde, so stimmt die Intuition hingegen: Der Zellmittelpunkt von Zelle $(7,9)$ ist sichtbar, während die Zellmittelpunkte der Zellen $(6,7)$ und $(7,7)$ nicht sichtbar sind.

Das Konzept der Gridwelt mit Gridzellen als im örtlichen Sinne nicht weiter unterteilbaren Einheiten schließt eine zur realen Welt vollkommen analoge Sichtbarkeitsbestimmung inhärent aus. Wären im Beispiel aus Abbildung 15 die Zellen $(6,7)$ oder $(7,7)$ und $(6,8)$ oder $(7,8)$ sichtbar, so widerspräche dies ebenfalls der Intuition, da die Zellmittelpunkte von den eingezeichneten realweltlichen Sichtlinien nur durch Durchkreuzung sichtbehin-

dernder Hindernisse erreicht werden. Ebenso würde eine Nichtsichtbarkeit von Zelle $(7,9)$ die Frage aufwerfen, warum diese Zelle nicht sichtbar ist, obwohl die auf der zugehörigen realweltlichen Sichtlinie basierende Intuition dies fordern würde.

Letztlich dient die in Abbildung 15 gewählte Darstellung der Gridzellen als Flächen vor allem der besseren Übersichtlichkeit und soll nicht darüber hinwegtäuschen, dass Gridzellen keine zweidimensionale Ausdehnung besitzen. Nichtsdestotrotz stellt der in der Abbildung dargestellte Fall die Ausnahme dar: In den meisten Fällen wird der Intuition wie in Abbildung 14 entsprochen. Zudem lässt sich die Intensität, mit der dieses auf den ersten Blick unintutive Verhalten auftritt, durch die Wahl einer geeigneten Granularität des Grids reduzieren und kann dadurch die reale Welt besser approximieren.

5.3 SICHTWEITE

Die Wahrnehmung eines Agenten a kann nicht nur durch Hindernisse beschränkt sein, sondern auch durch seine maximale Sichtweite v gemäß $\text{ViewRange}(a,v)$. Zur Bestimmung der Zellen, die innerhalb der Sichtweite eines Agenten liegen, soll der euklidische Abstand dienen, da dieser bei der Betrachtung des Grids als Abstraktion einer Fläche der realen Welt der realweltlichen Distanz zweier Orte entspricht. Dazu findet die Funktion euklid aus Kapitel 4.5 Verwendung. Durch Verwendung einer anderen Funktion wäre die Verwendung anderer Distanzmaße jedoch gleichermaßen denkbar.

Sei $\text{Loc}(x_a, y_a, a) \in S_t$ und $\text{ViewRange}(a,v) \in S_t$. Es soll im Folgenden die Menge $O^+_{S_t,a}$ der Beobachtungen eines Agenten a bzgl. seiner Sichtweite im Zustand S_t bestimmt werden. Für jede Gridzelle $(x_g, y_g) \neq (x_a, y_a)$ der Gridwelt wird dabei wie folgt vorgegangen:

Es gilt

$$
\begin{aligned}
&S_t \cap (\{\text{Grid}(x_g,y_g)\} \cup \{\text{Wall}(x_g,y_g)\} \cup \{\text{Fog}(x_g,y_g)\} \\
&\cup \{\text{Curtain}(x_g,y_g)\} \cup \{\text{Trench}(x_g,y_g)\} \\
&\cup \{\text{Object}(o) \mid \text{Loc}(x_g,y_g,o) \in S_t\} \\
&\qquad\qquad \cup \{\text{Agent}(z) \mid \text{Loc}(x_g,y_g,z) \in S_t\}) \subseteq O^+_{S_t,a}
\end{aligned}
$$

genau dann, wenn gilt

$$\text{euklid}(x_a, y_a, x_g, y_g) < v.$$

Abbildung 16 zeigt die Sichtbarkeit bzgl. Sichtweite im Beispiel für $v = 4$. Im Vorgriff auf die nachfolgende Beobachtungsbestimmung ist die immer sichtbare Zelle, in der sich der Agent aufhält, auch hier als sichtbar markiert.

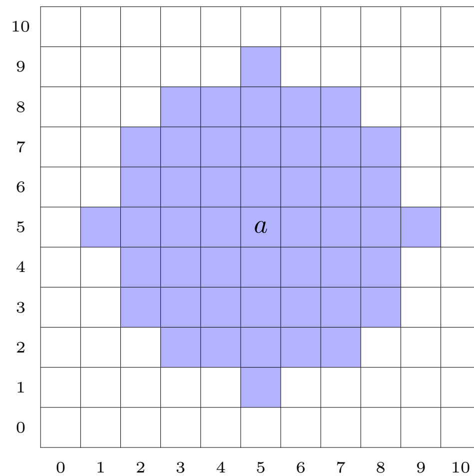

Abbildung 16: Sichtbarkeit bzgl. Sichtweite im Beispiel mit $v = 4$

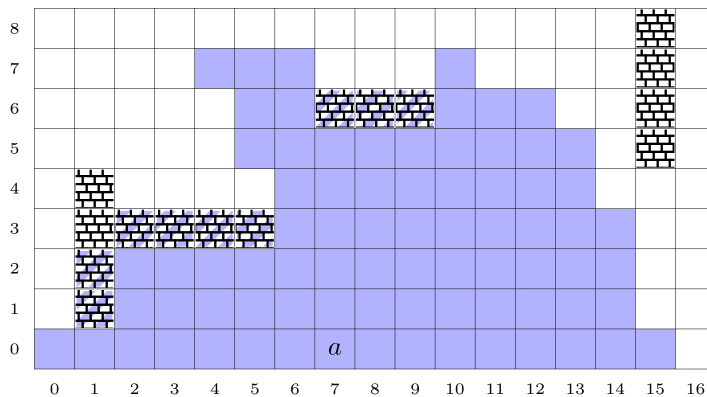

Abbildung 17: Beispiel aus Abbildung 14 mit zusätzlicher Beschränkung durch eine Sichtweite von 8

5.4 Bestimmung der Beobachtungsmenge

In den Kapiteln 5.2.3 und 5.3 wurden die Mengen $O^*_{S_t,a}$ der Wahrnehmung bzgl. Hindernissen und $O^+_{S_t,a}$ der Wahrnehmung bzgl. Sichtweite bestimmt. Zur Bestimmung der Menge $O_{S_t,a}$ aller Wahrnehmungen eines Agenten a im Zustand S_t sind diese Mengen noch nicht ausreichend. Zum einen wurde die aktuelle Zelle des Agenten noch nicht betrachtet, zum anderen fehlen noch verschiedene beobachtbare Fakten, z. B. vom Typ $\text{Loc}(x, y, z)$ oder $\text{Contains}(z, o)$.

Sei $O^0_{S_t,a} = O^*_{S_t,a} \cap O^+_{S_t,a}$. Diese Menge beinhaltet alle vom Agenten außerhalb seiner eigenen Zelle wahrnehmbaren Fakten vom Typ $\text{Object}(o)$, $\text{Agent}(a)$, $\text{Grid}(x, y)$, $\text{Trench}(x, y)$, $\text{Wall}(x, y)$, $\text{Curtain}(x, y)$ und $\text{Fog}(x, y)$. Insbesondere beinhaltet die Menge alle für die Sicht relevanten Informationen: Die noch fehlenden Beobachtungen ergeben sich daher ohne Sichtberechnungen aus S_t und $O^0_{S_t,a}$.

Abbildung 17 veranschaulicht die Sichtinformation einer Menge $O^0_{S_t,a}$ durch Kombination des Beispiels aus Abbildung 14 mit einer Sichtweite von 8. Da Objekte und Agenten in $O^0_{S_t,a}$ dem Grid noch nicht über Fakten des Typs $\text{Loc}(x,y,a)$ zugeordnet sind, behinhaltet die Abbildung keine Agenten und Objekte (das eingezeichnete a dient nur der Orientierung).

Zunächst sollen die Fakten der Typen $\text{Object}(o)$, $\text{Agent}(a)$, $\text{Grid}(x,y)$, $\text{Curtain}(x,y)$ und $\text{Fog}(x,y)$ für die aktuelle Zelle des Agenten a Berücksichtigung finden. Fakten vom Typ $\text{Trench}(x,y)$ und $\text{Wall}(x,y)$ finden keine Berücksichtigung, da der Agent sich in einer Zelle, in der sich ein solches Hindernis befindet, nicht aufhalten könnte. Ferner kann ein Agent den Inhalt seiner Zelle immer wahrnehmen, also auch dann, wenn es einen Vorhang oder Nebel in der Zelle gibt. Sei erneut $\text{Loc}(x_a,y_a,a) \in S_t$:

$$\begin{aligned}O^1_{S_t,a} = O^0_{S_t,a} &\cup (S_t \cap (\{\text{Grid}(x_a,y_a)\} \cup \{\text{Fog}(x_a,y_a)\}\\ &\cup \{\text{Curtain}(x_a,y_a)\} \cup \{\text{Object}(o) \mid \text{Loc}(x_a,y_a,o) \in S_t\}\\ &\cup \{\text{Agent}(z) \mid \text{Loc}(x_a,y_a,z) \in S_t\}))\end{aligned}$$

$O^1_{S_t,a}$ beinhaltet alle vom Agenten beobachtbaren auf dem Grid verorteten Objekte, Agenten, Zellen und Zelltypen. Im nächsten Schritt sollen alle Fakten vom Typ $\text{Loc}(x,y,z)$ für die beobachtbaren Objekte und Agenten der Beobachtungsmenge $O^1_{S_t,a}$ hinzugefügt werden. Für die resultierende Menge $O^2_{S_t,a}$ gilt:

$$\begin{aligned}O^2_{S_t,a} = O^1_{S_t,a} \cup \{\text{Loc}(x,y,z) \in S_t \mid &\text{Object}(z) \in O^1_{S_t,a}\\ &\vee \text{Agent}(z) \in O^1_{S_t,a}\}\end{aligned}$$

In der Beobachtungmenge $O^2_{S_t,a}$ existieren keine Fakten, welche die freien Kapazitäten und die Kapazitätsbedarfe der beobachtbaren Objekten und Agenten und die freien Kapazitäten von Zellen beschreiben. Dies soll in der Menge $O^3_{S_t,a}$ nicht mehr der Fall sein. Daher gilt:

$$\begin{aligned}O^3_{S_t,a} = O^2_{S_t,a} &\cup \{\text{FreeCap}(z,c) \in S_t \mid \text{Object}(z) \in O^2_{S_t,a}\\ &\vee \text{Agent}(z) \in O^2_{S_t,a}\}\\ &\cup \{\text{CapNeed}(z,c) \in S_t \mid \text{Object}(z) \in O^2_{S_t,a}\\ &\vee \text{Agent}(z) \in O^2_{S_t,a}\}\\ &\cup \{\text{FreeCellCap}(x,y,c) \in S_t \mid \text{Grid}(x,y) \in O^2_{S_t,a}\}\end{aligned}$$

In der Menge $O^4_{S_t,a}$ sollen zusätzlich zu den Beobachtungen der Menge $O^3_{S_t,a}$ Beobachtungen über Schließfächer enthalten sein:

$$\begin{aligned}O^4_{S_t,a} = O^3_{S_t,a} &\cup \{\text{SafeDepositBox}(o) \in S_t \mid \text{Object}(o) \in O^3_{S_t,a}\}\\ &\cup \{\text{Locked}(o) \in S_t \mid \text{Object}(o) \in O^3_{S_t,a}\}\end{aligned}$$

In Kapitel 4.2 werden benutzerdefinierte Objekttypen diskutiert, die bspw. von Agenten dazu verwendet werden können, um Objekten eine Wertschätzung beizumessen. In der Menge $O^5_{S_t,a}$ sollen auch diese Objekttypen beobachtbar sein. Daher gelte für alle benutzerdefinierten Objekttypen $U(o)$:

$$O^5_{S_t,a} = O^4_{S_t,a} \cup \{U(o) \in S_t \mid \text{Object}(o) \in O^4_{S_t,a}\}$$

Inventare von Agenten sind privat, außer ein Agent wurde zusätzlich als Objekt deklariert. Ein Agent kann daher nur beobachten, welche Objekte sich in seinem eigenen Inventar befinden. Die Enthaltenseinsbeziehung zwischen Objekten und die in einem beobachtbaren Objekt enthaltenen Objekte sind hingegen öffentlich und beobachtbar. Dies gilt jedoch nicht für Objekte, die in einem Objekt enthalten sind, welches in einem Objekt enthalten ist. Ebenso sind Objekte, die sich in einem verschlossenen Schließfach befinden, nicht beobachtbar.

Die Menge $O^6_{S_t,a}$ berücksichtigt das Inventar des Agenten a selbst:

$$\begin{aligned} O^6_{S_t,a} = O^5_{S_t,a} \cup \{\text{Object}(o) \in S_t \mid \text{Contains}(a,o) \in S_t\} \\ \cup \{\text{Contains}(a,o) \in S_t\} \end{aligned}$$

Alle beobachtbaren Objekte, jedoch noch nicht alle Enthaltenseinsbeziehungen, sind in der Menge $O^7_{S_t,a}$ enthalten:

$$\begin{aligned} O^7_{S_t,a} = O^6_{S_t,a} \cup \{\text{Object}(o) \in S_t \mid \text{Contains}(z,o) \in S_t \\ \wedge \text{Object}(z) \in O^6_{S_t,a} \wedge \text{Locked}(z) \notin S_t\} \end{aligned}$$

Bis auf die Enthaltenseinsbeziehungen zwischen Objekten sind in $O^7_{S_t,a}$ alle beobachtbaren Fakten enthalten. Sei daher

$$\begin{aligned} O^8_{S_t,a} = O^7_{S_t,a} \cup \{\text{Contains}(z,o) \in S_t \mid \text{Object}(z) \in O^7_{S_t,a} \\ \wedge \text{Object}(o) \in O^7_{S_t,a}\} \end{aligned}$$

die Menge aller Beobachtungen eines Agenten a im Zustand S_t mit Ausnahme von Nachrichten, welche im folgenden Kapitel diskutiert werden.

6 KOMMUNIKATION

In diesem Kapitel wird das Konzept zur Kommunikation zwischen Agenten vorgestellt.

6.1 EINLEITUNG

Kommunikation ist in Umgebungen nicht uneingeschränkt möglich. Im Folgenden soll deshalb skizziert werden, von welchen Einflussgrößen Kommunikation in Umgebungen abhängt und wie sich Kommunikation in Bezug auf Umgebungszustand und Wahrnehmung darstellt.

6.1.1 *Prototypisches Beispiel*

Tweety kann zwar jetzt in der Umgebung agieren (siehe Kapitel 4.1.1) und diese auch wahrnehmen (siehe Kapitel 5.1.1), aber er würde auch gerne mit den anderen Agenten in der Umgebung reden können. Glücklicherweise gibt es in der Umgebung Schallwellen, so dass andere Agenten hören können, was er sagt und er auch hören kann, was andere Agenten sagen.

Je weiter ein anderer Agent, mit dem er sprechen möchte, von ihm entfernt ist, desto stärker werden die von ihm erzeugten Schallwellen gedämpft. Deshalb ist es bei der Kommunikation für ihn von Vorteil, eine laute Stimme zu haben. Wenn er anderen Agenten zuhört, die sich etwas weiter entfernt von ihm befinden, kann er sie trotzdem oft noch hören, weil er sehr gute Ohren hat. Sein Freund Geeko ist hingegen etwas schwerhörig und muss anderen Agenten daher etwas näher kommen, wenn er hören möchte, was sie sagen.

Leider gibt es in der Gridwelt Antischallgeneratoren, die von den Agenten auch Interferenzen genannt werden. Treffen die Schallwellen eines Agenten auf so eine Interferenz, können die sich dahinter befindlichen Agenten – ähnlich wie bei den sichtbehindernden Hindernissen im Bereich der Wahrnehmung – nicht hören, was der Agent gesagt hat. Glücklicherweise sind wenigstens Mauern, Vorhänge, Nebel und Gräben schalldurchlässig, es sei denn, es befindet sich auch eine Interferenz in ihrer Zelle.

Agenten haben in der Gridwelt sehr generische Stimmen und können deshalb einander an der Stimme nicht erkennen. Wenn Tweety einen Agenten sprechen hört, den er nicht sehen kann,

dann weiß er deshalb nicht automatisch, um welchen Agenten es sich handelt.

Befindet er sich in der gleichen Zelle wie ein anderer Agent, kann er mit diesem flüstern, ohne dass ein anderer Agent davon etwas mitbekommt. Ist sein Kommunikationspartner jedoch nicht in der gleichen Zelle wie er, dann können potentiell auch andere Agenten hören, was er sagt, weil er laut sprechen muss.

6.1.2 *Allgemeines Vorgehen*

In Kapitel 3.1.2.2 und 3.1.2.4 wird die Zielsetzung von Kommunikation bei GridWorldSim bereits erläutert: Die Kommunikation zwischen Agenten findet über die Umgebung statt, da diese die Kommunikation abhängig vom aktuellen Umgebungszustand einschränkt. Dabei wird zwischen öffentlichen Nachrichten, die potenziell von allen Agenten wahrgenommen werden können, und privaten Nachrichten, die nur der intendierte Empfänger wahrnehmen kann, unterschieden.

Nachrichten sind dabei Teil des Umgebungzustandes, wobei eine Nachricht aus einem Zustand S_t nicht mehr in S_{t+1} enthalten sein soll. Ein Agent a nimmt die für ihn empfangbaren Nachrichten als Teil seine Beobachtungsmenge $O_{S_t,a}$ im Sinne von Kapitel 5 wahr.

In der realen Welt hängt der Erfolg, ob eine per Schall oder Funk versandte Nachricht tatsächlich beim Empfänger ankommt, vor allem von drei Einflussgrößen ab:

1. der Empfindlichkeit der akustischen Wahrnehmung bzw. des Empfangsmoduls des Empfängers (im Folgenden als *Empfangsempfindlichkeit* bezeichnet)

2. der Schalldruckpegel bzw. die Sendeintensität mit dem die Nachricht erzeugt wird (im Folgenden als *Sendestärke* bezeichnet)

3. kommunikationsblockierenden Hindernissen zwischen Absender und Empfänger

Für diese drei Einflussgrößen soll eine geeignete Gridwelt-Abstraktion sowie ein Verfahren zur Bestimmung von Kommunikationseinschränkungen gefunden werden.

6.2 NACHRICHTENTYPEN

Es gibt zwei Ausprägungen von Nachrichten: *öffentliche Nachrichten* und *private Nachrichten*. Eine öffentliche Nachricht hat keinen intendierten Empfänger und wird von allen Agenten empfangen, für die es die Kombination aus Empfangsempfindlichkeit,

Sendestärke und kommunikationsblockierenden Hindernissen erlaubt. Eine private Nachricht hingegen hat immer einen bekannten Absender und zudem einen intendierten Empfänger. Private Nachrichten können nur zwischen Agenten verschickt werden, die sich in der gleichen Zelle befinden. Möchte ein Agent nichtöffentliche Informationen an einen anderen Agenten verschicken, der sich nicht in der gleichen Zelle befindet, so könnte er sich unabhängig von der Umgebung mit dem Empfänger auf ein Verschlüsselungsverfahren einigen und den Nachrichteninhalt verschlüsselt als öffentliche Nachricht versenden.

In der realen Welt ist es u. U. nicht möglich, den Absender einer öffentlichen Nachricht zu erkennen, z. B. wenn bei einer akustischen Nachricht keine Sichtverbindung zwischen Absender und Empfänger besteht und die Stimme des Absenders vom Empfänger nicht erkennbar ist. GridWorldSim unterstützt optional, dass der Absender vom Empfänger einer Nachricht nicht wahrgenommen werden kann, wenn keine Sichtverbindung zwischen Absender und Empfänger besteht. Ist diese Funktion aktiviert nimmt ein Empfänger, welcher zwar die Nachricht eines Absenders empfangen, diesen aber nicht sehen kann, eine öffentliche Nachricht ohne Absender wahr.

Zur Kodierung von öffentlichen Nachrichten im Umgebungszustand und zur Wahrnehmung von öffentlichen Nachrichten durch Agenten sind zwei verschiedene Ansätze denkbar:

1. Es werden für jede öffentliche Nachricht unabhängig von der Anzahl der Agenten Fakten erzeugt, welche dem Umgebungszustand hinzugefügt werden und die Wahrnehmung bestimmt, welche Agenten diese Fakten vollständig, teilweise oder gar nicht wahrnehmen können. Naheliegend wäre dabei eine Kodierung, in welcher der öffentliche Nachrichtentyp nur über Nachrichteninhalt und einen eindeutigen Identifikator verfügt (z. B. PubMsg(m, id)) und der Absender über Fakten eines separaten Typs zugeordnet wird (z. B. Sender(id, a)). Die Wahrnehmung würde nun bestimmen, welche Agenten die Nachricht nicht wahrnehmen, nur die Nachricht ohne Absender wahrnehmen oder die Nachricht mit Absender wahrnehmen.

2. Es wird für jeden Empfänger einer öffentlichen Nachricht ein eigenes Faktum dem Umgebungszustand hinzugefügt, wobei es zwei Typen von Fakten gibt: Einen für öffentliche Nachrichten, bei denen der Empfänger den Absender wahrnehmen kann und einen für öffentliche Nachrichten, bei denen dies nicht der Fall ist.

Beispiel 14: Angenommen, eine Umgebung realisiert Ansatz 1 und es gibt drei Agenten in der Umgebung: Tweety, Geeko und

Tux. Tweety verschickt eine öffentliche Nachricht mit dem Inhalt „Fish!“. Dazu werden dem Umgebungszustand zwei Fakten hinzugefügt: PubMsg(„Fish!“, 0) und Sender(0, tweety), wobei 0 einen Identifikator darstellt. Angenommen, Geeko kann Tweety sehen und deshalb den Absender der Nachricht wahrnehmen und Tux kann Tweety nicht sehen und daher den Absender der Nachricht nicht wahrnehmen. Im Zuge der Wahrnehmung erhält Geeko deshalb die Fakten PubMsg(„Fish!“, 0) und Sender(0, tweety) und kennt somit den Absender der Nachricht, Tux hingegen erhält nur PubMsg(„Fish!“, 0) und kennt damit zwar den Inhalt der Nachricht, nicht jedoch den Absender.

Beispiel 15: Angenommen es besteht die gleiche Ausgangssituation wie im letzten Beispiel, mit dem Unterschied, dass die Umgebung Ansatz 2 realisiert. Es würde nun nach Absenden der Nachricht durch Tweety für jeden der beiden anderen Agenten ein eigenes Faktum dem Umgebungszustand hinzugefügt: PubMsg(tweety, geeko, „Fish!“) für Geeko und das Faktum AnonPubMsg(tux, „Fish!“) für Tux. Die Wahrnehmung wäre – von der unterschiedlichen Kodierung abgesehen – die gleiche wie bei Ansatz 1, mit dem Unterschied, dass bei der Nachrichtenerstellung entschieden wurde, welcher Agent die Nachricht erhält und nicht bei der Erzeugung der Wahrnehmung.

Bei genauerer Betrachtung erscheint Möglichkeit 2 konsistenter mit der Vorgehensweise zur Wahrnehmung in Kapitel 5. Die Wahrnehmung von Agenten, Objekten und Hindernissen erfolgt in der realen Welt durch elektromagnetische Wellen (z. B. Licht, Infrarot, Radar), die in einer Gridwelt nicht explizit im Zustand modelliert sind. Die Wahrnehmung abstrahiert dabei geeignete Wellen, um zu bestimmen, welche Agenten, Objekte und Hindernisse für einen Agenten sichtbar sind. Im Falle von Nachrichten stellen die elektromagnetischen Wellen bzw. Schallwellen jedoch die wahrzunehmende Entität selbst dar und sind damit keine Entitäten, deren Beobachtbarkeit im Zuge der Wahrnehmung zu bestimmen wäre. Deshalb wird Möglichkeit 2 zur Modellierung verwendet, wodurch die Ausbreitung von Nachrichten durch Zustandsüberführungsregeln bestimmt wird. Ungeachtet der vom Autor als konsistenter empfundenen Modellierung durch den zweiten Ansatz, wäre der erste Ansatz gleichermaßen und ohne praktische Nachteile realisierbar.

Zur Kodierung im Umgebungszustand dienen dabei Fakten folgender Typen:

- **PubMsg**(a, a_1, m): Es liegt eine öffentliche Nachricht m mit erkennbarem Absender für Agent a_1 vor, welche von Agent a versandt wurde.

- **PrivMsg**(a, a_1, m): Es liegt eine private Nachricht m von Agent a für den Agenten a_1 vor.
- **MsgSightReq**: Die optionale Funktion, dass Empfängern der Absender von öffentlichen Nachrichten nur dann bekannt ist, wenn eine Sichtverbindung zwischen Absender und Empfänger besteht, ist aktiviert.
- **AnonPubMsg**(a_1, m): Agent a_1 erhält eine öffentliche Nachricht m mit unbekanntem Absender.

Zum Versand von Nachrichten dienen die folgenden Aktionsanforderungen:

- **SendPubMsg**(a, m): Der Agent a möchte eine öffentliche Nachricht m versenden.
- **SendPrivMsg**(a, a_1, m): Der Agent a möchte eine private Nachricht m an Agent a_1 versenden.

6.3 kommunikationsblockierende hindernisse

Zur Bestimmung der Einschränkung von Kommunikation durch kommunikationsblockierende Hindernisse findet die Funktion $\text{dsl}(x_a, y_a, x_g, y_g)$ aus Kapitel 5.2.2 Verwendung. Ähnlich der Einschränkung von Wahrnehmung durch Hindernisse (siehe Kapitel 5.2.3) wird Kommunikation verhindert, wenn sich auf der diskreten Sichtlinie zwischen zwei Agenten ein Hindernis der Ausprägung *kommunikationsblockierend* befindet und zwar auch dann, wenn sich das kommunikationsblockierende Hindernis in der Zelle eines der beiden Agenten befindet.

Die öffentliche Nachricht eines sendenden Agenten a in Zelle (x_1, y_1) kann von einem Agenten a_1 in Zelle (x_2, y_2) hinsichtlich kommunikationsblockierender Hindernisse genau dann empfangen werden, wenn gilt:

$$\forall (x_i, y_i) \in \text{dsl}(x_1, y_1, x_2, y_2) : \text{Interference}(x_i, y_i) \notin S_t$$

In der realen Welt gilt, dass Schall- und Funkwellen nicht auf direktem Wege vom Absender zum Empfänger gelangen müssen, da diese z. B. von Gebäuden reflektiert werden. Deshalb muss in der realen Welt nicht zwangsläufig eine direkte schall- bzw. funkwellendurchlässige Verbindung zwischen Absender und Empfänger bestehen, sondern Schall- bzw. Funkwellen können den Empfänger auch auf indirektem Wege erreichen. Für die Umgebung gilt jedoch, dass Schall- bzw. Funkwellen nicht an Mauern, Objekten und anderen Elementen der Gridwelt reflektieren. Zur erfolgreichen Kommunikation muss deshalb eine diskrete gerade Linie zwischen kommunizierenden Agenten bestehen, die nicht

durch ein kommunikationsblockierendes Hindernis unterbrochen ist.

6.4 EMPFANGSEMPFINDLICHKEIT UND SENDESTÄRKE

Für jeden Agenten, der über die Fähigkeit verfügt, Nachrichten zu erhalten, wird seine Empfangsempfindlichkeit im Umgebungszustand als Faktum des folgenden Typs kodiert:

> **Hearing**(a, h): Die Empfangsempfindlichkeit des Agenten a beträgt h.

Ein Agent, für den kein Faktum vom Typ Hearing(a, h) vorliegt, kann keine Nachrichten empfangen.

Für jeden Agenten, der über die Fähigkeit verfügt, Nachrichten zu versenden, wird seine Sendestärke im Umgebungszustand als Faktum folgenden Typs kodiert:

> **SoundIntensity**(a, s): Die Sendestärke des Agenten a beträgt s.

Ein Agent, für den kein Faktum vom Typ SoundIntensity(a, s) vorliegt, kann keine Nachrichten versenden.

Zur Bestimmung der Empfangbarkeit von Nachrichten bzgl. Empfangsempfindlichkeit und Sendestärke soll erneut die Funktion euklid(x_1, y_1, x_2, y_2) aus Kapitel 4.5 dienen. Die öffentliche Nachricht eines sendenden Agenten a mit Sendestärke s in Zelle (x_1, y_1) kann von einem Agenten a_1 mit Empfangsempfindlichkeit h in Zelle (x_2, y_2) hinsichtlich Empfangsempfindlichkeit und Sendestärke genau dann empfangen werden, wenn gilt:

$$h + s \geq \text{euklid}(x_1, y_1, x_2, y_2)$$

Wird die Kommunikation nicht durch kommunikationsblockierende Hindernisse weiter beschränkt, kann ein Agent a_1 die öffentliche Nachricht eines Agenten a genau dann empfangen, wenn die Summe seiner Empfangsempfindlichkeit und der Sendestärke des sendenden Agenten nicht kleiner ist als der euklidische Abstand der Zellpositionen beider Agenten.

Ist es gewünscht, dass nur die Empfangsempfindlichkeit oder nur die Sendestärke zur Einschränkung der Kommunikationsreichweite dienen soll, kann dies dadurch erreicht werden, dass der jeweils andere Wert mit 0 spezifiziert wird.

6.5 ERZEUGUNG VON NACHRICHTEN

Zur Erzeugung öffentlicher und privater Nachrichten dienen unterschiedliche Zustandsüberführungsregeln, die im Folgenden erläutert werden.

6.5.1 *Private Nachrichten*

Zur Erzeugung privater Nachrichten dient die Zustandsüberführungsregel **CreatePrivMsg**, welche Aktionsanforderungen vom Typ SendPrivMsg(a, a_1, m) behandelt. Der sendende und der empfangende Agent müssen sich in der gleichen Zelle befinden, der sendende Agent muss über die generelle Fähigkeit zum Versand von Nachrichten mit einer beliebigen Sendestärke verfügen, der empfangene Agent muss die generelle Fähigkeit besitzen, Nachrichten mit einer beliebigen Empfangsempfindlichkeit empfangen zu können und die Zelle, in der sich beide Agenten befinden, darf nicht durch Interferenz gestört sein. Sind diese Bedingungen erfüllt, wird ein entsprechendes PrivMsg(a, a_1, m)-Faktum dem Zustand hinzugefügt.

Die Zustandsüberführungsregel lautet wie folgt:

CreatePrivMsg	SendPrivMsg(a, a_1, m)
Pre:	$\{$Loc(x, y, a), Loc(x, y, a_1), SoundIntensity(a, s), Hearing(a_1, h), $\neg$Interference$(x, y)\}$
Prob:	1
Post:	$\{$PrivMsg$(a, a_1, m)\}$

6.5.2 *Öffentliche Nachrichten*

Gemäß Kapitel 4.6 gilt für jede ausgelöste Regel, dass für jede Belegung der freien Variablen, welche die Vorbedingungen erfüllt, die Nachbedingungen mit Wahrscheinlichkeit *Prob* angewendet werden. Wird eine Aktionsanforderung zum Nachrichtenversand einer öffentlichen Nachricht von einem Agenten a gestellt, so soll für alle Agenten, denen der Empfang möglich ist, ein entsprechendes Nachrichtenfaktum erzeugt werden.

Zuerst wird der Fall behandelt, dass MsgSightReq nicht gilt, also dass jeder empfangende Agent den Absender einer empfangbaren öffentlichen Nachricht kennt. Eine öffentliche Nachricht mit bekanntem Absender wird genau dann von einem anderen Agenten empfangen, wenn Empfangbarkeit bzgl. kommunikationsblockierenden Hindernissen und Empfangbarkeit bzgl. Empfangsempfindlichkeit und Sendestärke gegeben ist. Dazu dient die folgende Zustandsüberführungsregel **CreatePubMsgAll**:

CreatePubMsgAll	SendPubMsg(a, m)
Pre:	$\{$Loc(x_1, y_1, a),

$\mathrm{Loc}(x_2, y_2, a_1)$,
$\mathrm{SoundIntensity}(a, s)$,
$\mathrm{Hearing}(a_1, h)$,
$\forall (x_i, y_i) \in \mathrm{dsl}(x_1, y_1, x_2, y_2) : \neg\mathrm{Interference}(x_i, y_i)$,
$\mathrm{euklid}(x_1, y_1, x_2, y_2) \leq s + h$,
$\neg\mathrm{MsgSightReq}\}$

Prob: 1

Post: $\{\mathrm{PubMsg}(a, a_1, m)\}$

Gilt hingegen MsgSightReq, soll also der Absender einer empfangbaren Nachricht nur dann bekannt sein, wenn eine Sichtverbindung zwischen Absender und Empfänger besteht, so bedarf es zunächst einer Zustandsüberführungsregel **CreatePubMsgVisi** zur Erzeugung der öffentlichen Nachrichten mit sichtbarem Empfänger. Diese ist wie folgt gegeben:

CreatePubMsgVisi — $\mathrm{SendPubMsg}(a, m)$

Pre: $\{\mathrm{Loc}(x_1, y_1, a)$,
$\mathrm{Loc}(x_2, y_2, a_1)$,
$\mathrm{SoundIntensity}(a, s)$,
$\mathrm{Hearing}(a_1, h)$,
$\forall (x_i, y_i) \in \mathrm{dsl}(x_1, y_1, x_2, y_2) : \neg\mathrm{Interference}(x_i, y_i)$,
$\mathrm{euklid}(x_1, y_1, x_2, y_2) \leq s + h$,
$\forall (x_i, y_i) \in \mathrm{dsl}(x_2, y_2, x_1, y_1) \setminus \{(x_2, y_2)\} :$
$\neg\mathrm{Wall}(x_i, y_i) \wedge \neg\mathrm{Curtain}(x_i, y_i) \wedge \neg\mathrm{Fog}(x_i, y_i)$,
$\neg(a = a_1)$,
$\mathrm{MsgSightReq}\}$

Prob: 1

Post: $\{\mathrm{PubMsg}(a, a_1, m)\}$

Gilt MsgSightReq, werden die empfangbaren Nachrichten mit unbekanntem Absender für die Fälle, in denen keine Sichtverbindung zwischen Absender und Empfänger besteht, von der folgenden Regel **CreatePubMsgInvisi** erzeugt:

CreatePubMsgInvisi — $\mathrm{SendPubMsg}(a, m)$

Pre: $\{\mathrm{Loc}(x_1, y_1, a)$,
$\mathrm{Loc}(x_2, y_2, a_1)$,
$\mathrm{SoundIntensity}(a, s)$,
$\mathrm{Hearing}(a_1, h)$,
$\forall (x_i, y_i) \in \mathrm{dsl}(x_1, y_1, x_2, y_2) : \neg\mathrm{Interference}(x_i, y_i)$,
$\mathrm{euklid}(x_1, y_1, x_2, y_2) \leq s + h$,

$\exists (x_i, y_i) \in \text{dsl}(x_1, y_1, x_2, y_2) \backslash \{(x_2, y_2)\}:$
$\quad \text{Wall}(x_i, y_i) \vee \text{Curtain}(x_i, y_i) \vee 7\text{Fog}(x_i, y_i),$
$\neg(a = a_1),$
$\text{MsgSightReq}\}$

Prob: 1

Post: $\{\text{AnonPubMsg}(a_1, m)\}$

Beispiel 16: Abbildung 18 zeigt eine Situation, in der ein Agent a in Zelle $(7, 0)$ mit einer Sendestärke von 12 eine öffentliche Nachricht versenden möchte, die von einem Agenten a_1 in Zelle $(13, 13)$ und einer Empfangsempfindlichkeit von 4 empfangen werden soll. Gelte $\text{MsgSightReq} \notin S_t$, so dass die Sichtbarkeit zwischen den Agenten keine Berücksichtigung finden muss. Es gibt zwar Zellen, in denen sich die Sendereichweite von a und die Empfangsreichweite von a_1 überlappen, da Kommunikation jedoch nicht reflektierend ist, stört das kommunikationsblockierende Hindernis in Zelle $(12, 10)$ die Kommunikation. Agent a_1 nimmt die Nachricht von Agent a nicht wahr.

In Abbildung 19 befindet sich Agent a_1 eine Zelle weiter westlich. Zudem soll nun $\text{MsgSightReq} \in S_t$ gelten. Sei angenommen, der Agent a_1 habe eine Sichtweite von 12. Da sich auf der diskreten Linie zwischen a und a_1 kein kommunikationsblockierendes Hindernis mehr befindet und da sich Sendereichweite von a und Empfangsreichweite von a_1 überlappen, kann a_1 die öffentliche Nachricht von a empfangen. Allerdings erhält a_1 die öffentliche Nachricht ohne Absenderangabe als Nachricht vom Typ $\text{AnonPubMsg}(a_1, m)$, da sich a nicht im Sichtfeld von a_1 befindet.

6.6 wahrnehmung von nachrichten

Der in Kapitel 5.4 bestimmten Beobachtungsmenge $O^8_{S_t,a}$ sollen zur Bestimmung der endgültigen Beobachtungsmenge $O_{S_t,a}$ die für den Agenten a empfangbaren Nachrichten hinzugefügt werden. Dies ist vergleichweise einfach, da sich die für einen Agenten a empfangbaren Nachrichten direkt aus dem Umgebungszustand ergeben.

Es gilt:

$$O_{S_t,a} = O^8_{S_t,a} \cup \{\text{PrivMsg}(z, a, m) \in S_t\}$$
$$\cup \{\text{PubMsg}(z, a, m) \in S_t\} \cup \{\text{AnonPubMsg}(a, m) \in S_t\}$$

6.7 entfernung alter nachrichten

Wie zu Kapitelanfang beschrieben soll jede Nachricht nur zu einem Zeitpunkt t anliegen, d. h. eine im Zustand S_t enthaltene

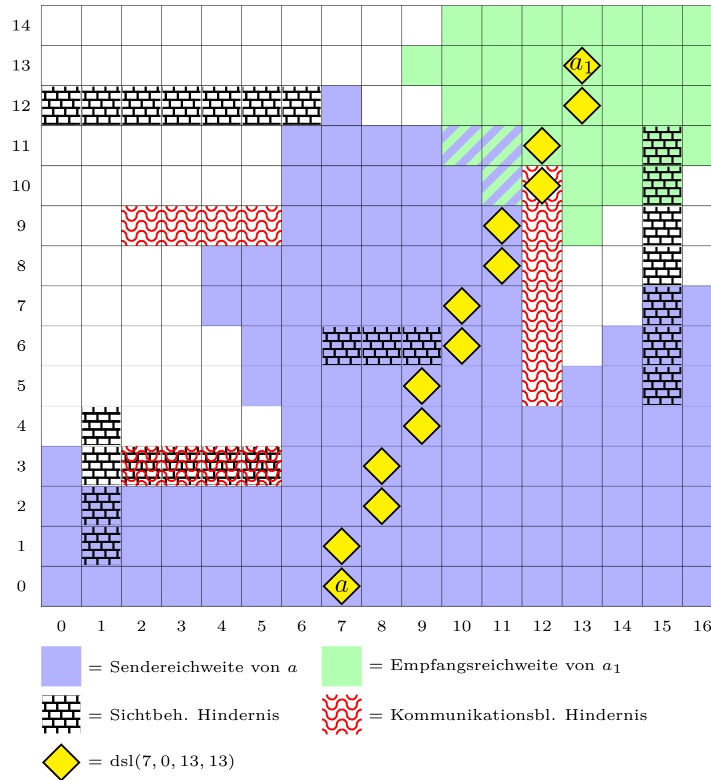

Abbildung 18: Beispiel für erfolglose Kommunikation

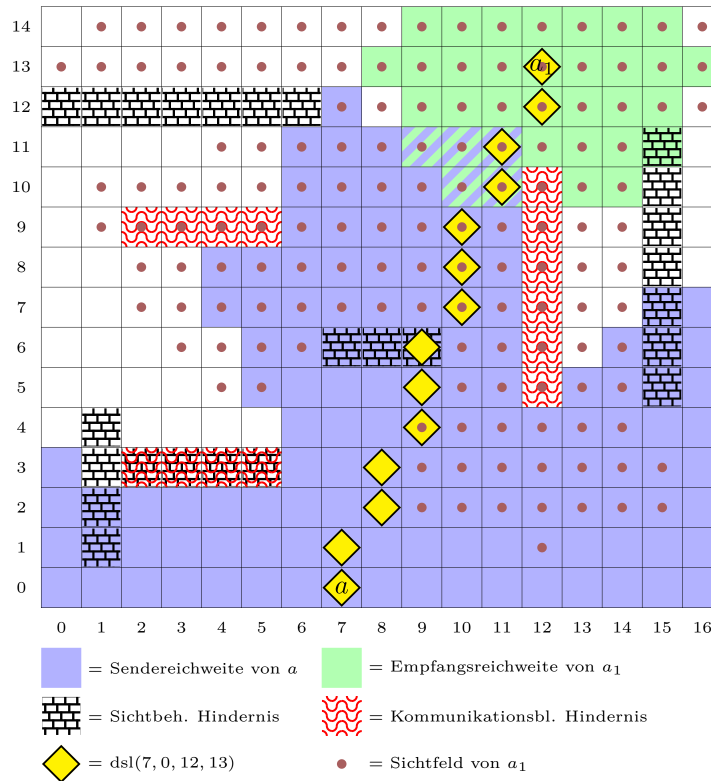

Abbildung 19: Beispiel für erfolgreiche Kommunikation (mit Sichtfeld)

Nachricht soll im Zustand S_{t+1} nicht mehr enthalten sein. Um das Entfernen alter Nachrichten durch eine Zustandsüberführungsregel zu veranlassen, bedarf es eines immer anliegenden internen Einflusses, der diese Regel auslöst. Dieser sei durch die Einflusserzeugungsregel **Nature** gegeben. Als Vorbedingung dient dabei S_t, damit dieser Einfluss für jeden Zeitpunkt t nur genau einmal erzeugt wird.

Die Einflusserzeugungsregel ist wie folgt gegeben:

Nature	
Pre:	$\top$
Prob:	1
Post:	NatureInfluence

Der von dieser Regel erzeugte Einfluss löst die folgenden Zustandsüberführungsregeln aus, welche sämtliche Nachrichten entfernen. Zur Entfernung von privaten Nachrichten dient dabei die folgende Regel:

RemoveOldPrivMsg		NatureInfluence
Pre:	$\{\text{PrivMsg}(a, a_1, m)\}$	
Prob:	1	
Post:	$\{\neg\text{PrivMsg}(a, a_1, m)\}$	

Analog dazu die Regel zur Entfernung öffentlicher Nachrichten mit bekanntem Absender:

RemoveOldPubMsg		NatureInfluence
Pre:	$\{\text{PubMsg}(a, a_1, m)\}$	
Prob:	1	
Post:	$\{\neg\text{PubMsg}(a, a_1, m)\}$	

Und die Zustandsüberführungsregel zur Entfernung öffentlicher Nachrichten mit unbekanntem Absender:

RemoveOldAnonPubMsg		NatureInfluence
Pre:	$\{\text{AnonPubMsg}(a_1, m)\}$	
Prob:	1	
Post:	$\{\neg\text{AnonPubMsg}(a_1, m)\}$	

Damit nicht Nachrichten, deren Erzeugung im Zustand S_t angefordert wurde, bei der Zustandsüberführung vor Erreichen von S_{t+1} von diesen Regeln entfernt werden, soll für die totale

Ordnung π, welche die Verarbeitungsreihenfolge der Einflüsse bei der Zustandsüberführung festlegt, gelten:

$$\forall l \in I_t \setminus \{\text{Nature}\} : \text{Nature} <_\pi l$$

Dadurch wird sichergestellt, dass alte Nachrichten vor der Erzeugung neuer Nachrichten entfernt werden.

7 IMPLEMENTIERUNG

Die Implementierung der in den Kapiteln 4, 5 und 6 deklarativ vorgestellen Funktionalität erfolgt objektorientiert in der Programmiersprache Java[1]. Gridwelten werden dabei von einer im Folgenden als GridWorldSim-Server bezeichneten Komponente den Agenten bereitgestellt, indem der Server über eine TCP-Verbindung den Agenten Wahrnehmungen in einer XML-Sprache mitteilt und von diesen Aktionsanforderungen über die gleiche TCP-Verbindung in Form einer anderen XML-Sprache entgegennimmt. Der GridWorldSim-Server liest dabei seine Konfiguration einschließlich des initialen Umgebungszustands aus einer Spezifikationsdatei. Bevor Agenten an einer Umgebung teilnehmen, müssen sich diese mit einer Anmeldekennung und ggf. mit einem Passwort beim Server identifizieren.

Zur Demonstration und zum Testen der Funktionalität des GridWorldSim-Servers existiert ein grafischer Agenten-Client, mit dem ein Agent vom Benutzer manuell kontrolliert werden kann. Ferner existiert ein grafischer Beobachter-Client, der nach der Anmeldung beim Server die von diesem bereitgestellte Gridwelt vollständig anzeigen und manuell Zustandsüberführungen anstoßen kann. Der Beobachter-Client nutzt dazu die gleichen XML-Sprachen, die auch von Agenten zur Kommunikation mit dem Server verwendet werden.

Zur einfachen Verwendung des Servers mit Agenten bzw. Clients, die ebenfalls in Java implementiert sind, wird darüber hinaus eine Server-API genannte Klassenbibliothek zur Verfügung gestellt, welche die zur Kommunikation mit dem Server notwendigen Protokolle implementiert, Gridwelt-Wahrnehmungen in Form geeigneter Datenstrukturen anbietet und eine Klasse zum komfortablen Absenden von Aktionsanforderungen beinhaltet. Die Server-API wird sowohl vom Agenten- als auch vom Beobachter-Client verwendet.

7.1 INHALT DER CD-ROM

Die Implementierung der GridWorldSim-Komponenten ist auf der beiliegenden CD-ROM enthalten. Auf dieser findet sich die in Spezifikation 6 angegebene Verzeichnisstruktur.

Die ausführbaren JAR-Dateien im Verzeichnis `/build/` versuchen automatisch die notwendigen Bibliotheken im Unterver-

1 Java SE 6

Pfad	Beschreibung
`/build/`	GridWorldSim-Server, Agenten-Client und Beobachter-Client in Form mittels „`java -jar dateiname.jar`“ ausführbarer JAR-Dateien sowie eine Beispielumgebung (`server.xml`)
`/build/lib/`	zur Ausführung notwendige Klassenbibliotheken
`/javadoc/`	Javadoc-Dokumentation aller GridWorldSim-Komponenten
`/os_x/`	Mac OS X-Application-Bundles des Agenten- und Beobachter-Clients
`/schemas/`	XML-Schemas der verwendeten XML-Sprachen
`/src/`	Java-Quellcode aller GridWorldSim-Komponenten

Spezifikation 6: Verzeichnisstruktur der beiliegenden CD-ROM

Komponente	Beschreibung
`agentclient`	Agenten-Client, abhängig von: `commons`, `clientcommons`, `log4j`, `serverapi`
`clientcommons`	gemeinsame Klassen des Agenten- und Beobachter-Clients, abhängig von: `commons`, `log4j`, `serverapi`
`commons`	gemeinsame Klassen aller GridWorldSim-Komponenten, abhängig von: `log4j`
`observerclient`	Beobachter-Client, abhängig von: `commons`, `clientcommons`, `log4j`, `serverapi`
`server`	GridWorldSim-Server, abhängig von: `commons`, `log4j`
`serverapi`	Bibliothek zur Kommunikation mit dem Server und zur Bereitstellung von Datenstrukturen für Wahrnehmungen, abhängig von: `commons`, `log4j`

Spezifikation 7: GridWorldSim-Komponenten und ihre Abhängigkeiten im Überblick

zeichnis `lib/` zu laden, schlägt dies fehl, so müssen diese dem Klassenpfad hinzugefügt werden. Im Verzeichnis `/src/` ist der Quellcode gemäß den zwischen den Paketen vorliegenden Abhängigkeiten in sechs verschiedene Komponenten gegliedert. Spezifikation 7 beschreibt die Komponenten und ihre Abhängigkeiten sowohl in Hinblick auf die JAR-Dateien als auch in Hinblick auf den Quellcode. Bei `log4j` handelt es sich um „Apache log4j 1.2“, eine Bibliothek, die der Ausgabe von Statusinformationen dient und die nicht im Rahmen von GridWorldSim entwickelt wurde. Die Bibliothek befindet sich auf der CD-ROM im Verzeichnis `/build/lib/`.

7.2 eigenschaften und einordnung

Die Semantik der Implementierung wird im Allgemeinen durch die Kapitel 4, 5 und 6 festgelegt. Dieser Abschnitt befasst sich mit den implementatorischen Entsprechungen der in diesen Kapiteln vorgestellten Konzepte und erläutert implementationsspezifische Eigenschaften.

7.2.1 *Umgebungszustand*

Der Umgebungszustand (siehe Kapitel 4.2) wird von der Implementierung in Form einer objektorientierten Datenstruktur vorgehalten. Das grundlegende Java-Objekt zur Repräsentation einer Gridwelt entsteht dabei aus der Java-Klasse `GridWorld`. Hinsichtlich des Umgebungszustands beinhaltet ein `GridWorld`-Objekt die Menge der noch unverarbeiteten Nachrichten, MsgSightReq (siehe Kapitel 6.2) und eine Menge von Objekten vom Typ `GridCell`. Ein `GridCell`-Objekt repräsentiert eine Gridzelle (x, y) samt ihrer Kapazität und eventuell in ihr vorhandener Hindernisse und Interferenzen und beinhaltet die Menge aller Agenten und Objekte z für die $\mathrm{Loc}(x, y, z)$ gilt.

Agenten und Objekte werden wiederum durch Java-Objekte vom Typ `GridObject` und `Agent` repräsentiert, welche die Eigenschaften der Agenten und Objekte umfassen einschließlich des aus weiteren Objekten von Typ `GridObject` bestehenden Inventars.

7.2.2 *Integritätsbedingungen*

Die Klasse `GridWorld` bietet darüber hinaus Methoden zur Bewegung und zum Hinzufügen von Objekten und Agenten, wobei diese Methoden die Einhaltung der in Kapitel 4.3 beschriebenen Integritätsbedingungen sicherstellen, indem keine Bewegungen

oder Hinzufüge-Operationen ausgeführt werden, welche die Integritätsbedingungen verletzen würden.

Damit ist auch für den Fall, dass in der Spezifikationsdatei ein Startzustand spezifiziert wird, der die Integritätsbedingungen verletzt, sichergestellt, dass der von der Implementierung tatsächlich erzeugte Startzustand die Integritätsbedingungen erfüllt. Die Teile der Spezifikation, die zur Verletzung der Integritätsbedingungen führen, werden dazu bei der Erzeugung des Startzustands ignoriert. Ferner kann bei der Implementierung von Zustandsüberführungsregeln auf eine Überprüfung, ob die Integritätsbedingungen eingehalten werden, verzichtet werden, da die Datenstruktur die Überprüfung selbst vornimmt und verletzende Operationen nicht ausführt.

7.2.3 *Zustandsüberführungsregeln*

Die in den Kapiteln 4.7.2, 4.11.1 und 4.11.2 beschriebenen Zustandsüberführungsregeln finden ihre implementatorische Entsprechung in den Klassen des Pakets `net.tittel.gridworldsim.server.statetrans`. Benutzereigene Regeln sind durch Implementierung eigener Regelklassen möglich, welche das Interface `StateTransRule` implementieren müssen. Eine Regel kann über in der Spezifikationsdatei konfigurierbare Parameter verfügen, bei denen es sich um ein Tupel aus Parametername und -wert handelt. Welche Parameter unterstützt werden, hängt dabei von der jeweiligen Regelklasse ab.

Für die Regelklassen von GridWorldSim gilt in Bezug auf die Regeln der Kapitel 4.7.2, 4.11.1 und 4.11.2 und ihre Parameter:

- `MoveAgentRule` entspricht der Regel AgentMove. Die Regel besitzt den Parameter `diagcorrect` mit den möglichen Werten `true` und `false`, welcher angibt, ob die Diagonalkorrektur aktiviert ist. Wird der Parameter nicht gesetzt, so gilt `false`.

- `TakeObjectRule` entspricht der Regel TakeObjectFromGrid.

- `ReleaseObjectRule` entspricht der Regel ReleaseObjectToGrid.

- `MoveObjectRule` entspricht den Regeln MoveObjectOutOfCurrentCell und MoveObjectFromNeighborCell. Analog zu `MoveAgentRule` verfügt auch diese Regel über den Parameter `diagcorrect`.

- `CreateAcceptRule` entspricht der Regel CreateAccept.

- `RemoveAcceptRule` entspricht der Regel RemoveAccept.

- `TransferObjectRule` entspricht den Regeln TransferObjectCurrentCell und TransferObjectNeighborCell.
- `LoadObjectRule` entspricht den Regeln LoadObjectFromGrid und LoadObjectFromInventory.
- `UnloadObjectRule` entspricht den Regeln UnloadObjectToGrid und UnloadObjectToInventory.
- `FogRule` entspricht den Regeln CreateFog und UncreateFog. Für diese Regel sind folgende Parameter möglich:
  - `createProb` gibt an, mit welcher Wahrscheinlichkeit Nebel pro Zustandsüberführungsschritt in einer Zelle erzeugt werden soll. Möglich sind Werte zwischen 0 und 1.
  - `removeProb` gibt analog dazu an, mit welcher Wahrscheinlichkeit Nebel entfernt werden soll.
  - `first` gibt an, ob zuerst Nebel erzeugt oder zuerst Nebel entfernt oder ob die Reihenfolge bei jeder Zustandsüberführung zufällig festgelegt werden soll (vgl. Kapitel 4.11.1). Mögliche Werte sind: `create`, `remove` und `random`.
- `LockUnlockRule` entspricht den beiden Regeln LockDepositBox und UnlockDepositBox.

7.2.4 *Wahrnehmung*

Die Wahrnehmung, einschließlich der Wahrnehmung von Nachrichten, wird für Beobachterwerkzeuge und für jeden Agenten durch die Klasse `OutputPerceptionFactory` berechnet. Diese Klasse wird einmalig für Beobachterwerkzeuge sowie für jeden Agenten instanziiert und greift auf die in Kapitel 7.2.1 skizzierte Datenstruktur zurück, um die Menge der aktuellen Beobachtungen gemäß Kapitel 5 und 6.6 zu berechnen.

Die Verwendung eigener Klassen zur Berechnung von Beobachtungen ist nicht explizit vorgesehen. Da die Beobachtungsbestimmung vollständig in einer eigenen Klasse gekapselt ist, kann bei Vorliegen des Quellcodes diese dennoch einfach modifiziert oder ausgetauscht werden.

7.2.5 *Kommunikation*

Kommunikation ist in der Implementierung nicht als Regel, sondern als Teil der Hauptklasse `Server` implementiert. Der Inhalt einer Nachricht kann beliebig sein, sofern er für den `CDATA`-Abschnitt einer XML-Datei gültig ist. Der Nachrichteninhalt wird

in diesem Fall vom Server unmodifiziert an die Empfänger weitergeleitet.

Die Server-API unterstützt den Versand von beliebigen Java-Objekten und verhält sich dabei wie folgt:

- Ist der zu versendende Nachrichteninhalt ein Java-Objekt vom Typ `String`, so wird der `String` unmodifiziert versendet.

- Handelt es sich beim zu versendenden Nachrichteninhalt nicht um einen `String`, so muss das Objekt serialisierbar sein. In diesem Falle wird es serialisiert und Base64-kodiert versendet.

- Im Falle eingehender Nachrichten wird versucht, diese mit Base64 zu dekodieren und anschließend zu deserialisieren. Gelingt dies, so ist der wahrgenommene Nachrichteninhalt vom Typ des deserialisierten Java-Objekts, anderenfalls der unmodifizierte Nachrichteninhalt vom Typ `String`.

Das Versenden unmodifizierter Strings ist vor allem dann notwendig, wenn nicht in Java implementierte Agenten Teil der Gridwelt sind und somit serialisierte Java-Objekte kein geeignetes Austauschformat für Nachrichten darstellen. In diesem Fall muss der Agent die Kodierung und Dekodierung selbst vornehmen, um sie dann der Server-API als `String` zu übergeben.

Sind hingegehen alle Agenten der Umgebung in Java implementiert, so bietet die Server-API eine komfortable Möglichkeit, beliebige serialisierbare Java-Objekte zu verschicken.

7.2.6 *Ausführungsreihenfolge*

Gemäß Kapitel 4.6 hängt die Ausführungsreihenfolge bei der Zustandsüberführung von zwei totalen Ordnungen π und ρ ab. Für die Implementierung gilt, dass jedem Agenten eine Priorität zugewiesen werden kann. Die Aktionsanforderung eines höher priorisierten Agenten wird dabei vor der Aktionsanforderung eines niedriger priorisierten Agenten behandelt. Ist zwei Agenten die gleiche Priorität zugewiesen, dann ist unbestimmt, wessen Aktionsanforderung zuerst behandelt wird. Ist eine totale Ordnung wie in Kapitel 4.6 gewünscht, müssen Agenten daher eindeutige Prioritäten zugewiesen werden. Die totale Ordnung π ergibt sich damit aus den Prioritäten der Agenten, wobei der immer geltende interne Einfluss Vorrang vor Aktionsanforderungen besitzt.

Für die Ordnung ρ gilt, dass diese in der Implementierung vom vorliegenden Einfluss unabhängig ist. Für jeden Einfluss werden

die Regeln also immer in der gleichen Reihenfolge angewendet. Die totale Ordnung ρ ergibt sich aus der Reihenfolge, mit der die Regeln in der Spezifikationsdatei spezifiziert sind, wobei eine näher am Beginn der Datei spezifizierte Regel Vorrang vor einer vom Beginn weiter entfernten Regel besitzt.

7.2.7 *Hinzufügen und Entfernen von Agenten und Objekten*

In Gridwelten gemäß Kapitel 4 ergibt sich ein Zustand S_{t+1} aus einem Zustand S_t durch die Anwendung von Regeln auf Einflüsse, wobei es sich bei einem Einfluss entweder um eine Aktionsanforderung oder um einen internen Einfluss handeln kann. Dabei wird das Vorhandensein aller Agenten und Objekte, die Teil der Gridwelt sind, im Zustand S_0 vorausgesetzt. Ebenso können Agenten und Objekte, die Teil der Gridwelt sind, nicht ihre Existenz innerhalb der Gridwelt beenden, da es keine Regel gibt, die dies erlauben würde. Dieses Vorgehen ist konzeptionell darin begründet, dass eine Gridwelt in sich geschlossen sein und nicht durch Faktoren, die außerhalb der Gridwelt liegen, manipuliert werden soll. Insbesondere sollen praktische Ereignisse der realen Welt wie z. B. „Agent verbindet sich mit dem Server“ oder „Agent verliert Verbindung zum Server“ nicht Teil der Semantik von Gridwelten sein, da in der Gridwelt Konzepte wie „Verbindung“ und „Server“ gar nicht existieren.

Aus praktisch-implementatorischer Sicht betrachtet ist es jedoch wünschenswert, wenn nicht alle Agenten bereits Teil des Startzustandes sein müssen, damit Agenten, die aus realweltlichen Gründen noch nicht zum Startzeitpunkt der Simulation anwesend sein können, von einer späteren Teilnahme nicht grundlegend ausgeschlossen sind. Noch dringlicher ist jedoch die Entfernung von Agenten, da sich in der realen Welt ungewollte Verbindungsabbrüche nicht immer vermeiden lassen. Für den Fall, dass die Umgebung in einen neuen Umgebungszustand nur dann überführt wird, wenn von jedem Agenten eine Aktionsanforderung vorliegt, würde ohne Möglichkeit zur Entfernung von Agenten keine Zustandsüberführung in der Umgebung mehr stattfinden, wenn ein einzelner Agent die Verbindung zum Server verlieren würde.

Ist das Hinzufügen und Entfernen von Agenten möglich, so hat dies auch Auswirkungen auf die Menge der in der Gridwelt existierenden Objekte, da Agenten bereits zum Zeitpunkt ihres Betretens der Gridwelt über Objekte in ihrem Inventar verfügen können. Ebenso stellt sich die Frage, was mit Objekten im Inventar eines Agenten geschehen soll, wenn der Agent die Gridwelt verlässt. Dabei sind zwei Möglichkeiten denkbar:

- Die Objekte im Inventar eines Agenten, der die Gridwelt verlässt, werden in der Gridzelle abgelegt, in der sich der Agent zum Zeitpunkt seines Verlassens der Gridwelt befand.
- Die Objekte im Inventar werden zusammen mit dem Agenten aus der Gridwelt entfernt.

Beide Möglichkeiten sind problembehaftet: Im ersten Fall könnten Agenten, die zum Zeitpunkt des Betretens der Gridwelt über Objekte in ihrem Inventar verfügen, durch wiederholten Auf- und Abbau der Serververbindung Objekte beliebig vermehren. Im zweiten Fall könnten Agenten durch wiederholten Auf- und Abbau der Serververbindung Objekte aus der Gridwelt entfernen.

Für die Implementierung gelten die folgenden Vereinbarungen:

- Für die intendierte Abmeldung vom Server gilt für Agenten und Beobachter-Clients, dass diese lediglich ihre TCP-Verbindung schließen. Der Server erkennt daraufhin das Schließen der Verbindung und nimmt die Deregistrierung des Agenten oder Beobachter-Clients automatisch vor.
- Trennt oder verliert ein Agent die Verbindung zum Server, so wird der Agent bei der nächsten Zustandsüberführung vor Anwendung der Zustandsüberführungsregeln aus der Umgebung samt der in seinem Inventar enthaltenen Objekte entfernt. Das Entfernen der Objekte erscheint dabei als das geringere Übel. Möchte ein Agent Objekte dem Zugriff anderer Agenten entziehen, könnte er dies auch ohne Verbindungsabbau erreichen, indem er die Objekte in seinem Inventar behält. Schließlich gibt es keine Möglichkeit für andere Agenten, Objekte aus dem Inventar eines Agenten ohne dessen Einwilligung zu entfernen.

  Für den Fall, dass im gewählten Ausführungsmodus eine Zustandsüberführung durch jede Aktionsanforderung eines Agenten ausgelöst wird, erzeugt auch der Verbindungsabbruch eines Agenten einen neuen Zustand.
- Baut ein Agent eine Verbindung zum Server auf, so wird der Agent einschließlich des für ihn vorgegebenen Inventars bei der nächsten Zustandsüberführung vor Anwendung der Zustandsüberführungsregeln, aber nach Entfernung eventuell zu entfernender Agenten der Umgebung hinzugefügt, wenn die Integritätsbedingungen dies erlauben. Soll sichergestellt sein, dass die Agenten, die bereits Teil der Umgebung sind, das Hinzufügen eines Agenten nicht blockieren können, empfiehlt es sich, als Startposition von Agenten eine Zelle mit unbegrenzter Kapazität zu wählen.

7.2.8 *Parameter und Properties*

In Kapitel 4.2 werden benutzerspezifizierte Fakten diskutiert, mit denen ein Agent oder Objekt beobachtbare Eigenschaften besitzen kann, die für die Zustandsüberführungssemantik jedoch keine Rolle spielen und dazu dienen, dass Agenten über eine Grundlage zur Bewertung von Objekten und anderer Agenten verfügen. Ein Agent kann also z. B. die Eigenschaft *attraktiv* besitzen, die von anderen Agenten z. B. im Rahmen einer sozialen Simulation als Bewertungsgrundlage für das Verhalten gegenüber diesem Agenten herangezogen wird.

Unabhängig von benutzerspezifizierten Fakten besitzt bspw. ein Objekt o vom Typ „Schließfach" immer die beobachtbare Eigenschaft SafeDepositBox(o) (siehe Kapitel 4.11.2). Solche objektinhärenten Eigenschaften sind im Regelfall auch bei Objekttypen erwünscht, mit welchen der Benutzer die Semantik von GridWorldSim erweitert. Möchte ein Benutzer z. B. GridWorldSim um ein Objekt vom Typ „Fitnessgerät" ergänzen, welches es Agenten ermöglicht, ihre Kapazität bei jeder Benutzung um k Kapazitätseinheiten zu erhöhen, so wird er in der Regel wünschen, dass Agenten Objekte vom Typ „Fitnessgerät" auch als solche wahrnehmen können. Zudem wäre es vermutlich wünschenswert, den Wert von k für unterschiedene Objekte vom Typ „Fitnessgerät" in der Spezifikationsdatei spezifizieren zu können.

GridWorldSim geht dabei wie folgt vor:

- Jeder Agent und jedes Objekt kann eine Menge von *Properties* besitzen. Properties können dabei für Agenten beobachtbar oder nicht beobachtbar sein, wobei ein Agent seine eigenen Properties immer vollständig wahrnehmen kann. Im Falle eines Schließfachs ist zum Beispiel die Property, die das Objekt als Schließfach ausweist, von Agenten beobachtbar. Das aktuell für das Schließfach gesetzte Passwort ist ebenfalls eine Property, aber nicht von Agenten, sondern nur vom Beobachter-Client wahrnehmbar.

- Properties können entweder in der Spezifikationsdatei festgelegt werden oder aber vom Java-Objekt, welches den Agenten oder das Objekt repräsentiert, selbst erzeugt werden. Ein Schließfachobjekt besitzt immer (d. h. ohne dass dies in der Spezifikationsdatei spezifiziert wäre) eine Property, die es als Schließfach ausweist, und im verschlossenen Zustand immer eine von Agenten nicht beobachtbare Property, die das aktuelle Passwort angibt. Soll ein Schließfach jedoch zusätzlich golden sein, so handelt es sich dabei um ein benutzerspezifiziertes Faktum, welches in der Spezifikationsdatei spezifiziert wird.

- Zur Konfiguration eines benutzerdefinierten Objekts (z. B. Festlegung des Werts k für Objekte vom Typ „Fitnessgerät“) können in der Spezifikationsdatei Tupel aus Parametername und -wert angegeben werden. Die unterstützten Parameternamen sind dabei von der Java-Klasse des Agenten oder Objekts abhängig. Im Fall des Fitnessgeräts könnte zum Beispiel ein Parameter mit dem Namen „improvement“ unterstützt werden, dessen Wert den Wert von k festlegt.

- Das Spezifizieren von Properties in der Spezifikationsdatei erfolgt über das Setzen des Parameters „property“. Mehrere Properties werden dabei durch mehrfaches Setzen des Parameters erreicht. In der Spezifikationsdatei definierte Properties sind von Agenten immer beobachtbar. Nicht von Agenten beobachtbare Properties werden ausschließlich von der Java-Klasse erzeugt, welche den Agenten oder das Objekt realisiert.

7.2.9 *Bereitschaft zum Empfang von Objekten*

Agenten können zusätzlich zur in Kapitel 4.4 beschrieben Möglichkeit, die Bereitschaft zur Aufnahme von Objekten zu erklären, dem Server mitteilen, dass sie bereit sind, ein Objekt von allen Agenten oder alle Objekte von einem Agenten in Empfang zu nehmen. Gleiches gilt für das Widerrufen dieser Bereitschaft.

Diese allgemeineren Erklärungen sind dabei von den spezifischen Erklärungen, genau ein bestimmtes Objekt von einem bestimmten Agenten in Empfang nehmen oder nicht mehr in Empfang nehmen zu wollen, unabhängig. Dies bedeutet, dass eine erklärte Bereitschaft, ein Objekt von allen Agenten oder alle Objekte von einem Agenten annehmen zu wollen, so lange bestehen bleibt, bis diese auf die gleiche Art widerrufen wurde.

7.2.10 *Offene und geschlossene Agententypen*

Es gibt bei GridWorldSim zwei Arten von Agententypen:

- Für einen *offenen Agententypen* kann sich ein Agent direkt mit einer Anmeldekennung und ggf. einem Passwort beim Server anmelden. Dabei gilt, dass sich mehrere Agenten mit den gleichen Zugangsdaten anmelden können. Jeder Agent erhält nach der Anmeldung als Agentennamen den Namen des offenen Agententyps gefolgt von einem Punkt und einer eindeutigen Nummer.

  Die Intention hinter offenen Agententypen besteht darin, dass es damit einfach möglich ist, einer großen Menge

gleichförmiger Agenten die Anmeldung zu ermöglichen, ohne jeden Agenten einzeln spezifizieren zu müssen. Die Verwendung offener Agententypen birgt jedoch die Gefahr, dass sich ein Agent mehrfach bei der Umgebung anmelden kann, um daraus Vorteile zu ziehen. Für Wettbewerbssituationen sind offene Agententypen daher ohne weitere Schutzmaßnahmen nicht geeignet. Ebenso ist es bei diesen Agententypen nicht möglich, dass ein Agent bereits zum Zeitpunkt der Anmeldung über Objekte in seinem Inventar verfügt.

- Bei *geschlossenen Agententypen* ist keine unmittelbare Anmeldung möglich, sondern sie dienen als Grundlage für *individuell spezifizierte Agenten*. Ein solcher Agent kann dabei nur einmal zeitgleich an der Umgebung teilnehmen und darüber hinaus zum Zeitpunkt des Betretens der Umgebung über Objekte in seinem Inventar verfügen.

Die Unterscheidung zwischen diesen Agententypen bezieht sich ausschließlich auf die Art und Weise, wie Agenten im Spezifikationsformat spezifiziert werden und dient der Vereinfachung, da es mit offenen Agententypen möglich ist, zu Testzwecken beliebig viele Agenten an der Umgebung teilnehmen zu lassen, ohne dass diese einzeln spezifiziert werden müssten, zum anderen mit geschlossenen Typen eine Möglichkeit zur Verfügung steht, Wettbewerbssituationen gerecht zu werden. Insbesondere besteht nach der Anmeldung eines Agenten weder aus konzeptioneller noch aus Sicht der Implementierung ein Unterschied zwischen Agenten unterschiedlichen Typs.

7.3 Spezifikationsformat

Das Format zur Spezifikation der GridWorldSim-Serverkonfiguration dient der Erfüllung folgender Aufgaben:

- Festlegung des initialen Umgebungszustandes
- Festlegung der Ausführungsreihenfolge von Aktionsanforderungen und Regeln
- Festlegung, unter welchen Umständen eine Zustandsüberführung vorgenommen wird
- Festlegung, welche Agenten in einem beliebigen Simulationszeitpunkt die Umgebung betreten können und welche Eigenschaften diese Agenten besitzen
- Festlegung technischer Aspekte, wie z. B. die Portnummern, auf welchen der Server auf eingehende Verbindungen war-

tet oder die Festlegung der zulässigen Zugangsdaten für Beobachter-Clients

Im folgenden Abschnitt erfolgt eine Beschreibung des für das Spezifikationsformat verwendeten Sprachtypus. Darauffolgend werden die einzelnen Elemente des Spezifikationsformats im Detail erläutert.

7.3.1 *Sprachtypus*

Das Spezifikationsformat ist eine XML-Sprache, welche durch die Schemasprache *XML Schema* beschrieben wird. Dies bietet zwei wesentliche Vorteile gegenüber einem proprietären Spezifikationsformat: Zum einen existieren für XML-Sprachen Bearbeitungswerkzeuge, welche eine XML-Sprache anhand genereller XML-Spracheigenschaften und anhand des Schemas der Sprache erkennen und damit explizit unterstützen können. Dies ermöglicht die Verwendung (grafischer) XML-Editoren, welche komfortabel vom tatsächlichen Inhalt einer Spezifikationsdatei abstrahieren können und in der Lage sind, die Korrektheit einer Spezifikation bereits zum Erstellungszeitpunkt zu überprüfen. Zum anderen kann die Software, welche die Spezifikation einliest (in diesem Falle also der GridWorldSim-Server) durch Verwendung eines validierenden XML-Parsers die Korrektheit einer Spezifikation überprüfen, ohne dass dazu Implementierungsaufwand in nennenswertem Umfang erforderlich wäre. Insbesondere entfällt dadurch eine umfangreiche Implementierung der Verifikation der Eingabe, die ansonsten aus einer größeren Menge von Gültigkeitsabfragen bestehen würde. Stattdessen kann angenommen werden, dass die Spezifikation hinsichtlich der mittels *XML Schema* vorgenommenen Gültigkeitseinschränkungen korrekt ist. Davon unberührt bleiben Einschränkungen, die nicht im Schema spezifiziert sind, wie z. B. der Umstand, dass die Summe der Kapazitätsbedarfe aller Objekte und Agenten in einer Zelle die freie Kapazität der Zelle nicht überschreiten darf.

Der GridWorldSim-Server akzeptiert nur Spezifikationsdateien, die gegen das Schema validieren. Es empfiehlt sich daher zur Bearbeitung von Spezifikationsdateien die Verwendung eines XML-Editors, welcher die Korrektheit der Spezifikation bereits bei der Erstellung überprüft.

7.3.2 *Beschreibung*

Im Spezifikationsformat ist die Reihenfolge der Elemente in vielen Fällen vorgegeben. Die Beschreibung erfolgt von oben nach unten gemäß dieser Reihenfolge. In den Fällen, in denen die

```
<gridworld xsi:noNamespaceSchemaLocation="../schemas
 /server.xsd" xmlns:xsi="http://www.w3.org/2001
 /XMLSchema-instance" xDimension="17" yDimension="15">
```

Spezifikation 8: Wurzelelement im Beispiel

Reihenfolge von Elementen beliebig ist, erfolgt die Beschreibung in der Reihenfolge, mit der die Elemente im Schema angegeben sind. Sofern nicht anders angegeben, müssen alle Zahlenwerte aus dem Bereich 0 bis $2^{31} - 1$ stammen und kein String darf leer sein. Das Schema, welches das Spezifikationsformat beschreibt, befindet sich auf der CD-ROM unter `/schemas/server.xsd`.

7.3.2.1 *Wurzelelement*

Das Wurzelelement einer GridWorldSim-Serverspezifikation heißt `<gridworldsim>`. Dessen Attribute `xDimension` und `yDimension` geben die Dimension der Gridwelt an und sind zwingend erforderlich. Optional steht das boolsche Attribut `msgsightreq` zur Verfügung, welches angibt, ob ein Agent den Absender einer von ihm empfangenen öffentlichen Nachricht wahrnehmen können muss, um den Absender der Nachricht zu erkennen. Ist das Attribut wahr, so muss dies der Fall sein, anderenfalls kann der Empfänger den Absender einer empfangenen Nachricht immer erkennen. Wird das Attribut nicht angegeben, so wird es als wahr angenommen.

Darüber hinaus sollte im Wurzelelement die verwendete Schemasprache und der Ort des Schemas wie bei XML-Sprachen üblich angegeben werden. Spezifikation 8 zeigt ein Beispiel für das Wurzelelement einer Gridwelt der Dimension 17 mal 15.

7.3.2.2 *Allgemeine Servereigenschaften und Ausführungsmodus*

Das `<server>`-Element beginnt den Abschnitt, der die grundlegenden Servereigenschaften spezifiziert, und ist genau einmal in der Spezifikationsdatei enthalten. Es verfügt über zwei notwendige Attribute:

- `agentPort`: Gibt an, auf welchem TCP-Port der Server auf Verbindungen von Agenten wartet.

- `observerPort`: Gibt an, auf welchem TCP-Port der Server auf Verbindungen von Beobachter-Clients wartet.

Jedes `<server>`-Element muss darüber hinaus ein `<execmode>`-Element beinhalten sowie optional ein `<debug>`-Element (die Reihenfolge ist beliebig). Es gilt dabei:

- Das `execmode`-Element beinhaltet genau eines der folgenden Elemente:

    - `<ondemand>`: Jede Aktionsanforderung und jede Anmeldung eines neuen Agenten löst die Überführung in einen neuen Umgebungszustand aus.
    - `<waitforall>`: Eine Zustandsüberführung wird dann durchgeführt, wenn von jedem Agenten eine Aktionsanforderung vorliegt. Mit dem optionalen Attribut `timeout` (Wertebereich 1 bis $2^{31} - 1$) kann zusätzlich ein Timeout in Millisekunden spezifiziert werden, bei dessen Erreichen auch dann eine Überführung stattfindet, wenn noch nicht von allen Agenten eine Aktionsanforderung vorliegt.
    - `<fixed>`: Die Zustandsüberführung findet nach einer festen Zeitspanne zwischen 1 und $2^{31} - 1$ Millisekunden statt, welche über das notwendige Attribut `interval` angegeben wird.
    - `<manual>`: Die Zustandsüberführung findet nur dann statt, wenn sie durch einen Beobachter-Client explizit ausgelöst wurde.

- Das Element `debug` hat keine Nachfahren und besitzt die folgenden optionalen Attribute:
    - `level`: Gibt an, wie ausführlich der Server Statusinformationen ausgeben soll. Mögliche Werte sind `debug` (ausführliche Ausgaben) und `info` (weniger ausführliche Ausgaben). Wird das Attribut nicht angegeben, wird `info` angenommen.
    - `path`: Gibt den Dateisystempfad an, in welchen Debug-Ausgaben geschrieben werden sollen.
    - `data`: Dieses boolsche Attribut gibt an, ob alle vom Server versandten und empfangenen XML-Dokumente in Dateien gespeichert werden sollen. Damit diese Option wirksam wird, muss das Attribut `path` gesetzt sein. Wird das Attribut nicht angegeben, so gilt es als `false`.
    - `filelog`: Dieses boolsche Attribut gibt an, ob die Statusausgaben des Servers in einer Datei gespeichert werden sollen. Auch für diese Option muss das Attribut `path` gesetzt sein. Unabhängig vom Wert dieses Attributs werden Statusausgaben immer über die Standardausgabe ausgegeben. Wird das Attribut nicht angegeben, so gilt es als `false`.

Spezifikation 9 gibt ein Beispiel für einen Server an, welcher auf Port 7777 auf Agentenverbindungen und auf Port 8888 auf Verbindungen von Beobachter-Clients wartet und welcher

```
<server agentPort="7777" observerPort="8888">
  <execmode>
    <waitforall timeout="1000"/>
  </execmode>
  <debug level="info" data="false" filelog="true"
   path="/var/log/gridworldsim/"/>
</server>
```

Spezifikation 9: `<server>`-Element und Nachfahren im Beispiel

in einen neuen Zustand genau dann überführt, wenn entweder eine Aktionsanforderung von jedem Agent vorliegt oder wenn eine Sekunde seit der letzten Zustandsüberführung vergangen ist. Statusausgaben werden in niedriger Ausführlichkeit in `/var/log/gridworldsim/` gespeichert und eine Speicherung der vom Server gesendeten und empfangenen XML-Dokumente findet nicht statt.

7.3.2.3 *Regeln*

Im `<rules>`-Element werden die Zustandsüberführungsregeln angegeben, welche bei Zustandsüberführungen Verwendung finden sollen. Das `<rules>`-Element kommt genau einmal vor, besitzt keine Attribute und beinhaltet beliebig viele `<rule>`-Elemente, mindestens jedoch eins. Jedes `<rule>`-Element entspricht dabei genau einer Regel,

Ein `<rule>`-Element besitzt genau ein Attribut: `class`. Dieses Attribut ist notwendig und gibt die Java-Klasse an, welche die Regel implementiert. Darüber hinaus können Regeln Parameter besitzen, mit denen ihr Verhalten konfiguriert werden kann (z. B. Aktivierung oder Deaktivierung der Diagonalkorrektur, siehe Kapitel 7.2.3). Zur Festlegung der Parameter kann ein `<rule>`-Element daher über eine beliebige Anzahl an `<parameter>`-Elementen verfügen, welche zwei notwendige Attribute besitzen:

- `name`: Dieses String-Attribut gibt den Namen des Parameters an. Jeder Parametername darf dabei innerhalb einer Regel nur einmal vorkommen.

- `value`: Dieses String-Attribut gibt den Wert des Parameters an.

Für jede Regel gilt, dass jeder Parametername in ihrem `<parameters>`-Abschnitt nur einmalig vorkommen darf.

Ein Beispiel für das `<rules>`-Element und seine Nachfahren gibt Spezifikation 10.

```
<rules>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.FogRule">
    <parameter paraName="createProb" value="0.001"/>
    <parameter paraName="removeProb" value="0.05"/>
    <parameter paraName="first" value="random"/>
  </rule>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.MoveAgentRule">
    <parameter paraName="diagcorrect" value="false"/>
  </rule>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.MoveObjectRule">
    <parameter paraName="diagcorrect" value="false"/>
  </rule>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.TakeObjectRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.ReleaseObjectRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.CreateAcceptRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.RemoveAcceptRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.TransferObjectRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.LoadObjectRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.UnloadObjectRule"/>
  <rule class="net.tittel.gridworldsim.server.
   statetrans.LockUnlockRule"/>
</rules>
```

Spezifikation 10: <rules>-Element und Nachfahren im Beispiel

```
<observers>
  <observer name="bigbrother"/>
  <observer name="obrien" password="fivefingers"/>
</observers>
```

Spezifikation 11: `<observers>`-Element und Nachfahren im Beispiel

7.3.2.4 *Beobachter-Clients*

Das optionale Element `<observers>` besitzt keine Attribute und enthält beliebig viele `<observer>`-Elemente, mindestens jedoch eins. Ein `<observer>`-Element beschreibt Zugangsdaten für die Anmeldung eines Beobachter-Clients und besitzt zwei Attribute:

- `name`: Dieses notwendige String-Attribut gibt die Anmeldekennung für einen Beobachter-Client an.
- `password`: Dieses optionale String-Attribut beinhaltet das zugehörige Passwort.

Wird kein Passwort angegeben, ist dem Beobachter-Client eine Anmeldung nur mit der Anmeldekennung ohne Passwort möglich. Ein Beispiel zum `<observers>`-Element und seinen Nachfahren findet sich in Spezifikation 11.

7.3.2.5 *Agenten*

Spezifikation 12 zeigt die allgemeine Struktur des Abschnitts zur Agentenspezifikation. In eckigen Klammern ist dabei der Wert angegeben, wie oft ein Element vorkommen muss (erster Wert) und wie häufig es höchstens vorkommen darf (zweiter Wert). Ein Stern bedeutet dabei „unendlich oft".

Das `<type>`-Element innerhalb von `<open>` spezifiziert dabei einen offenen und das `<type>`-Element innerhalb von `<closed>` einen geschlossenen Agententyp. Die `<instance>`-Elemente, die unmittelbar von `<instances>` umschlossen werden, definieren einen individuell spezifizierten Agenten und referenzieren dabei die Spezifikation eines geschlossenen Agententyps (siehe Kapitel 7.2.10).

GEMEINSAME EIGENSCHAFTEN Viele Agenteneigenschaften können innerhalb mehrerer Elemente spezifiziert werden. Die Elemente `<types>`, `<type>` (sowohl innerhalb von `<open>` als auch von `<closed>`) und `<instance>` (nur außerhalb von `<contains>`) haben folgende optionale Attribute gemein:

- `freeCap`: Dieses Attribut gibt die initiale freie Kapazität eines Agenten an. Neben 0 bis $2^{31}-1$ ist `unbounded` ein

```
<agents> <!--[1,1]-->
  <types> <!--[1,1]-->
    <open> <!--[0,1]-->
      <type> <!--[1,*]-->
        <parameters> <!--[0,1]-->
          <parameter/> <!--[1,*]-->
        </parameters>
      </type>
    </open>
    <closed> <!--[0,1]-->
      <type> <!--[1,*]-->
        <parameters> <!--[0,1]-->
          <parameter/> <!--[1,*]-->
        </parameters>
      </type>
    </closed>
  </types>
  <instances> <!--[0,1]-->
    <instance> <!--[1,*]-->
      <parameters> <!--[0,1]-->
        <parameter/> <!--[1,*]-->
      </parameters>
      <contains> <!--[0,1]-->
        <instance> <!--[1,*]-->
<!-- beliebig tiefe contains-Schachtelung -->
        <instance>
      </contains>
    </instance>
  </instances>
</agents>
```

Spezifikation 12: Struktur der Agentenspezifikation (ohne Attribute)

möglicher Wert, welcher unbeschränkte Kapazität repräsentiert. Wird an keiner für einen Agenten relevanten Stelle `freeCap` spezifiziert, so gilt `unbounded`.

- `capNeed`: Dieses Attribut gibt den initialen Kapazitätsbedarf an. Wird es an keiner für einen Agenten relevanten Stelle spezifiziert, so gilt 0.
- `moveForce`: Die Verschiebekraft von Agenten wird durch dieses Attribut gesetzt. Auch hier ist `unbounded` möglich und gilt als Vorgabewert.
- `viewRange`: Dieses Attribut besitzt den gleichen Wertebereich wie `freeCap` und `moveForce` und beschreibt den Wert für die Sichtweite von Agenten. Auch hier gilt `unbounded` als Vorgabe, wenn das Attribut an keiner für einen Agenten relevanten Stelle gesetzt ist.
- `hearing`: Mit diesem Attribut kann der Wert für die Empfangsempfindlichkeit von Agenten geändert werden. `unbounded` ist auch hier möglich und gleichzeitig der Vorgabewert, wenn das Attribut an keiner für einen Agenten relevanten Stelle gesetzt ist.
- `soundIntensity`: Analog zu `hearing` beschreibt dieses Attribut den Vorgabewert für die Sendestärke von Agenten.
- `priority`: Gibt die Priorität an, mit der die Aktionsanforderungen von Agenten bearbeitet werden (siehe Kapitel 7.2.6).

Ferner können die Elemente `<type>` und `<instance>` über ein `<parameters>`-Element verfügen, welches beliebig viele, mindestens jedoch ein `<parameter>`-Element beinhaltet. Ein `<parameter>`-Element besitzt die notwendigen String-Attribute `paraName` und `value`. Solche Werte-Tupel werden der Java-Klasse, die den Agenten realisiert, zur beliebigen Verwendung übergeben, wobei die Verwendung von `property` als `paraName` dem Agenten eine Property hinzufügt (siehe Kapitel 7.2.8).

Hinsichtlich der gemeinsamen Agenteneigenschaften gilt für Agenten offenen Typs:

1. Ist ein Wert für sein `<type>`-Element gesetzt, so gilt dieser. Ist dies nicht der Fall, so gilt der entsprechende Wert aus dem `<types>`-Element.
2. Ist der Wert auch für sein `<types>`-Element nicht gesetzt, gilt der allgemeine Vorgabewert.

Für individuell spezifizierte Agenten gilt:

1. Ist ein Wert für sein `<instance>`-Element gesetzt, so gilt dieser. Anderenfalls gilt der Wert für das referenzierte `<type>`-Element unterhalb von `<closed>`.

2. Ist ein Wert auch für sein `<type>`-Element nicht gesetzt, so gilt der entsprechende Wert aus dem `<types>`-Element.

3. Existiert auch kein zugehöriger Wert für das `<types>`-Element, so gilt der allgemeine Vorgabewert.

Für beide Typen von Agenten gilt, dass der Inhalt aller `<parameters>`-Abschnitte der Agentenklasse übergeben wird. Ob ein Parameter gleichen Namens überschrieben wird, wenn er in unterschiedlichen `<parameters>`-Abschnitten einer Agentenspezifikation vorkommt, hängt dabei von der Java-Klasse ab, die den Agenten implementiert. Im Falle von Parametern des Typs `property` findet eine Vereinigung statt, ein Agent verfügt also über die Properties aus allen für ihn relevanten Abschnitten.

AGENTEN OFFENEN TYPS Über die gemeinsamen Attribute hinaus, gibt es für `<type>`-Elemente unterhalb von `<open>` (welche der Spezifikation der Agenten offenen Typs dienen) folgende Attribute:

- `typeName`: Dieses notwendige Attribut gibt den Namen des Typs an. Der Name dient gleichzeitig auch als Anmeldekennung zur Anmeldung von Agenten und als Basis für den vergebenen Agentennamen, er stellt jedoch keine innerhalb der Gridwelt existierende Eigenschaft dar. Darüber hinaus darf es keine zwei `typeName`-Attribute mit gleichem Wert geben.

- `password`: Dieses optionale Argument spezifiziert das zur Anmeldung notwendige Passwort.

- `xPos` und `yPos`: Diese notwendigen Attribute geben an, auf welcher Position des Grids sich ein Agent dieses Typs nach der Anmeldung befindet. Ist die Position nicht gültig (z. B. weil die Zelle eine Mauer oder einen Graben beinhaltet oder weil die Koordinaten außerhalb des Grids liegen), so ist eine Anmeldung des Agenten nicht möglich.

- `class`: Mittels dieses optionalen Attributs kann statt der Standard-Agentenklasse eine eigene Java-Klasse zur Realisierung dieses Agententyps spezifiziert werden.

Ein Beispiel zur Spezifikation eines offenen Agententyps findet sich in Spezifikation 13. Ein Agent könnte sich nun mit der Anmeldekennung „Luna“ ohne Angabe eines Passworts beim Umgebungsserver anmelden und erhielte einen Kapazitätsbedarf

```
<open>
  <type typeName="Luna" xPos="1" yPos="0" capNeed="3">
    <parameters>
      <parameter paraName="property" value="Cat"/>
      <parameter paraName="property" value="White"/>
    </parameters>
  </type>
</open>
```

Spezifikation 13: Offener Agententyp im Beispiel

von 3. Die anderen Eigenschaften wie z. B. freie Kapazität oder Sichtweite würden aus dem umschließenden `<types>`-Element übernommen, sofern sie dort spezifiziert sind. Falls nicht, gelten die allgemeinen Vorgabewerte. Die Eigenschaften „Cat“ und „White“ können von anderen Agenten wahrgenommen werden, haben aber auf die Semantik von GridWorldSim keinen Einfluss.

INDIVIDUELL SPEZIFIZIERTE AGENTEN Unterhalb des Elements `<closed>` besitzen `<type>`-Elemente neben den gemeinsamen Eigenschaften die folgenden Attribute:

- `typeName`: Dieses Attribut gibt den Namen des Typs an, der Name kann jedoch nicht zur Anmeldung eines Agenten bei der Umgebung verwendet werden. Stattdessen wird der Typname im `<instance>`-Element eines individuell spezifizierten Agenten referenziert. Von dieser Referenzierung abgesehen hat `typeName` keine weitere Bedeutung. `typeName`-Attribute müssen auch hier eindeutig sein und ein `typeName`-Attribut innerhalb von `<closed>` darf keinem `typeName`-Attribut innerhalb von `<open>` entsprechen.

- `class`: Dieses Attribut ist optional und gibt wie bei den offenen Agententypen die Java-Klasse an, die zur Realisierung von Agenten dieses Typs dient.

`<instance>`-Elemente, die Kinder von `<instances>` sind, referenzieren einen geschlossenen Agententypen und besitzen die folgenden Attribute neben den gemeinsamen Eigenschaften:

- `agentTypeRef`: Dieses Attribut ist notwendig und sein Wert muss dem `typeName` eines geschlossenen Agententypen entsprechen.

- `name`: Dieses Attribut repräsentiert den Anmeldenamen zur Anmeldung beim GridWorldSim-Server sowie den Namen des Agenten und ist notwendig. Jeder Wert von `name` muss eindeutig sein.

```
<types soundIntensity="10" capNeed="60">
  <closed>
    <type typeName="Singer" soundIntensity="unbounded">
      <parameters>
        <parameter paraName="property" value="Moody"/>
      </parameters>
    </type>
  </closed>
</types>
<instances>
  <instance agentTypeRef="Singer" name="Madonna"
   xPos="0" yPos="0" freeCap="30">
    <contains>
<!--enthaltene Objekte-->
    </contains>
    <parameters>
      <parameter paraName="property" value="Rich"/>
    </parameters>
  </instance>
</instances>
```

Spezifikation 14: Geschlossener Agententyp und individuell spezifizierter Agent im Beispiel

- `xPos` und `yPos`: Die Position, auf welcher der Agent nach der Anmeldung die Gridwelt betritt, wird durch dieses notwendige Attribut angegeben.

Neben `<parameters>` kann ein `<instance>`-Element ein `<contains>`-Element enthalten, welches wiederum `<instance>`-Elemente enthalten kann und der Spezifikation von Objekten dient, die der Agent zum Zeitpunkt des Betretens der Gridwelt in seinem Inventar mit sich führt. Diese `<instance>`-Elemente sind jedoch von einem anderen Typ als die `<instance>`-Elemente direkt unterhalb von `<instances>`, da sie sich auf (bisher noch nicht behandelte) Objekttypen und nicht auf Agententypen beziehen.

Spezifikation 14 zeigt einen individuell spezifizierten Agenten und den zugehörigen geschlossenen Agententyp im Beispiel. Der Agent „Madonna“ betritt das Grid in Zelle $(0, 0)$, verfügt über die Eigenschaften „Moody“ und „Rich“, besitzt eine unbegrenzte Sendestärke und einen Kapazitätsbedarf von 60. Für die restlichen Eigenschaften übernimmt der Agent die allgemeinen Vorgabewerte.

7.3.2.6 *Objekte*

Objekte und Objekttypen werden im `<objects>`-Abschnitt der Spezifikationsdatei spezifiziert. Objekte, die ein Agent beim Be-

```
<objects> <!--[0,1]-->
  <types> <!--[1,1]-->
    <type> <!--[1,∗]-->
      <parameters> <!--[0,1]-->
        <parameter/> <!--[1,∗]-->
      </parameters>
    </type>
  </types>
  <instances> <!--[0,1]-->
    <instance> <!--[1,∗]-->
      <parameters> <!--[0,1]-->
        <parameter/> <!--[1,∗]-->
      </parameters>
      <contains> <!--[0,1]-->
        <instance> <!--[1,∗]-->
          <parameters> <!--[0,1]-->
            <parameter/> <!--[1,∗]-->
          </parameters>
          <contains> <!--[0,1]-->
<!-- beliebig tiefe <contains>-Schachtelung -->
          </contains>
        </instance>
      </contains>
    </instance>
  </instances>
</objects>
```

Spezifikation 15: Struktur der Objektspezifikation (ohne Attribute)

treten der Gridwelt in seinem Inventar vorhalten soll, werden darüber hinaus innerhalb eines <contains>-Elements im <agents>-Abschnitt spezifiziert.

Das generelle Vorgehen bei der Spezifikation von Objekttypen und Objektinstanzen entspricht dem Vorgehen bei der Spezifikation individuell spezifizierter Agenten. Spezifikation 15 zeigt den allgemeinen Aufbau, wobei über <type>-Elemente Objekttypen spezifiziert werden, welche in <instance>-Elementen referenziert werden, um Objekte auf dem Grid zu instanziieren.

<type> und <instance> verfügen dabei über die gemeinsamen Attribute freeCap und capNeed sowie über die Möglichkeit, ein <parameters>-Element zu enthalten, welches <parameter>-Elemente beinhaltet. Die Bedeutung und die Vorgabewerte sind hierbei analog zur Spezifikation von Agenten. Ebenso gilt auch hier, dass die spezifischere Angabe Vorrang vor der allgemeineren Angabe besitzt (<instance> hat Vorrang vor <type>), falls beide Angaben in der Spezifikation eines Objekts auftreten, und

dass die Parameter beider für das Objekt relevanten Abschnitte der Java-Klasse übergeben werden, die das Objekt implementiert.

Das Element `<type>` verfügt darüber hinaus über die Attribute `typeName` und `class`, mit analoger Bedeutung wie beim `<type>`-Element für geschlossene Agententypen. `<instance>`-Elemente referenzieren den `typeName` von `<type>`-Elementen, um den Objekttyp des Objekts festzulegen. Darüber hinaus verfügen sie über das optionale String-Attribut `name`, mit welchem dem Objekt ein Name zugewiesen werden kann. Wird das `name`-Attribut verwendet, muss dessen Wert eindeutig sein und darf auch nicht dem Wert eines `name`-Attributs aus dem `<agents>`-Abschnitt entsprechen. Ist das Attribut nicht gesetzt, wird der Name automatisch aus dem `typeName` erzeugt. `<instance>`-Elemente außerhalb von `<contains>` besitzen die notwendigen Attribute `xPos` und `yPos`, die angeben, in welcher Gridzelle das Objekt erzeugt werden soll.

`<instance>`-Elemente können ein `<contains>`-Element beinhalten, welches eine beliebige Anzahl weiterer `<instance>`-Elemente enthält, die wiederum ein `<contains>`-Element beinhalten können. Dies führt sich rekursiv fort, so dass eine beliebig tiefe Schachtelung von `<contains>`- und `<instance>`-Elementen möglich ist (wobei ein jedes `<contains>`-Element immer mindestens ein `<instance>`-Element umschließen muss). Die `<instance>`-Elemente eines `<contains>`-Elements spezifizieren dabei die Objekte, die im Inventar des jeweils umschließenden Objekts enthalten sind.

Da Agenten nicht Teil eines Inventars sein können, gilt auch für die `<contains>`-Abschnitte von geschlossenen Agententypen, dass diese `<instance>`-Elemente (ohne `xPos`- und `yPos`-Attribute) beinhalten, welche Objekttypen referenzieren.

Spezifikation 16 demonstriert ein Schließfach mit Property „ExtraFirm" in Gridzelle $(4, 0)$ mit einem Kapazitätsbedarf von 10 und einer freien Kapazität von 30, welches ein generisches Objekt mit dem Namen „GrandmasRing" enthält, das über einen Kapazitätsbedarf von 2 und eine freie Kapazität von 0 verfügt und die Property „Valuable" besitzt.

Es gilt auch bei Objekten, dass der `typeName` nur der Referenzierung und ggf. der Erzeugung des Objektnamens (falls dieser nicht spezifiziert ist) dient und keine beobachtbare Eigenschaft des Objekts darstellt. Sofern dies nicht als Property spezifiziert wird, ist der Umstand, dass es sich bei „GrandmasRing" um „jewelry" handelt, nicht Teil der Umgebung.

7.3.2.7 *Hindernisse*

Hindernisse werden im `<obstacles>`-Abschnitt der Spezifikationsdatei spezifiziert. `<obstacles>` kann Elemente der Typen `<walls>` (für Mauern), `<trenches>` (für Gräben), `<curtains>` (für

```
<objects>
  <types capNeed="10" freeCap="30">
    <type typeName="locker" class="net.tittel.
     gridworldsim.server.LockerObject"/>
    <type typeName="jewelry" freeCap="0" capNeed="2">
      <parameters>
        <parameter paraName="property"
         value="Valuable"/>
      </parameters>
    </type>
  </types>
  <instances>
    <instance objectTypeRef="locker" xPos="4" yPos="0">
      <parameters>
        <parameter paraName="property"
         value="ExtraFirm"/>
      </parameters>
      <contains>
        <instance objectTypeRef="jewlry"
         name="GrandmasRing"/>
      </contains>
    </instance>
  </instances>
</objects>
```

Spezifikation 16: `<objects>`-Element und Nachfahren im Beispiel

```xml
<obstacles>
  <walls interfering="true">
    <multi xPos="1" yPos="1" length="6" direction="e"/>
  </walls>
  <trenches>
    <multi xPos="2" yPos="3" length="10" direction="e"/>
    <single xPos="8" yPos="4"/>
  </trenches>
  <curtains>
    <single xPos="10" yPos="0"/>
  </curtains>
  <interferences>
    <multi xPos="9" yPos="9" length="4" direction="e"/>
  </interferences>
</obstacles>
```

Spezifikation 17: `<obstacles>`-Element und Nachfahren im Beispiel

Vorhänge) und `<interferences>` (für Interferenzen) umfassen, wobei jedes dieser Element höchstens einmal vorkommen darf.

`<walls>`-, `<trenches>`- und `<curtains>`-Elemente besitzen das optionale boolsche Attribut `interfering`. Ist dieses `true`, so sind alle Hindernisse des jeweiligen Typs gleichzeitig auch Interferenzen. Damit können z. B. alle Mauern komfortabel als kommunikationsblockierend definiert werden, ohne dass für jede Mauerspezifikation eine identische Interferenzspezifikation erfolgen muss. Ist das Attribut nicht gesetzt, so gilt es als `false`.

Alle vier Hinderniselemente können beliebig viele Elemente der Typen `<multi>` und `<single>` beinhalten. Ersteres gibt ein Hindernis an, welches mehrere Zellen umspannt, zweiteres ein Hindernis, welches nur eine Zelle umfasst. Für `<multi>` gelten die folgenden Attribute:

- `xPos` und `yPos`: Diese notwendigen Attribute geben die Startzelle des Hindernisses an.

- `direction`: Dieses notwendige Attribut gibt an, in welche Richtung sich das Hindernis erstrecken soll. Mögliche Werte sind `n` wie Norden, `e` wie Osten, `s` wie Süden und `w` wie Westen.

- `length`: Durch dieses notwendige Attribut wird angegeben, wie viele Zellen das Hindernis in der angegebenen Richtung umfassen soll.

Das Element `<single>` umfasst nur die zwingenden Attribute `xPos` und `yPos`, welche die Zelle des Hindernisses angeben.

```
<cellcapacities default="90">
  <cell xPos="0" yPos="0" capacity="unbounded"/>
  <cell xPos="3" yPos="8" capacity="23"/>
  <cell xPos="9" yPos="1" capacity="1"/>
</cellcapacities>
```

Spezifikation 18: `<cellcapacities>`-Element und Nachfahren im Beispiel

Spezifikation 17 zeigt ein Beispiel zur Spezifikation von Hindernissen. Die Möglichkeit, größere Hindernisse kompakt mittels `<multi>` zu spezifizieren, reduziert den Aufwand bei der Spezifikation größerer Gridwelten mit längeren Hindernissen erheblich.

7.3.2.8 *Zellkapazitäten*

Das Element `<cellcapacities>` umschließt den Abschnitt, der die Zellkapazitäten definiert. Es verfügt über ein notwendiges Attribut namens `default`, dessen Wertebereich 0 bis $2^{31} - 1$ und `unbounded` umfasst und das den Vorgabewert für die Kapazität von Zellen setzt. Alle Zellen, für die keine andere Kapazität spezifiziert wird, übernehmen diesen Vorgabewert.

Innerhalb eines `<cellcapacities>`-Elements können beliebig viele `<cell>`-Elemente vorkommen. Diese haben die folgenden notwendigen Attribute:

- `xPos` und `yPos`: Diese Attribute geben die Koordinaten der Zelle an, deren Kapazität spezifiziert werden soll.
- `capacity`: Dieses Attribut mit Wertebereich 0 bis $2^{31} - 1$ und `unbounded` spezifiziert die Kapazität der Zelle.

Ein Beispiel dazu gibt Spezifikation 18.

7.4 Kommunikationsprotokoll

Die Kommunikation zwischen Server und Clients findet über eine TCP-Verbindung statt, über welche XML-Dokumente versendet werden. Damit die jeweilige Gegenstelle weiß, wie lang das nächste zu übertragende XML-Dokument ist, werden vor jedem XML-Dokument vier Bytes versendet, welche die Länge des nachfolgenden XML-Dokuments im Zweierkomplement kodieren. Da die Länge von Dokumenten immer positiv ist, findet das Zweierkomplement effektiv keine Anwendung und es werden nur 31 der insgesamt 32 Bit verwendet. Die maximale Größe eines XML-Dokuments darf daher zwei Gibibyte nicht übersteigen.

Zur Kommunikation zwischen Server und Agenten-Client und zur Kommunikation zwischen Server und Beobachter-Client werden die gleichen XML-Sprachen verwendet. Es finden jedoch

für die unterschiedlichen Kommunikationsrichtungen (Server → Client und Client → Server) unterschiedliche XML-Sprachen Verwendung, die im Folgenden erläutert werden.

7.4.1 *KQML*

Bei KQML (vgl. [CFF+92]) handelt es sich um ein Protokoll zur Kommunikation zwischen Agenten und zwischen Expertensystemen. Eine von GridWorldSim realisierte Umgebung stellt dabei weder das eine noch das andere dar. Insbesondere findet keine von Wissensanfragen und zugehörigen Antworten geprägte Kommunikation zwischen Agenten und GridWorldSim-Server statt. Auch wenn eine Adaption von KQML zur Verwendung als Protokoll zur Kommunikation zwischen Agenten bzw. Beobachter-Clients und dem GridWorldSim-Server trotzdem grundsätzlich denkbar wäre, so würde dabei nur ein verschwindend geringer Teil des Sprachumfangs von KQML Verwendung finden. Deshalb werden stattdessen in den nächsten beiden Abschnitten erheblich einfachere Kommunikationsprotokolle vorgestellt.

Davon unberührt können Agenten, die KQML verwenden, um miteinander zu kommunizieren, dies tun, indem sie KQML-Nachrichten in GridWorldSim-Nachrichten kapseln.

7.4.2 *Aktionsanforderungen*

Die XML-Sprache, welche in Kommunikationsrichtung Client → Server Verwendung findet, dient der Kodierung von Aktionsanforderungen. Ein Beobachter-Client nimmt zwar nicht handelnd an einer Umgebung teil, kann jedoch das sofortige Auslösen einer Zustandsüberführung anfordern. Eine solches Anfordern wird im Sinne der XML-Sprache als parameterlose Aktionsanforderung des Types `statetrans` behandelt. Davon unberührt dient die in diesem Abschnitt vorgestellte Sprache hauptsächlich dem Versand von Aktionsanforderungen von Agenten an den Server.

Das zu dieser Sprache gehörige Schema findet sich auf der beiliegenden CD-ROM unter `/schemas/actionrequest.xsd`.

Bei Aktionsanforderungen gemäß Kapitel 4.4 handelt es sich um Prädikate mit vorgegebener Stelligkeit. Die einzelnen Stellen werden für die Übertragung per XML abhängig vom Typ der Aktionsanforderung benannt, so dass Wertpaare aus Parametername und -wert entstehen. Für Prädikatstellen, die eine Richtung kodieren, findet bspw. der Parametername `direction` Verwendung. Die Stelle in Aktionsanforderungen, welche den handelnden Agenten angibt, wird nicht berücksichtigt. Die Zuordnung von Aktionsanforderungen zu Agenten wird vom GridWorldSim-Server vorgenommen, abhängig davon, über welche

```
<action xsi:noNamespaceSchemaLocation="http://www.
 tittel.net/gridworldsim/schemas/actionrequest-1.0.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 type="move">
  <parameters>
    <parameter>
      <name>direction</name>
      <value>e</value>
    </parameter>
  </parameters>
  <internalstate>Happy</internalstate>
</action>
```

Spezifikation 19: XML-Dokument für Aktionsanforderungen im Beispiel

TCP-Verbindung eine Aktionsanforderung empfangen wurde. Dadurch wird verhindert, dass Agenten Aktionsanforderungen im Namen anderer Agenten stellen können.

Die XML-Sprache ist vergleichsweise einfach aufgebaut. Das Wurzelelement <action> beinhaltet das notwendige String-Attribut type, höchstens ein Element mit dem Namen <parameters> und höchstens ein Element mit dem Namen <internalstate>. Das Attribut type gibt dabei die Art der Aktionsanforderung an.

Das <parameters>-Element beinhaltet eine beliebige Anzahl von <parameter>-Elementen (mindestens jedoch eins). Ein <parameter>-Element besitzt die notwendigen Elemente <name> und <value>, welche den Parameternamen und -wert als nicht leeren String beinhalten.

Es soll im Beobachter-Client möglich sein, die internen Zustände von Agenten einzusehen. Dazu kann ein Agent zusammen mit einer Aktionsanforderung dem Server seinen internen Zustand mitteilen. Dies geschieht durch Verwendung des Elements <internalstate>, welches den internen Zustand als nicht leeren String beinhaltet.

Spezifikation 19 zeigt ein XML-Dokument für Aktionsanforderungen im Beispiel. Der Agent besitzt den internen Zustand „Happy“ und möchte sich gerne Richtung Osten bewegen.

Spezifikation 20 zeigt die Aktionsanforderungen aus den Kapiteln 4.4, 4.11.2 und 6.2 und ihre Entsprechungen hinsichtlich der XML-Sprache für Aktionsanforderungen. Dabei bedeuted „k. E.“, dass keine Entsprechung existiert, da der GridWorldSim-Server den Wert des entsprechenden Parameters selbst feststellen kann.

Für declareAccept und rectractAccept gilt außerdem: Wird nur der Parameter agent angegeben, so werden von diesem Agenten alle Objekte akzeptiert bzw. nicht mehr akzeptiert. Gleichermaßen wird ein Objekt von allen Agenten akzeptiert bzw. nicht

Prädikat	type	Zuordnung
Move(a, d)	`move`	d = `direction`
Take(a, o)	`take`	o = `object`
Release(a, o)	`release`	o = `object`
MoveObject(a, o, d)	`moveObject`	o = `object`, d = `direction`
HandOver(a, a_1, o)	`handOver`	a_1 = `agent`, o = `object`
DeclareAccept(a, a_1, o)	`declareAccept`	a_1 = `agent`, o = `object`
RetractAccept(a, a_1, o)	`retractAccept`	a_1 = `agent`, o = `object`
Load(a, o_1, o_2)	`load`	o_1 = `object`, o_2 = `receiver`
UnloadToGrid(a, o_1, o_2)	`unload`	o_1 = `object`, o_2 = k. E., `destination` = `grid`
UnloadToInventory(a, o_1, o_2)	`unload`	o_1 = `object`, o_2 = k. E., `destination` = `inventory`
NoOp(a)	`noop`	
Lock(a, b, p)	`lock`	b = `object`, p = `password`
Unlock(a, b, p)	`unlock`	b = `object`, p = `password`
SendPubMsg(a, m)	`publicMsg`	m = `message`
SendPrivMsg(a, a_1, m)	`privateMsg`	a_1 = `receiver`, m = `message`

Spezifikation 20: Zuordnung von Aktionsanforderungen

```
<gridworld-percept xsi:noNamespaceSchemaLocation=
 "http://www.tittel.net/gridworldsim/schemas/
 perception-1.0.xsd" xmlns:xsi="http://www.w3.org/
 2001/XMLSchema-instance" yDimension="15"
 xDimension="17" time="5">
```

Spezifikation 21: Wurzelelement eines Wahrnehmungsdokuments im Beispiel

mehr akzeptiert, wenn nur der Parameter `object` angegeben wird (siehe Kapitel 7.2.9).

Bei Nachrichten wird empfohlen, dass diese in einem CDATA-Abschnitt gekapselt werden, damit der Inhalt vom XML-Parser des Servers und des Empfängers nicht geparst wird. Notwendig ist dies jedoch nicht.

7.4.3 *Wahrnehmung*

Die XML-Sprache, welche in Kommunikationsrichtung Server $\rightarrow$ Client verwendet wird, dient der Kodierung von Wahrnehmungen. Bei der folgenden Erläuterung des Formats gilt – sofern nicht anders angegeben – für Zahlenattribute der Wertebereich 0 bis $2^{31} - 1$ und für String-Attribute, dass es sich um nicht leere Strings handeln muss.

Das zugehörige Schema findet sich auf der beiliegenden CD-ROM unter `/schemas/perception.xsd`.

7.4.3.1 *Wurzelelement*

Das Wurzelelement der Sprache heißt `gridworld-percept` und besitzt die notwendigen Attribute `xDimension`, `yDimension` und `time` (Wertebereich 0 bis $2^{63} - 1$). `xDimension` und `yDimension` geben die Dimension der Gridwelt an und `time` die aktuelle Zeit, d. h. den Zeitpunkt t des aktuellen Zustands S_t. Bei diesen Informationen handelt es sich um Erweiterungen der in Kapitel 5.4 beschriebenen Beobachtungsmenge, die aus praktischen Gründen erfolgt. Spezifikation 21 zeigt dazu ein Beispiel.

Das Wurzelelement beinhaltet genau ein Element mit Namen `<cells>`[2] und höchstens ein Element mit dem Namen `<messages>`.

7.4.3.2 *Zellen*

Für jede im aktuellen Zustand wahrgenommene Zelle beinhaltet das `<cells>`-Element ein `<cell>`-Element, welches angibt, dass

2 Da ein Agent immer die Zelle, in der er sich aufhält, wahrnimmt, wird immer mindestens eine Zelle wahrgenommen.

```
<cells>
  <cell curtain="true" freeCap="50" xPos="0" yPos="0">
    <agents>
<!-- Angabe in der Zelle enthaltener Agenten -->
    </agents>
    <objects>
<!-- Angabe in der Zelle enthaltener Objekte -->
    </objects>
  </cell>
  <cell fog="true" freeCap="5" xPos="1" yPos="0"/>
</cells>
```

Spezifikation 22: `<cells>`-Element und Nachfahren im Beispiel

diese Zelle für den Agenten entweder vollständig sichtbar ist (also der Zellinhalt wahrgenommen werden kann) oder zumindest teilweise sichtbar ist (in der Zelle befindliche Objekte können nicht wahrgenommen werden, es kann aber wahrgenommen werden, dass die Zelle ein Hindernis vom Typ Mauer, Vorhang oder Nebel beinhaltet). Ein `<cell>`-Element beinhaltet die notwendigen Attribute `xPos` und `yPos`, welche die Position der Zelle kodieren. Dieser Teil des `<cell>`-Elements entspricht einem Grid(x, y)-Prädikat aus der Beobachtungsmenge gemäß Kapitel 5. Das ebenfalls notwendige Attribut `freeCap` (Wertebereich 0 bis $2^{31} - 1$ und `unbounded`) gibt die freie Kapazität der Zelle an und entspricht zusammen mit `xPos` und `yPos` den FreeCellCap(x, y, c)-Prädikaten aus der Beobachtungsmenge.

Darüber hinaus existieren die optionalen boolschen Attribute `wall`, `trench`, `fog`, `curtain` und `interference` mit Vorgabewert `false`, die angeben, ob sich in der jeweiligen Zelle eine Mauer, ein Graben, Nebel, ein Vorhang oder eine Interferenz befindet. Da Agenten Interferenzen nicht wahrnehmen, findet `interference` nur beim Versand von Wahrnehmungen an einen Beobachter-Client Verwendung. Dies entspricht zusammen mit `xPos` und `yPos` den Prädikaten Wall(x, y), Trench(x, y), Fog(x, y) und Curtain(x, y) aus der Beobachtungsmenge.

Des Weiteren kann ein `<cell>`-Element die Elemente `<agents>` und `<objects>` (in dieser Reihenfolge) beinhalten, um die in der Zelle enthaltenen Agenten und Objekte anzugeben. Spezifikation 22 zeigt ein `<cells>`-Element im Beispiel.

7.4.3.3 *Agenten*

Ein `<agents>`-Element beinhaltet ein oder mehrere `<agent>`-Elemente, welche die in der entsprechenden Zelle enthaltenen Agenten beschreiben. Diese entsprechen in Verbindung mit dem `xPos`- und `yPos`-Attribut der Zelle den Loc(x, y, a)-Fakten aus der Be-

obachtungsmenge für diese Zelle. Ein `<agent>`-Element verfügt dabei über die folgenden Attribute:

- `name`: Dieses notwendige String-Attribut gibt den Namen des Agenten an.
- `freeCap`: Dieses notwendige Attribut gibt die freie Kapazität des Agenten an (Wertebereich $2^{31} - 1$ und `unbounded`).
- `capNeed`: Dieses notwendige Attribut gibt den Kapazitätsbedarf des Agenten an.
- `isYou`: Dieses optionale boolsche Attribut gibt an, ob es sich bei dem durch `<agent>` beschriebenen Agenten um den Agenten handelt, der diese Wahrnehmung empfängt. Der Vorgabewert ist `false`.
- `soundIntensity`, `hearing` und `moveForce` sind optionale Attribute, welche die Sendestärke, Empfangsempfindlichkeit und Verschiebekraft des Agenten beschreiben (Wertebereich jeweils 0 bis $2^{31} - 1$ und `unbounded`).
- `priority`: Dieses optionale Attribut gibt die Priorität des Agenten hinsichtlich der Ausführungsreihenfolge von Aktionsanforderungen an.

Das Attribut `name` entspricht der Belegung von a für den jeweiligen Agenten. `freeCap` und `capNeed` finden ihre Entsprechung in Verbindung mit `name` in den FreeCap(z, c)- und CapNeed(z, c)-Prädikaten der Beobachtungsmenge. Bei `isYou` handelt es sich um ein Attribut, welches die Implementierung aus Komfortgründen unterstützt, da ansonsten in Verbindung mit GridWorldSim genutzte Agenten eine gewisse Mindestkomplexität aufweisen müssten, um ihren eigenen Standort bestimmen zu können. Das Attribut `isYou` findet nur bei der Kommunikation mit Agenten Verwendung, nicht bei der Kommunikation mit Beobachter-Clients.

Die Attribute `soundIntensity`, `hearing`, `moveForce` und `priority` werden hingegen nur an Beobachter-Clients versendet und haben daher keine Entsprechung in der in Kapitel 5 beschriebenen Beobachtungsmenge.

Ein `<agent>`-Element kann drei verschiedene Elemente höchstens je einmal enthalten: `<properties>`, `<objects>` und `<internalstate>`.

Das Element `<properties>` beinhaltet ein oder mehrere `<property>`-Elemente, die über ein notwendiges String-Attribut `value` verfügen, welches eine beobachtbare Property angibt. Dies entspricht den benutzerdefinierten Objekt- bzw. Agenteneigenschaften $U(o)$ gemäß Kapitel 4.2 und 5.4.

```
<agents>
  <agent capNeed="8" freeCap="5" isYou="true"
   name="Tweety">
    <properties>
      <property value="Bird"/>
      <property value="Penguin"/>
    </properties>
    <objects>
<!-- Objekte im Inventar -->
    </objects>
  </agent>
  <agent capNeed="3" freeCap="1" name="Geeko"/>
</agents>
```

Spezifikation 23: <agents>-Element und Nachfahren im Beispiel

Ein <objects>-Element gibt die (bisher noch nicht beschriebene) Wahrnehmung von Objekten an, die der Agent in seinem Inventar vorhält und <internalstate> dient der Mitteilung des internen Zustands des jeweiligen Agenten zwecks Weiterleitung an Beobachter-Clients. Spezifikation 23 gibt dazu ein Beispiel, wobei es sich um ein Wahrnehmungsdokument für einen Agenten und nicht für einen Beobachter-Client handelt. Im Falle eines Beobachter-Clients wären die Attribute `soundIntensity`, `hearing`, `moveForce` und `priority` gesetzt, das Attribut `isYou` wäre nicht möglich und es könnte zudem ein <internalstate>-Element geben, das den internen Zustand des Agenten beschreibt.

7.4.3.4 *Objekte*

Wahrgenommene Objekte werden als <object>-Elemente innerhalb eines <objects>-Elements angegeben. Ein <objects>-Element kann dabei entweder als direkter Nachfahre eines <cell>-Elements auftreten (falls sich das Objekt in einer Gridzelle befindet) oder als Nachfahre eines <agent>- oder <object>-Elements (falls sich das Objekt im Inventar eines Agenten oder anderen Objekts befindet). Befindet sich ein <object>-Element innerhalb eines <objects>-Elements, welches in einem <cell>-Element enthalten ist, so entspricht dies in Verbindung mit den Attributen `xPos` und `yPos` der Zelle den Fakten des Typs $\text{Loc}(x, y, a)$ aus der Beobachtungsmenge. Ist das <objects>-Element hingegen in einem <agent>- oder <object>-Element enthalten, so findet das Element seine Entsprechung in den $\text{Contains}(a, o)$- bzw. $\text{Contains}(z, o)$-Fakten der Beobachtungsmenge.

Für Agenten gilt, dass sie nur die Objekte wahrnehmen können, die sich in einer Gridzelle oder in ihrem Inventar befinden oder die sich in einem Objekt befinden, welches sich in ihrem Inventar

```
<objects>
  <object capNeed="5" freeCap="30" name="Locker.0">
    <properties>
      <property value="Unlocked"/>
      <property value="Locker"/>
    </properties>
    <objects>
      <object capNeed="3" freeCap="0" name="Shirt.0"/>
    <objects>
  </object>
</objects>
```

Spezifikation 24: `<objects>`-Element und Nachfahren im Beispiel

befindet oder die sich im Inventar eines Objekts befinden, welches sich in einer Gridzelle befindet. Für Beobachter-Clients gilt hingegen, dass sie beliebig tiefe Objektschachtelungen wahrnehmen können, weshalb die Tiefe der Schachtelung von `<objects>`- und `<object>`-Elementen nicht begrenzt ist.

Ein `<object>`-Element ist hinsichtlich seiner Attribute und hinsichtlich der Elemente mit einem `<agent>`-Element identisch, mit Ausnahme der Attribute `isYou`, `soundIntensity`, `hearing`, `moveForce` und `priority` sowie dem möglichen Element `<internalstate>`. Diese agentenspezifischen Komponenten entfallen bei Objekten. Die Entsprechung in der Beobachtungsmenge verhält sich ebenfalls analog, wobei es eine Erweiterung gibt: Bei der Spezifikation eines Objekts mit den Namen o finden SafeDepositBox(o) und Locked(z) ihre Entsprechung in den Properties `Locker` und `Locked`. Im Gegensatz zur Wahrnehmungsmenge wird in einem XML-Wahrnehmungsdokument zudem auch die Property `Unlocked` explizit angegeben, wenn das Schließfach nicht verschlossen ist.

Spezifikation 24 demonstriert ein unverschlossenes Schließfach, welches ein Objekt namens „Shirt.0“ beinhaltet. Dieser `<objects>`-Abschnitt könnte sowohl in einem `<cell>`-Element, als auch in einem `agent`- oder `object`-Element auftreten.

7.4.3.5 *Nachrichten*

Der Abschnitt des XML-Dokuments, der Nachrichten beinhaltet, wird durch ein Element namens `<messages>` eingeleitet. Ein solches `<messages>`-Element enthält ein oder mehrere `<message>`-Elemente. Ein `<message>`-Element besitzt die folgenden Attribute:

- `sender`: Dieses optionale String-Attribut gibt den Absender einer Nachricht an. Im Falle einer öffentlichen Nachricht, deren Absender nicht bekannt ist, wird das Attribut nicht

gesetzt. Für Beobachter-Clients ist der Absender immer bekannt.

- `receiver`: Dieses optionale String-Attribut findet nur bei Wahrnehmungsdokumenten Verwendung, welche an Beobachter-Clients versandt werden und gibt den Empfänger einer privaten Nachricht an. Im Falle des Empfangs einer privaten Nachricht durch einen Agenten ist dies nicht notwendig, da es sich beim Empfänger immer um den Agenten selbst handelt.

- `isPublic`: Dieses notwendige boolsche Attribut gibt an, ob es sich um eine öffentliche Nachricht (`true`) oder um eine private Nachricht handelt (`false`).

Beim Nachrichteninhalt selbst handelt es sich um den Inhalt des `<message>`-Elements.

Die Erweiterung der Beobachtungsmenge um Nachrichten wird in Kapitel 6.6 angegeben. Dabei gilt, dass eine Nachricht, für die `isPublic` den Wert `true` besitzt, aber das Attribut `sender` nicht gesetzt ist, $\text{AnonPubMsg}(a, m)$ entspricht. m ist dabei der Nachrichteninhalt und a der Agent, der die Nachricht empfängt. Da durch den Empfang einer Nachricht m durch den Agenten a für diesen implizit gilt, dass a die Nachricht empfangen kann, gibt es keine Entsprechung für a im Wahrnehmungsdokument.

Nachrichten, für die `isPublic` gilt und bei denen zusätzlich das Attribut `sender` gesetzt ist, entsprechen Fakten des Typs $\text{PubMsg}(z, a, m)$. Dabei entspricht z dem Attribut `sender`, m der Nachricht und a ist der Agent, der die Nachricht empfängt. Aus den gleichen Gründen wie bei $\text{AnonPubMsg}(a, m)$ ist a im Wahrnehmungsdokument nicht enthalten.

Private Nachrichten verhalten sich analog zu öffentlichen Nachrichten mit bekanntem Absender, mit dem Unterschied, dass `isPublic` den Wert `false` besitzt.

Spezifikation 25 zeigt eine öffentliche Nachricht ohne bekannten Absender, eine öffentliche Nachricht mit bekanntem Absender und eine private Nachricht im Beispiel. Es handelt sich dabei um ein Wahrnehmungsdokument, welches an einen Agenten versandt wird. Würde das Wahrnehmungsdokument an einen Beobachter-Client verschickt, wäre der Absender der ersten Nachricht bekannt und die private Nachricht würde über das Attribut `receiver` verfügen, welches den Empfänger angibt. Das Beispiel folgt der Empfehlung aus Kapitel 7.4.2, die Nachrichteninhalte in CDATA-Abschnitten zu kapseln.

```xml
<messages>
 <message isPublic="true">
   <![CDATA[Hey everybody!]]>
 </message>
 <message isPublic="true" sender="Apollo">
   <![CDATA[Meow!]]>
 </message>
 <message isPublic="false" sender="Madonna">
   <![CDATA[Sigmund Freud, analyze this!]]>
 </message>
</messages>
```

Spezifikation 25: `<messages>`-Element und Nachfahren im Beispiel

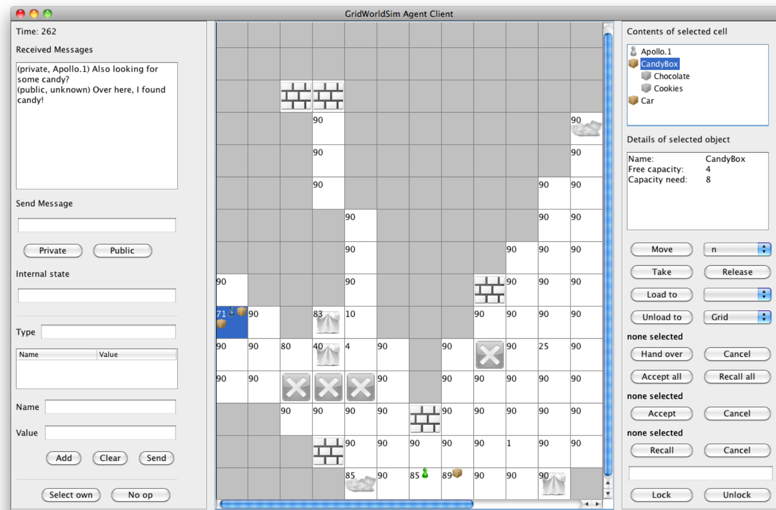

Abbildung 20: Screenshot des Agenten-Clients

7.5 CLIENTS

Neben dem Server bietet GridWorldSim zwei verschiedene grafische Clients an: Den Agenten-Client, welcher sich als Agent mit dem Server verbindet und es dem Benutzer erlaubt, selbst Aktionen in der Gridwelt auszuführen, sowie den Beobachter-Client, der zur Beobachtung der Gridwelt dient. Die Benutzung beider Clients wird in diesem Abschnitt beschrieben.

7.5.1 *Agenten-Client*

Abbildung 20 zeigt den Aufbau des Agenten-Client. Über das (nicht abgebildete) Menü „Connection“ können Verbindungen zum Server auf- und abgebaut werden. Im Gegensatz zum Be-

Norden:	„e“ oder „Pfeiltaste oben“
Nord-Osten:	„r“
Osten:	„f“ oder „Pfeiltaste rechts“
Süd-Osten:	„v“
Süden:	„c“ oder „Pfeiltaste unten“
Süd-Westen:	„x“
Westen:	„s“ oder „Pfeiltaste links“
Nord-Westen:	„w“

Spezifikation 26: Tastenzuordnung zur Bewegung des Agenten

obachter-Client nimmt der Agenten-Client die Gridwelt im Allgemeinen nicht vollständig wahr, sondern kann nur den Teil darstellen, den der zugehörige Agent in der Lage ist wahrzunehmen.

Die Benutzeroberfläche des Agenten-Clients gliedert sich in drei Teile: Der linke Teil dient der Anzeige des aktuellen Zeitpunkts, dem Nachrichtenempfang- und versand, der Festlegung des internen Zustands des Agenten, dem Versand händisch spezifizierter Aktionsanforderungen, der Selektierung der Zelle, die den gesteuerten Agenten beinhaltet, und dem Versand von NoOp-Aktionsanforderungen.

Der mittlere Teil stellt die Gridzellen der Gridwelt samt den Zellinhalten und Hindernistypen grafisch dar. Gilt für eine Zelle, dass sie für den Agenten vollständig unsichtbar ist (d. h. der Agent kann nicht nur die enthaltenen Inhalte, sondern auch die Hindernisse der Zelle sowie die Zellkapazität nicht erkennen), so ist sie grau unterlegt. Die Zahl oben links in jeder Zelle gibt die freie Zellkapazität an.

Der rechte Teil dient der Darstellung des Inhalts der selektierten Zelle in Form eines ausklappbaren Baums, der Anzeige von Details zu dem im Baum selektierten Objekt und der Auslösung von Aktionsanforderungen.

Der Agent, der vom Benutzer gesteuert wird, ist in grüner Farbe dargestellt. Seine Zellposition kann jederzeit durch Betätigung der Schaltfläche **Select own** im linken Teil selektiert werden. Der Versand eines XML-Dokuments zur Bewegung des Agenten wird mittels Tastatur angefordert, wobei die in Spezifikation 26 angegebene Tastenzuordnung gilt. Die Tastatursteuerung funktioniert nur dann, wenn derzeit kein Texteingabefeld selektiert ist.

Tabellenzellen können mit Hilfe der Maus selektiert werden. Der wahrgenommene Inhalt der gewählten Zelle wird im ausklappbaren Baum oben rechts dargestellt. Ist im Baum ein Objekt Kindelement eines anderen Objekts, so befindet es sich in dessen Inventar. Das Symbol von Objekten, die über andere Objekte in ihrem Inventar verfügen, ist dabei im eingeklappten Zustand

hervorgehoben und das Objekt kann in diesem Falle über einen Doppelklick expandiert werden. Das Element für den eigenen Agenten wird dabei automatisch expandiert, damit sein Inventar ohne Eingriff des Benutzers sichtbar ist. Das Textfeld unterhalb des Baums zeigt die wahrgenommenen Eigenschaften des im Baum selektierten Agenten oder Objekts an.

Die Betätigung von Schaltflächen dient dem Versand von XML-Dokumenten, welche Aktionsanforderungen repräsentieren. Die Bedeutung der Schaltflächen ist im Folgenden informell erklärt, eine genaue Beschreibung, welche XML-Dokumente welchen Aktionsanforderungen gemäß Kapitel 4.4, 4.11.2 und 6.2 entsprechen und wie XML-Dokumente für Aktionsanforderungen genau beschaffen sind, findet sich in Kapitel 7.4.2.

- Die Schaltfläche **Move** dient dem Verschieben von Objekten (nicht der Bewegung des eigenen Agenten) und versendet nach Betätigung eine Aktionsanforderung zum Verschieben des im Baum selektierten Objekts in die rechts von der Schaltfläche gewählte Richtung.

- Die Schaltflächen **Take** und **Release** dienen dem Aufnehmen eines Objekts, welches sich direkt auf dem Grid befindet in das Inventar des Agenten und dem Ablegen eines Objekts, welches sich direkt im Inventar des Agenten befindet, auf das Grid.

- Mit der Schaltfläche **Load to** wird ein XML-Dokument erzeugt, welches das Beladen des rechts von der Schaltfläche ausgewählten Objekts mit dem im Baum ausgewählten Objekt anfordert. Im Unterschied zu „Take“ und in Übereinstimmung mit den von Load(a, o_1, o_2) ausgelösten Regeln gemäß Kapitel 4.7.2.5 kann ein Agent auf diese Weise kein Objekt in sein Inventar aufnehmen.

- Zur Anforderung des Entfernens eines Objekts aus dem Inventar eines anderen Objekts dient die Schaltfläche **Unload to**. Dabei wird eine Anforderung erzeugt, das im Baum selektierte Objekt aus dem Inventar des umgebenden Objekts zu entfernen und an den rechts von der Schaltfläche ausgewählten Ort (wahlweise die aktuelle Zelle oder das eigene Inventar) zu bewegen. Auch hier gilt, dass es im Unterschied zu „Release“ dabei nicht möglich ist, ein Objekt aus dem eigenen Inventar abzulegen, da es sich dabei um unterschiedliche Aktionsanforderungen handelt.

- Die Schaltfläche **Hand over** dient dazu, die Übergabe eines Objekts aus dem eigenen Inventar an einen anderen Agenten anzufordern. Die beiden Parameter „zu übergebendes Objekt“ und „Agent, der das Objekt empfangen

soll“ werden dabei in zwei Schritten ausgewählt: Zunächst wird einer dieser beiden Parameter im Baum selektiert, danach wird die Hand-over-Schaltfläche betätigt. Der erste Parameter ist nun registriert und wird angezeigt. Im zweiten Schritt wird der andere Parameter im Baum selektiert. Nach erneuter Betätigung der Hand-over-Schaltfläche wird das entsprechende XML-Dokument versandt. Der Agenten-Client stellt dabei sicher, dass es sich bei einem Parameter um einen Agenten und beim anderen Parameter um ein Objekt handelt und nimmt keinen weiteren Agenten bzw. kein weiteres Objekt als Parameter an, wenn bereits ein Agent bzw. Objekt als Parameter registriert ist. Die rechts liegende Schaltfläche **Cancel** macht die Registrierung eines eventuell registrierten ersten Parameters rückgängig.

- Die Schaltfläche **Accept all** erklärt, abhängig davon, ob im Baum ein Agent oder Objekt selektiert ist, dass der Agent alle Objekte vom selektierten Agenten bzw. das selektierte Objekt von allen Agenten anzunehmen bereit ist.

- Analog dazu wird mit der Schaltfläche **Retract all** die Bereitschaft widerrufen, alle Objekte vom gewählten Agenten bzw. das gewählte Objekt von allen Agenten anzunehmen.

- Mit der Schaltfläche **Accept** wird die Bereitschaft erklärt, ein bestimmtes Objekt von einem bestimmten Agenten anzunehmen. Die Auswahl der Parameter erfolgt dazu wie bei Hand-over.

- Mit der Schaltfläche **Retract** wird die Bereitschaft widerrufen, ein bestimmtes Objekt von einem bestimmten Agenten anzunehmen. Auch hier erfolgt die Auswahl der Parameter wie bei Hand-over.

- Mit den Schaltflächen **Lock** und **Unlock** wird eine Anforderung erzeugt, das im Baum gewählte (Schließfach-)Objekt mit dem im Feld über der Schaltfläche spezifizierten Passwort zu öffnen oder zu schließen.

- Die Schaltfläche **Public** auf der linken Seite erzeugt die Anforderung, den Inhalt des über der Schaltfläche liegenden Textfelds als öffentliche Nachricht zu verschicken.

- Die Schaltfläche **Private** erzeugt die Anforderung, den Inhalt des über ihr liegenden Textfelds als private Nachricht an den im Baum selektierten Agenten zu versenden. Der Agenten-Client verschickt die Anforderung nicht, wenn ein Objekt oder der eigene Agent selektiert ist.

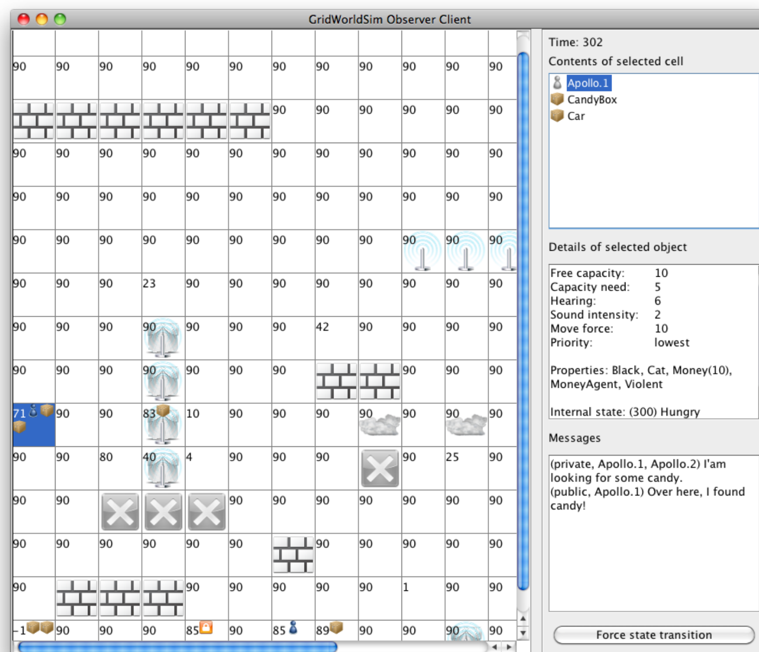

Abbildung 21: Screenshot des Beobachter-Clients

- Das mit **Internal State** bezeichnete Textfeld dient dazu, zusammen mit der nächsten Aktionsanforderung den Inhalt dieses Textfelds als Information über den aktuellen internen Zustand des Agenten zu verschicken.

- Die Schaltfläche **NoOp** versendet eine NoOp-Aktionsanforderung.

Links unten besteht zudem die Möglichkeit, eigene XML-Aktionsanforderungsdokumente zu erstellen (siehe Kapitel 7.4.2). Der Wert des `type`-Attributs wird dazu in das mit **Type** gekennzeichnete Textfeld eingetragen. Paare aus Parameternamen und -werten können in den unter der Tabelle befindlichen Textfeldern eingetragen und dem Dokument durch Betätigung der Schaltfläche **Add** hinzugefügt werden. Mittels der Schaltfläche **Clear** können die in der Tabelle selektierten Parameter aus dem Dokument wieder entfernt werden. Die Betätigung der Schaltfläche **Send** versendet das auf diese Weise erzeugte XML-Dokument an den Server.

7.5.2 *Beobachter-Client*

Der Aufbau des Beobachter-Clients ist in Abbildung 21 dargestellt. Die einzige vom Beobachter-Client unterstützte Aktion

ist das manuelle Auslösen einer Zustandsüberführung. Dies geschieht durch die Schaltfläche **Force state transition**.

Davon abgesehen unterscheidet sich der Beobachter-Client vom Agenten-Client vor allem dadurch, dass ihm die Schaltflächen zur Auslösung von Aktionsanforderungen fehlen. Der Umstand, dass ein Beobachter-Client im Gegensatz zum Agenten-Client vollständige Wahrnehmungen der Gridwelt darstellt, liegt darin begründet, dass der Server einem als Beobachter-Client angemeldeten Client vollständige Wahrnehmungen sendet. Die zur Darstellung der Gridwelt und zur Kommunikation mit dem Server verwendeten Komponenten sind beim Agenten-Client und Beobachter-Client identisch. Somit wäre es möglich, sich mit einem Agenten-Client als Beobachter anzumelden (wobei die Schaltflächen des Agenten-Client nicht funktionieren und nur unnötig Platz einnehmen würden) und sich mit einem Beobachter-Client als Agenten-Client anzumelden (wobei der Beobachter-Client keine Aktionsanforderungen für den Agenten stellen könnte).

Die Unterschiede in der vom Server erhaltenen Wahrnehmung stellen sie wie folgt dar:

- Ein Beobachter-Client nimmt immer alle Gridzellen der Gridwelt samt Inhalt wahr.

- Während ein Agent Objekte, welche sich im Inventar eines Objekts befinden, das sich im Inventar eines anderen Objekts befindet, nicht wahrnehmen kann, werden einem Beobachter-Client die Enthaltenseinsbeziehungen von Objekten in beliebiger Tiefe mitgeteilt.

- Ein Beobachter-Client bekommt die internen Zustände von Agenten im mit **Details of selected object** bezeichneten Textfeld angezeigt. Die Zahl in Klammern gibt dabei an, in welchem Umgebungszustand die Zustandsinformation vom Agenten das letzte Mal aktualisiert wurde.

- Im Gegensatz zu Agenten nimmt ein Beobachter-Client Interferenzen wahr.

- Im Falle privater Nachrichten erhält ein Beobachter-Client neben Absender und Inhalt auch den Empfänger der Nachricht. Ferner nimmt er alle öffentlichen und privaten Nachrichten in der Umgebung wahr. Bei öffentlichen Nachrichten kennt er dabei immer den Absender.

- Ein Beobachter-Client bekommt alle nicht öffentlichen Eigenschaften von Agenten und Objekten mitgeteilt. So sieht er zum Beispiel das gesetzte Passwort eines verschlossenen

Schließfachs oder die Sendestärke, Empfangsempfindlichkeit und Priorität von Agenten.

7.6 ERWEITERUNGEN

Die Semantik von GridWorldSim kann durch neue Regeln und neue Arten von Objekten und Agenten erweitert werden, welche durch eigene Java-Klassen realisiert werden. Die Verwendung solcher Regeln, Objekte und Agenten in einer Gridwelt-Spezifikation wird in Kapitel 7.3.2 erläutert.

Es gilt:

- Eine eigene Regelklasse implementiert das Interface `net.tittel.gridworldsim.statetrans.StateTransRule`.
- Eine eigene Objektklasse erweitert die Klasse `net.tittel.gridworldsim.GridObject`.
- Eine eigene Agentenklasse erweitert die Klasse `net.tittel.gridworldsim.Agent`.

Die Erweiterung von GridWorldSim soll an folgendem Beispiel demonstriert werden: Eine neue Agentenart „Geldagent“ soll in der Lage sein, Geld mit sich zu führen. Dies ließe sich zwar über Properties realisieren, wäre jedoch wenig komfortabel. Agenten dieser Agentenart können Geld an Geldautomaten abheben und natürlich auch ausgeben (wobei das Ausgeben nicht Teil dieses Beispiels ist). Jeder Geldautomat führt dazu Bankkonten für Agenten, wobei jedes Bankkonto einen bestimmten Geldbetrag enthält. Die Bankkonten unterschiedlicher Geldautomaten sind voneinander unabhängig. Ein Agent kann genau dann eine bestimmte Summe von einem Geldautomaten abheben, wenn er bei diesem Geldautomaten ein Bankkonto besitzt, welches ausreichend gedeckt ist, und wenn er sich in der gleichen Zelle wie der Geldautomat befindet.

Für eine detaillierte Erklärung der beteiligten GridWorldSim-Klassen sei auf die Javadoc-Dokumentation verwiesen, die sich auf der beiliegenden CD-ROM befindet. Zum Verständnis des Beispiel ist diese jedoch nicht erforderlich.

Konzeptionell ließe sich dieses Beispiel durch die Einführung neuer Typen von Fakten sowie eines neuen Typs von Einfluss und einer neuen Zustandsüberführungsregel spezifizieren. Für die neuen Typen von Fakten gilt:

- **MoneyAgent**(a): a ist ein Geldagent.
- **ATM**(o): o ist ein Geldautomat.
- **Money**(a, k): a verfügt über den Geldbetrag k.

- **Account**(o, a, b): a besitzt ein Konto beim Geldautomaten o mit einem Kontostand von b.

Sei **Withdraw**(a, o, l) eine neue Aktionsanforderung, die besagt, dass Agent a beim Objekt o den Betrag l abheben möchte. Dann ist eine geeignete Zustandsüberführungsregel gegeben durch:

ATMTakeOut	Withdraw(a, o, l)
Pre:	$\{\text{Loc}(x, y, a),$ $\text{Loc}(x, y, o),$ $\text{MoneyAgent}(a),$ $\text{ATM}(o),$ $\text{Money}(a, k),$ $\text{Account}(o, a, b),$ $l \leq b\}$
Prob:	1
Post:	$\{\neg\text{Account}(o, a, b),$ $\neg\text{Money}(a, k),$ $\text{Account}(o, a, b - l),$ $\text{Money}(a, k + l)\}$

Implementatorisch besteht die Erweiterung aus drei Java-Klassen:

- `MoneyAgent` realisiert Geldagenten.
- `ATMObject` realisiert Geldautomaten.
- `ATMRule` ist eine Zustandsüberführungsregel, welche die Bedienung von Geldautomaten realisiert.

7.6.1 *Agent*

Spezifikation 27 zeigt den Quellcode der Klasse `MoneyAgent`. Wie in Zeile 1 zu sehen erweitert diese Klasse die Agentenstandardklasse `Agent` und führt die neue Instanzvariable `money` ein, welche das aktuell mitgeführte Geld repräsentiert (Zeile 3 und 7). Über die Methode `addMoney` kann der Agent neues Geld aufnehmen (Zeile 10–12) und über die Methode `spendMoney` Geld ausgeben, sofern er über genügend Geld verfügt (Zeile 14–21).

Für einen `MoneyAgent` soll gelten, dass er allgemein als `MoneyAgent` wahrgenommen werden kann. Ebenso soll es mit dem Beobachter-Client und für den Agenten selbst möglich sein, die Menge des aktuell mitgeführten Geldes festzustellen, wobei diese Information für andere Agenten nicht verfügbar sein soll. Da

```
public class MoneyAgent extends Agent {

  private int money;

  public MoneyAgent() {
    super();
    money = 0;
  }

  public void addMoney(int amount) {
    money += amount;
  }

  public boolean spendMoney(int amount) {
    if (money >= amount) {
      money -= amount;
      return true;
    } else {
      return false;
    }
  }

  @Override
  public Collection<String> getProperties(boolean
          visibleOnly) {
    Collection<String> props = new HashSet<String>();
    props.addAll(super.getProperties(visibleOnly));
    props.add("MoneyAgent");
    if (!visibleOnly) {
      props.add("Money("+money+")");
    }
      return props;
    }

  @Override
  public void addParameter(GridObjectParameter
          parameter) {
    super.addParameter(parameter);
    if (parameter.getName().equals("money")) {
      money = Integer.valueOf(parameter.getValue());
    }
  }
}
```

Spezifikation 27: Quellcode der Klasse MoneyAgent

bei GridWorldSim über die Standardfunktionalität hinausgehende beobachtbare Eigenschaften durch Properties realisiert sind, wird beginnend ab Zeile 24 die Methode `getProperties` überschrieben. Der Parameter `visibleOnly` dieser Methode gibt an, ob nur die allgemein sichtbaren Properties oder alle Properties zurückgegeben werden sollen.

Zur Erweiterung der Properties wird eine neue `Collection` erzeugt, welcher die Menge der Properties aus der Superklasse hinzugefügt wird (Zeile 26–27)[3]. Der neuen `Collection` wird in jedem Fall die Information hinzugefügt, dass es sich um einen `MoneyAgent` handelt (Zeile 28). Sollen auch nicht allgemein verfügbare Properties zurückgegeben werden, wird die Menge der Properties zudem um die Information Money(x) erweitert, wobei x der Wert des aktuell vom Agenten mitgeführten Geldes sei (Zeile 29–31).

Es soll ferner möglich sein, dass ein `MoneyAgent` schon von Beginn an über Geld verfügt. Dieser initiale Geldbetrag soll für unterschiedliche `MoneyAgent`-Instanzen individuell in der Spezifikationsdatei spezifizierbar sein. Dazu soll die Klasse einen Parameter (siehe Kapitel 7.2.8) namens „money“ unterstützen. Wird im Rahmen der Initialisierung des Agenten ein Parameter mit dem Namen „money“ übergeben, so wird der zugehörige Wert für die Instanzvariable `money` übernommen (Zeile 35–42).

7.6.2 *Objekt*

Spezifikation 28 zeigt den Quellcode der Klasse `ATMObject`, welche die allgemeinen Standardobjektklasse `GridObject` erweitert (Zeile 1). Bankkonten sind in dieser Klasse durch eine `Map` namens `bankAccounts` realisiert, welche Agentennamen auf Kontostände abbildet (Zeile 3 und 7).

Parameternamen (außer „property“) interpretiert die Klasse als Agentennamen und der zugehörige Parameterwert entspricht dem Guthaben auf dem Bankkonto des jeweiligen Agenten. Übergebene Parameter können somit der `Map` einfach hinzugefügt werden.[4] (Zeile 10–18).

Für die zurückzugebende Menge an Properties gilt für `ATMObject` Ähnliches wie für `MoneyAgent`. Jeder Agent soll wahrnehmen können, dass es sich bei einem `ATMObject` um einen Geldautomaten handelt. Dashalb wird die Property `ATM` der zurückgegebenen Property-Menge immer hinzugefügt (Zeile 25).

---

3 Es ist darauf zu achten, die neuen Properties nicht der von der Superklasse erhaltenen `Collection` hinzuzufügen.

4 Der Beispiel-Code berücksichtigt an dieser Stelle nicht, dass keine Garantie besteht, dass sich der Parameterwert tatsächlich in einen Zahlenwert umwandeln lässt.

```java
public class ATMObject extends GridObject {

  private Map<String, Integer> bankAccounts;

  public ATMObject() {
    super();
    bankAccounts = new HashMap<String, Integer>();
  }

  @Override
  public void addParameter(GridObjectParameter
          parameter) {
    super.addParameter(parameter);
    if (!parameter.getName().equals("property")) {
      bankAccounts.put(parameter.getName(),
              Integer.valueOf(parameter.getValue()));
    }
  }

  @Override
  public Collection<String> getProperties(boolean
          visibleOnly) {
    Collection<String> props = new HashSet<String>();
    props.addAll(super.getProperties(visibleOnly));
    props.add("ATM");
    if (!visibleOnly) {
      for (Iterator<String> i = bankAccounts.keySet()
              .iterator(); i.hasNext();) {
        String agent = i.next();
        props.add("Account("+agent+","
                +bankAccounts.get(agent)+")");
      }
    }
    return props;
  }

  public boolean withdrawMoney(Agent agent,
          int amount) {
    String agentName = agent.getName();
    Integer balance = bankAccounts.get(agentName);
    if (balance != null && balance.intValue()
            >= amount) {
      bankAccounts.put(agentName, new Integer
              (balance.intValue() - amount));
      return true;
    } else {
      return false;
    }
  }
}
```

Spezifikation 28: Quellcode der Klasse ATMObject

Mit dem Beobachter-Client soll es zudem möglich sein, sämtliche Bankkonten samt ihrem Kontostand in Erfahrung zu bringen, wobei Agenten diese Information nicht erhalten sollen. Daher werden Properties der Form (Agentenname, Kontostand) nur zurückgegeben, wenn `visibleOnly` nicht gesetzt ist. (Zeile 26–32).

Das Abheben von Geld geschieht durch die Methode `withdrawMoney` (Zeile 37–49). Geld kann genau dann abgehoben werden, wenn das zugehörige Konto ausreichend gedeckt ist. Ist dies der Fall, wird der Kontostand um den abgehobenen Betrag reduziert. Der boolsche Rückgabewert der Methode teilt mit, ob das Abheben erfolgreich war oder nicht.

7.6.3 *Regel*

Die Klasse `ATMRule` (dargestellt in Spezifikation 29) realisiert die Zustandsüberführungsregel zum Abheben von Geld an einem Geldautomaten und implementiert das Interface `StateTransRule` (Zeile 1). Das Interface besteht aus drei Methoden:

- `isTriggered` bekommt von der Server-Klasse eine Aktionsanforderung übergeben und eine Regel teilt daraufhin mit, ob sie von der vorliegenden Aktionsanforderung ausgelöst wird oder nicht. Dies entspricht dem Auslöser einer Zustandsüberführungsregel gemäß Kapitel 4.7.1.
- `doTransition` führt die Zustandsüberführung durch, indem die Aktionsanforderung auf die Gridwelt angewandt wird. Dies entspricht den Ausführungsbedingungen und Nachbedingungen von Zustandsüberführungsregeln gemäß Kapitel 4.7.1.
- Mittels `setParameter` kann eine Regel über die Spezifikationsdatei konfiguriert werden.

Die Regel `ATMRule` soll von einer Aktionsanforderung vom Typ „withdrawMoney“ ausgelöst werden. Ob dies für eine vorliegende Aktionsanforderung gilt, wird in den Zeilen 3 bis 7 überprüft.

Eine Aktionsanforderung in Form eines `ActionRequest`-Java-Objekts besteht aus einem Agenten und einem `ActionRequestOp`-Objekt. Beim Agenten handelt es sich bereits um das `Agent`-Objekt, welches diesen Agenten repräsentiert. Das `ActionRequestOp`-Objekt kapselt hingegen die per XML-Dokument erhaltenen Wünsche eines Agenten, eine Aktion auszuführen und besteht aus einer Menge von Strings. Es muss daher zu Beginn der Zustandsüberführung in der Methode `doTransition` zunächst basierend auf dem String, der den beteiligten Geldautomaten beschreibt, das zugehörige Java-Objekt aus der `GridWorld`-Datenstruktur abgefragt werden. Dies geschieht in den Zeilen 13

```
public class ATMRule implements StateTransRule {

  @Override
  public boolean isTriggered(ActionRequest request) {
    return request.getRequestOp().getType()
            .equals("withdrawMoney");
  }

  @Override
  public void doTransition(GridWorld gridWorld,
          ActionRequest request) {
    Agent agent = request.getAgent();
    GridObject atmObject = gridWorld
            .getGridObjectByName(request.getRequestOp()
            .getParameterValue("atm"));
    int amount = Integer.valueOf(request.getRequestOp()
            .getParameterValue("amount"));

    if (atmObject == null
            || !(atmObject instanceof ATMObject)
            || !(agent instanceof MoneyAgent)) {
      return;
    }

    if (!gridWorld.getGridObjectLocation(atmObject)
            .equals(gridWorld.getGridObjectLocation
            (agent))) {
      return;
    }

    ATMObject atm = (ATMObject) atmObject;
    MoneyAgent mAgent = (MoneyAgent) agent;

    boolean success = atm.withdrawMoney(agent, amount);
    if (success) {
      mAgent.addMoney(amount);
    }
  }

  @Override
  public void setParameter(String parameter,
          String value) {}
}
```

Spezifikation 29: Quellcode der Klasse ATMRule

bis 15. Ebenso wird der empfangene String, der den abzuhebenden Geldbetrag beschreibt, in einen Zahlenwert gewandelt (Zeile 16–17).

In den Zeilen 19–23 wird überprüft, ob das Objekt mit dem vom Agenten spezifizierten Namen tatsächlich exisitiert und ob es sich bei diesem Objekt um einen Geldautomaten und beim anfragenden Agenten um einen `MoneyAgent` handelt. Falls nicht, wird die Verarbeitung abgebrochen.

Damit die Aktionsanforderung Erfolg hat, muss gelten, dass sich Agent und Geldautomat in der gleichen Zelle befinden. Dies wird in den Zeilen 25 bis 29 überprüft, anderenfalls wird die Verarbeitung abgebrochen.

Die Entscheidung, ob der Agent über genügend Geld auf seinem Bankkonto verfügt, wird dem `ATMObject` überlassen. Erlaubt das `ATMObject` das Abheben des angeforderten Betrags, so wird dem Agenten dieser Betrag hinzugefügt (Zeile 34–37). Damit ist die Zustandsüberführung abgeschlossen.

Da die Regel keine konfigurierbaren Parameter besitzt, bleibt der Rumpf der Methode `setParameters` leer (Zeile 40–42).

7.7 SERVER-API

Bei der mit *Server-API* bezeichneten Komponente von GridWorldSim handelt es sich um eine Java-Bibliothek zur Kommunikation mit dem Server und zur Bereitstellung von Datenstrukturen für erhaltene Wahrnehmungen. Die Server-API dient dazu, dass in Java implementierte Agenten den GridWorldSim-Server einfach nutzen können, ohne dass dazu das Kommunikationsprotokoll einschließlich der Erzeugung von XML-Dateien sowie des Parsens erhaltener XML-Dateien in geeignete Datenstrukturen selbst implementiert werden müsste. Nicht in Java implementierte Agenten können die Server-API u. U. durch entsprechende *language bindings* nutzen, sofern diese verfügbar sind. Falls nicht, stellt die Server-API als Referenzimplementierung dennoch eine nützliche Vorlage zur Erstellung einer eigenen Implementierung in einer anderen Sprache dar.

Die Server-API ist im Java-Paket `net.tittel.gridworldsim.serverapi` und dessen Unterpaket `perceptions` enthalten. Das Unterpaket enthält dabei die Datenstrukturen, welche zur Repräsentation erhaltener Wahrnehmungen Verwendung finden.

Bevor eine Kommunikation mit dem Server stattfinden kann, muss zunächst eine Verbindung zu diesem aufgebaut werden. Dazu dient die Klasse `ServerConnection`, deren Konstruktor der Servername und Port für die zu erstellende Verbindung übergeben wird. Mittels der Methode `connect()` kann eine neue Verbindung zum Server aufgebaut werden, wobei die Methode ein

Objekt der Klasse `AgentRequestSender` zurückgibt, welches dem Versand von Aktionsanforderungen dient.

Der Aufruf der Methode `connect()` startet einen neuen Thread, welcher eingehende Wahrnehmungen vom Server empfängt. Die Schnittstelle zwischen der Implementierung eines eigenen Agenten oder Beobachter-Clients und diesem Thread findet über eine Warteschlange des Typs `LinkedBlockingQueue` statt. Objekte dieser Klasse besitzen zwei vorteilhafte Eigenschaften:

1. Warteschlangen dieses Typs erlauben blockierenden Zugriff (wahlweise auch mit Zeitschranke), d. h. es besteht für die eigene Implementierung eine einfache Möglichkeit, auf neue Wahrnehmungen zu warten.

2. Warteschlangen dieses Typs sind thread-sicher.

Der Thread soll sowohl über neue Wahrnehmungen als auch über einen Verbindungsabbruch berichten können. Beide Typen von Ereignissen werden in Objekten der Klasse `ClientQueueItem` gekapselt. Ein `ClientQueueItem` beinhaltet entweder eine empfangene Wahrnehmung in Form einer `GridWorldPerception` oder benachrichtigt über einen Verbindungsabbruch. Bei den vom Thread der Warteschlange hinzugefügten Objekten handelt es sich somit um Objekte vom Typ `ClientQueueItem`. Die Warteschlange der eingehenden Ereignisse wird vom `ServerConnection`-Objekt über die Methode `getInItems()` abgerufen. Nach Verbindungsaufbau erfolgt die Anmeldung beim Server über eine der `login`-Methoden des `ActionRequestSender`-Objekts.

Für die Klassen, welche zur Erzeugung von Objekten dienen, welche die Wahrnehmung repräsentieren, gilt:

- `GridWorldPerception` repräsentiert die Wahrnehmung der Gridwelt durch den Agenten in ihrer Ganzheit.

- Objekte vom Typ `GridCellPerception` sind Teil einer `GridWorldPerception` und repräsentieren die Wahrnehmung einer Gridzelle samt den enthaltenen Objekten und Hindernissen.

- Agenten und Objekte mitsamt ihren beobachtbaren Eigenschaften werden von Objekten der Klassen `AgentPerception` und `GridObjectPerception` repräsentiert. Sie sind entweder Teil einer `GridCellPerception` oder können im Falle von `GridObjectPerception`-Objekten auch Teil einer anderen `AgentPerception` oder `GridObjectPerception` sein.

- Objekte vom Typ `MessagePerception` repräsentieren Nachrichten und sind Teil einer `GridWorldPerception`.

```java
ServerConnection sc =
        new ServerConnection("localhost", 7777);
ActionRequestSender requester = sc.connect();
requester.login("Apollo");

LinkedBlockingQueue<ClientQueueItem> inItems =
        sc.getInItems();
ClientQueueItem item = inItems.take();

if (item.getType()==ClientQueueItem.PERCEPTION) {
  GridWorldPerception percept = item.getGwPercept();
  int myX = percept.getMyAgentXLocation();
  int myY = percept.getMyAgentYLocation();
  AgentPerception myAgent = percept.getMyAgent();

  if (percept.getGridCellPerceptions()[myX+1][myY]
          .isAccessible(myAgent.getCapNeed())) {
    requester.moveAgent(Constants.EAST);
  }
}
```

Spezifikation 30: Beispiel zur Nutzung der Server-API

Für eine genaue Beschreibung der Klassen sei auf die Javadoc-Dokumentation verwiesen, welche sich auf der beiliegenden CD-ROM befindet.

Spezifikation 30 zeigt ein Beispiel, in welchem sich ein Agent mit dem Namen „Apollo“ ohne Passwort mit dem Server `localhost` auf Port 7777 verbindet und genau dann einmalig eine Bewegung Richtung Osten auszuführen versucht, wenn für die aktuelle Wahrnehmung gilt, dass die östlich von ihm gelegene Zelle keine Mauer und keinen Graben beinhaltet und über genügend freie Kapazität für seinen aktuellen Kapazitätsbedarf verfügt.

Der Verbindungsaufbau und das Anmelden beim Server finden in den Zeilen 1 bis 4 statt. In den Zeilen 6 und 7 wird die Warteschlange der Verbindung abgerufen, aus welcher in Zeile 8 das erste `ClientQueueItem` entnommen wird. Da die Methode `take()` blockiert, wird an dieser Stelle so lange gewartet, bis ein Element in der Warteschlange verfügbar ist.

Für den Fall, dass die Anmeldung beim Server erfolgreich war und die Verbindung nicht anderweitig unterbrochen wurde, handelt es sich bei diesem `ClientQueueItem` um die erste Wahrnehmung des Agenten. Ob dies wirklich der Fall ist, wird in Zeile 10 überprüft.

In den Zeilen 11 bis 14 wird die aktuelle Wahrnehmung dem `ClientQueueItem` entnommen und der Agent erfragt seine eigene Position sowie seine Selbstwahrnehmung.

Die Überprüfung, ob die östlich gelegene Zelle im aktuell wahrgenommenen Zustand über ausreichend Kapazität zur Aufnahme des Agenten verfügt und darüber hinaus keine Mauer und keinen Graben enthält, findet in den Zeilen 16 und 17 statt. Ist dies der Fall, wird die Aktionsanforderung zur Bewegung nach Osten in Zeile 18 versendet.

7.8 server-implementierung

Im Folgenden wird das allgemeine Implementierungskonzept des GridWorldSim-Servers beschrieben. Dabei werden Objekte, die Teil der Gridwelt sind, zur Unterscheidung von Java-Objekten als *Gridwelt-Objekte* bezeichnet, um Mehrdeutigkeiten zu vermeiden.

Der GridWorldSim-Server besteht aus den Klassen des Pakets `net.tittel.gridworldsim.server` und dessen Unterpaketen, wobei `Server` die Hauptklasse des Servers darstellt. Nach dem Starten des Servers werden zwei neue Threads erzeugt und gestartet: Ein `AgentConnectorThread` und ein `ObserverConnectorThread`. Diese Threads sind für die Annahme neuer eingehender Verbindungen von Agenten bzw. Beobacher-Clients verantwortlich. Trifft eine solche Verbindung ein, wird für diese ein `AgentThread` bzw. `ObserverThread` erzeugt und gestartet, welcher die eingehenden Daten des verbundenen Agenten oder Beobachter-Clients annimmt.

7.8.1 *Thread-Synchronisation*

Alle Threads einschließlich des Server-Hauptthreads teilen sich – wie schon bei der Server-API – eine Warteschlange vom Typ `LinkedBlockingQueue`. Kommunikation zwischen Threads findet ausschließlich über diese Warteschlange statt, wobei eine Erzeuger-Verbraucher-Architektur vorliegt: Die Threads vom Typ `AgentThread` und `ObserverThread` agieren als Erzeuger, während der Server-Hauptthread die erzeugten Warteschlangeninhalte verbraucht. Für `AgentConnectorThread` und `ObserverConnectorThread` besteht hingegen kein Bedarf für Informationsaustausch mit bereits gestarteten Threads. Wie in Kapitel 7.7 erläutert ist eine `LinkedBlockingQueue` blockierend und thread-sicher, so dass sich das Problem der Thread-Sicherheit nicht stellt.

Inhalt der gemeinsamen `LinkedBlockingQueue` sind Objekte vom Typ `QueueItem`. Ein `QueueItem`-Objekt dient dazu, für den Server-Hauptthread relevante Ereignisse, die in einem `Agent-`

`Thread` oder `ObserverThread` entstehen, zu kapseln. Ein `QueueItem` kann fünf verschiedene Arten von Ereignissen kapseln:

1. Aktionsanforderungen von Agenten: In diesem Fall enthält das `QueueItem` ein Objekt der Klasse `ActionRequestOpItem`. Ein solches Objekt kapselt den Typ und die Parameter der Aktionsanforderung sowie die `SocketConnection`, die den zum Client gehörigen TCP-Socket beinhaltet. Der Server-Hauptthread verwaltet die Zuordnung von Agenten zu Objekten vom Typ `SocketConnection` und kann damit feststellen, von welchem Agenten die Aktionsanforderung stammt. Durch dieses Vorgehen wird vermieden, dass Agenten Aktionsanforderungen im Namen anderer Agenten stellen können.

2. Verbindungsabbrüche: Tritt ein Verbindungsabbruch auf, beinhaltet das der Warteschlange hinzugefügte `QueueItem` ein `DisconnectedItem`. Dieses enthält die Details des Verbindungsabbruchs, so dass der Server den Verbindungsabbruch geeignet behandeln kann.

3. Neue Verbindung von Agenten: Der Verbindungsaufbau eines Agenten zum Server wird nicht direkt behandelt, es wird jedoch für den Anmeldeversuch des Agenten ein `QueueItem` erzeugt, welches die Anmeldeinformation als `ActionRequestOpItem` beinhaltet.

4. Neue Verbindung von Beobachter-Clients: Auch der Verbindungsaufbau eines Beobachter-Clients wird nicht direkt behandelt. Sobald der Beobachter-Client sich anmeldet, wird jedoch ein `QueueItem` erzeugt, welches ein `NewObserverItem` beinhaltet.

5. Es wurde eine manuelle Zustandsüberführung ausgelöst.

Punkt 5 kann dabei nicht nur durch Beobachter-Clients entstehen, sondern auch durch einen `Timer`-Thread, welcher abhängig vom Ausführungsmodus nach Ablauf des Timers eine Zustandsüberführung auslöst.

7.8.2 *Erzeugung von Datenstrukturen*

Die Erzeugung von Datenstrukturen wie `GridWorld` (repräsentiert die gesamte Gridwelt), `GridCell` (repräsentiert eine Gridzelle), `GridObject` (repräsentiert ein Gridwelt-Objekt) oder `Agent` (repräsentiert einen Agenten) geschieht durch *Factory*-Objekte. Ein Factory-Objekt wird dabei mit den notwendigen Parametern konfiguriert, um bestimmte Arten von Objekten erzeugen zu können. Die dazu gehörigen Klassen befinden sich im Paket `net.`

`tittel.gridworldsim.server.factories`. Zur Erzeugung von Datenstrukturen dienen die folgenden Arten von Factory-Objekten:

- `ConfigFactory`: Ein Objekt dieses Typs wird unter Angabe des Pfades der Spezifikationsdatei konstruiert. Es ermöglicht die Erzeugung der Menge zulässiger Anmeldedaten für Beobachter-Clients und der Portnummern, auf denen der Server auf eingehende Verbindungen von Agenten und Beobachter-Clients wartet, sowie die Erzeugung der für die Konfiguration der Debug-Ausgaben relevanten Parameter. Vor allem jedoch wird die `GridWorld`-Datenstruktur erzeugt, welche die Gridwelt im Startzustand S_0 repräsentiert. Zu diesem Zweck wird u. a. für jeden in der Spezifikationsdatei spezifizierten Gridwelt-Objekttypen eine `GridObjectFactory` konfiguriert und in einer `Map` vorgehalten, welche die Namen der Gridwelt-Objekttypen aus der Spezifikationsdatei auf diejenige `GridObjectFactory` abbildet, die zur Erzeugung von `GridObject`-Objekten dieses Typs dient.

- `GridObjectFactory`: Ein Objekt dieses Typs dient der Erzeugung von Gridwelt-Objekten eines in der Spezifikationsdatei spezifizierten Typs und wird (durch eine `ConfigFactory`) mit den Vorgabewerten (wie z. B. dem Kapazitätsbedarf) und dem Namen des Gridwelt-Objekttyps sowie der `GridWorld`-Datenstruktur des Servers konstruiert. Darüber hinaus ist es möglich, ein `GridObjectFactory`-Objekt mit dem Namen der Java-Klasse zu konfigurieren, die für diesen Gridwelt-Objekttyp Verwendung finden soll. Eine `GridObjectFactory` hat keinen Zugriff auf die Spezifikationsdatei und erzeugt Gridwelt-Objekte eines Typs ausschließlich gemäß ihrer Konfiguration.

- `AgentFactory`: Ein Objekt vom Typ `AgentFactory` dient der Erzeugung von `Agent`-Objekten. Eine `AgentFactory` besitzt Zugriff auf die Spezifikationsdatei und auf die `Map`, welche die Namen von Gridwelt-Objekttypen auf Objekte vom Typ `GridObjectFactory` abbildet. Zur Erzeugung eines neuen `Agent`-Objekts nimmt die `AgentFactory` die von einem Agenten empfangenen Anmeldedaten entgegen. Für diese Anmeldedaten versucht sie, einen zugehörigen Eintrag in der Spezifikationsdatei zu finden. Gelingt dies, wird für den Agenten ein `Agent`-Objekt erzeugt. Zur Erzeugung der Gridwelt-Objekte, die sich im Inventar des Agenten befinden, wird die für den jeweiligen Gridwelt-Objekttypen zuständige `GridObjectFactory` verwendet.

Die Verwendung dieser Factory-Klassen bietet den Vorteil, dass zum einen der Code zum Parsen der Spezifikationsdatei und zur Erstellung von Datenstrukturen in dafür spezialisierte Klassen ausgelagert wird, zum anderen, dass im Gegensatz zur Auslagerung dieser Funktionalität in die Methoden spezialisierter Nicht-Factory-Klassen ein einmal konstruiertes und konfiguriertes Factory-Objekt wiederverwendet werden kann, ohne dass bei jedem Methodenaufruf sämtliche relevanten Parameter erneut übergeben werden müssten.

7.8.3 *Erzeugung von Wahrnehmungen*

Auch die Erzeugung von Wahrnehmungen wird durch Factory-Objekte realisiert. Dazu finden Factory-Objekte vom Typ `OutputPerceptionFactory` Verwendung. Für den Fall, dass die Beobachtungen für einen Agenten konstruiert werden sollen, wird die Factory mit dem `GridWorld`-Objekt der aktuellen Gridwelt und dem `Agent`-Objekt des Agenten, für den die Beobachtungen erzeugt werden sollen, konstruiert. Sollen Beobachtungen für einen Beobachter-Client erzeugt werden, wird die Factory hingegen mit dem `GridWorld`-Objekt der aktuellen Gridwelt und der Menge der internen Zustände aller Agenten konstruiert.

Das an einen Client zu versendende XML-Dokument, das dessen Wahrnehmungen im aktuellen Zustand beinhaltet, kann nach Konstruktion des `OutputPerceptionFactory`-Objekts von diesem über die Methode `getDocument` abgefragt werden.

7.8.4 *Kontrollfluss der Hauptklasse*

Der Kontrollfluss der Hauptklasse `Server` stellt sich wie im Folgenden beschrieben dar. Es gilt dabei, dass bei jeder Zustandsüberführung alle eventuell noch unbearbeiteten Aktionsanforderungen, Agenten-Anmeldungen und Agenten-Deregistrierungen vorgenommen werden und im Anschluss an die Zustandsüberführung jeder verbundene Agent und Beobachter-Client seine Wahrnehmung des neuen Zustandes mitgeteilt bekommt. Ebenso wird abhängig vom Ausführungsmodus ggf. ein neuer `Timer` gestartet oder es wird ggf. eine Menge aller aktiven Agenten erzeugt, aus der ein Agent wieder entfernt wird, wenn eine Aktionsanforderung von ihm vorliegt.

1. Nach dem Start des Servers erfolgt zunächst die Erzeugung einer `ConfigFactory`. Von dieser werden zum Starten des Servers relevante Informationen (wie z. B. die Portnummern für eingehende Verbindungen) abgefragt. Zudem wird die Warteschlange zur Thread-Synchronisation erzeugt. Danach

erfolgt das Erzeugen und Starten der Threads, die eingehende Verbindungen annehmen.

2. Als nächstes wird die `GridWorld`-Datenstruktur, welche die Gridwelt im initialen Umgebungszustand repräsentiert, von der `ConfigFactory` abgerufen und es wird eine Initialisierung diverser Hilfsdatenstrukturen vorgenommen. Zudem wird der Ausführungsmodus abgefragt und je nach Ausführungsmodus ein `Timer` gestartet. Danach ist der Server konfiguriert und einsatzbereit.

3. Im Folgenden wartet der Server auf neue Elemente in der Warteschlange. Wird von einem der Threads vom Typ `AgentThread`, `ObserverThread` oder `Timer` ein neues Element der Warteschlange hinzugefügt, wird dieses wie im Folgenden geschildert bearbeitet. Nach der jeweiligen Bearbeitung wartet der Server erneut auf neue Elemente in der Warteschlange.

   - Handelt es sich bei dem neuen Element in der Warteschlange um die Aktionsanforderung eines Agenten (einschließlich der Anmeldungen von Agenten beim Server) wird abhängig vom Ausführungsmodus überprüft, ob diese Aktionsanforderung eine Zustandsüberführung verursacht. Falls ja, wird die Zustandsüberführung durchgeführt. Anderenfalls wird die Aktionsanforderung in einer Hilfsdatenstruktur zur späteren Verwendung gespeichert und der Agent wird je nach Ausführungsmodus aus der Menge der Agenten, auf deren Aktionsanforderung noch gewartet wird, entfernt. Sofern bereits Agenten in der Umgebung existieren und sofern beim gewählten Ausführungsmodus nicht jede Aktionsanforderung eine Zustandsüberführung nach sich zieht, kann es deshalb vorkommen, dass ein neuer Agent nicht sofort Teil der Gridwelt wird und bis zur nächsten Zustandsüberführung warten muss, bis er eine Wahrnehmung mitgeteilt bekommt.
   - Liegt die Anmeldung eines neuen Beobachter-Clients vor, wird dieser in einer Datenstruktur registriert und erhält die Wahrnehmung für den aktuellen Zustand mitgeteilt. Dabei handelt es sich immer um einen Zustand S_t und nicht um einen der Zwischenzustände gemäß Kapitel 4.6, da Zustandsüberführungen immer vollständig in einer Sequenz durchgeführt werden und zu einem Zeitpunkt, in dem auf neue Elemente in der Warteschlange gewartet wird, die Datenstruktur somit nie einen Zwischenzustand repräsentiert.

- Handelt es sich bei dem neuen Element in der Warteschlange um die Benachrichtung über einen Verbindungabbruch, wird für den Fall, dass die Verbindung zu einem Agenten nicht mehr besteht, überprüft, ob abhängig vom gewählten Ausführungsmodus eine Zustandsüberführung zur Deregistrierung des Agenten durchgeführt werden kann. Anderenfalls wird der Agent zur Deregistrierung im Rahmen der nächsten Zustandsüberführung vorgemerkt. Wartet der Server beim gewählten Ausführungsmodus darauf, dass von jedem Agenten eine Aktionsanforderung vorliegt, wird der Agent zudem aus der Menge der Agenten, auf deren Aktionsanforderung noch gewartet wird, entfernt. Ist die Verbindung zu einem Beobachter-Client abgebrochen, kann dieser sofort aus der Menge der registrierten Beobachter-Clients entfernt werden.
- Handelt es sich beim Element in der Warteschlange um die sofortige Auslösung einer Zustandsüberführung durch einen `Timer` oder Beobachter-Client, wird die Zustandsüberführung ausgelöst.

Für eine detaillierte Beschreibung des Servers sei an die beiliegende CD-ROM verwiesen, welche die Javadoc-Dokumentation und den Quellcode des Servers enthält.

8 BEWERTUNG UND AUSBLICK

Im Folgenden soll das ihm Rahmen dieser Arbeit konzipierte und implementierte Framework mit den in Kapitel 2.2.1 und 2.2.2 vorgestellten Ansätzen verglichen werden. Abschließend erfolgt ein Fazit und ein Ausblick auf mögliche Erweiterungen.

8.1 VERGLEICH MIT ELMS

ELMS bietet gegenüber GridWorldSim den Vorteil, dass eigene Objekttypen durch Ausdrücke direkt in der Umgebungsspezifikation definiert werden können, ohne dabei wie im Falle von GridWorldSim eigene Java-Klassen implementieren zu müssen. In Verbindung mit einer nicht primär unter dem Aspekt der menschlichen Lesbarkeit entwickelten Sprachsyntax resultiert diese Fähigkeit jedoch leicht in großen und unübersichtlichen Umgebungsspezifikationen, selbst wenn nur einfache Aktionen wie die Bewegungsmöglichkeiten eines Agenten auf dem Grid beschrieben werden sollen. Die Implementierung eigener Agenten-, Objekt- und Regelklassen in Java bei GridWorldSim ist zwar nicht deklarativ, in komplexen Fällen jedoch einfacher und übersichtlicher. Zudem sind die für Ausdrücke zur Verfügung stehenden ELMS-Operatoren nicht mächtig genug, um komplexe Funktionalität zu realisieren.

Die vom ELMS-Interpreter unterstützte Handhabung der Ausführungsreihenfolge (siehe Kapitel 2.2.1.2) und das Fehlen von Authentifikation bei der Registrierung von Agenten verhindern einen sinnvollen Einsatz des ELMS-Interpreters für Multiagentensysteme, bei dem Agenten im Wettbewerb miteinander stehen.

Darüberhinaus kann ELMS nicht der Vermittlung von Nachrichten zwischen Agenten dienen, wie dies zur Einschränkung der Kommunikation zwischen Agenten erforderlich wäre. Auch die im Vergleich zu GridWorldSim fehlende Unterstützung proaktiver Umgebungen ist ein Nachteil von ELMS.

ELMS unterstützt im Vergleich zu GridWorldSim eine flexiblere Spezifikation, unter welchen Umständen ein Agent Dinge in der Welt wahrnehmen kann. So könnte es bspw. in einer Zelle Objekte geben, die ein Agent nur dann wahrnehmen kann, wenn er eine Eigenschaft besitzt, die ihm die Wahrnehmung besonders kleiner Objekte ermöglicht. Dies ließe sich in ELMS spezifizieren, bei GridWorldSim jedoch nicht, da die Wahrnehmung bei GridWorldSim fest davon abhängt, welche Sichtweite der Agent

besitzt, in welcher Gridzelle er sich befindet und welche sichtbehindernden Hindernisse in der Welt existieren. Wenngleich es bei GridWorldSim im Gegensatz zu ELMS nicht möglich ist, die Wahrnehmungsbestimmung direkt zu spezifizieren, so lässt sich zumindest die Sichtweite eines Agenten dynamisch modifizieren (z. B. durch das Erhöhen der Sichtweite eines Agenten, wenn sich in dessen Inventar ein Objekt vom Typ „Brille“ befindet). Auf der anderen Seite ist es bei ELMS nicht möglich, komplexe Wahrnehmungsberechnungen durchzuführen, so wie dies bei GridWorldSim basierend auf diskreten Sichtlinien geschieht.

Zusammengefasst lässt sich feststellen, dass sich mit ELMS einige Arten von Umgebungen spezifizieren lassen, die mit GridWorldSim nicht spezifizierbar sind. Dies gilt vor allem für den Bereich der Wahrnehmungsbestimmung, da GridWorldSim diesbezüglich keine flexible Spezifikation erlaubt. Ebenso lassen sich mit GridWorldSim Umgebungen (auch ohne Verwendung eigener Java-Klassen) spezifizieren, die mit ELMS nicht realisierbar sind. Dazu zählen insbesondere Umgebungen, die komplexe, von GridWorldSim explizit unterstützte Funktionalität beinhalten, wie z. B. die Einschränkung von Wahrnehmung unter Verwendung diskreter Sichtlinien und Hindernissen sowie Proaktivität.

Beim Vergleich zwischen ELMS und GridWorldSim ist zu berücksichtigen, dass ELMS als Komponente des MAS-SOC-Projekts und nicht in Hinblick auf die Erfüllung der Ziele dieser Arbeit entwickelt wird. Obwohl ELMS und GridWorldSim beide der Realisierung von Umgebungen für Multiagentensysteme dienen, bestehen nicht zu vernachlässigende konzeptionelle Unterschiede, die sich durch die unterschiedlichen Zielsetzungen erklären. Zum einen ist die Realisierung von Gridwelten nicht das primäre Ziel von ELMS, die Verwendung eines Grids ist daher bei ELMS auch nur eine optionale Funktion. Dadurch steht die Entwicklung gridbezogener Funktionen (wie z. B. Wahrnehmung basierend auf der eigenen Position und Hindernissen oder die explizite Unterstützung verschiedener Hindernistypen) viel weniger im Vordergrund als bei GridWorldSim. Zum anderen ist ELMS im Gegensatz zu GridWorldSim bewusst als Zwischensprache konzipiert, wobei im Idealfall ELMS-Spezifikationen von grafischen Erstellungswerkzeugen automatisch generiert werden. Ebenso ist es erklärtes Ziel der ELMS-Autoren, den Vorgang bei der Erstellung von Umgebungsspezifikationen so zu vereinfachen, dass der Benutzer über keine tiefergehenden Programmierkenntnisse verfügen muss. Unter diesem Aspekt erscheint eine Erweiterung von ELMS um frei programmierbare Komponenten im Rahmen der Zielsetzung als nicht wünschenswert.

Davon unabhängig wird durch das Fehlen eines (öffentlich verfügbaren) ELMS-Interpreters der Bedarf nach einem Frame-

work für Umgebungen ganz unabhängig von der prinzipiellen Eignung von ELMS nicht erfüllt.

8.2 VERGLEICH MIT MASSIM

MASSim kann als konzeptioneller Gegenentwurf zu ELMS betrachtet werden. Bei der Entwicklung von MASSim steht das Ziel im Vordergrund, ein technisches Hilfsmittel bereitzustellen, welches selbst in Java implementierte Umgebungen ausführt und über ein Netzwerk zur Verfügung stellt.

Im Gegensatz zum in dieser Arbeit vorgestellten Konzept und dessen Implementierung besitzt MASSim in Hinblick auf Zustandsüberführung, Wahrnehmung und Kommunikation im Wesentlichen keine Semantik, da zu deren Realisierung szenariospezifische Java-Klassen geladen werden, welche die Semantik der Umgebung implementieren. Durch diesen offenen Ansatz ist MASSim flexibler als GridWorldSim, da es durch den weitestgehenden Verzicht auf Vorgaben weniger einschränkende Vorgaben gibt. So ist zum Beispiel bei GridWorldSim die Erzeugung der Wahrnehmungsmenge durch Parameter wie Sichtweite, Sendestärke oder Empfangsempfindlichkeit beeinflussbar, jedoch ohne Änderung des Quellcodes (und damit Änderung des Konzepts) nicht beliebig bestimmbar.

Da MASSim für Wettbewerbe entwickelt wird, bei denen Agenten anhand vorgegebener Bewertungskriterien ihre Leistungsfähigkeit unter Beweis stellen können, unterstützt MASSim im Gegensatz zu GridWorldSim Mechanismen zur Verwaltung und zum Starten von Matches. Vorteile beim Einsatz von MASSim ergeben sich, wenn die gewünschten Umgebungen der Realisierung solcher Wettbewerbe dienen, das Verhalten einer solchen Umgebung größtenteils Funktionen erfordert, die von GridWorldSim nicht explizit unterstützt werden und daher auch bei GridWorldSim selbst implementiert werden müssten und wenn der Benutzer über ausreichende Programmierkenntnisse verfügt und größeren Aufwand zur Realisierung von Umgebungen nicht scheut.

Gelten diese Bedingungen mehrheitlich nicht, ist es mit GridWorldSim wesentlich weniger aufwändig, Umgebungen zu realisieren, da viele Funktionen, die bei MASSim erst selbst in Java implementiert werden müssten, bereits vorhanden sind und die Spezifikation einer Umgebung somit vollständig oder größtenteils deklarativ erfolgen kann. Insbesondere gilt, dass selbst einfache GridWorldSim-Umgebungen mit MASSim nur dadurch realisiert werden können, dass große Teile der GridWorldSim-Implementierung in Form eines MASSim-Szenarios neu implementiert werden. Selbst im Falle von Wettbewerben kann es daher weniger Aufwand bedeuten, GridWorldSim um Funktio-

nalität zur Verwaltung von Wettbewerben zu erweitern, als die Funktionalität von GridWorldSim nach MASSim zu portieren.

8.3 FAZIT UND AUSBLICK

Es lässt sich feststellen, dass das ihm Rahmen dieser Arbeit konzipierte und implementierte Framework in vielen Fällen die Realisierung von Gridwelt-Umgebungen im Vergleich zu bestehenden Ansätzen erheblich vereinfacht und dass es zur Realisierung von Gridwelt-Umgebungen – auch in Hinblick auf die Nichtverfügbarkeit einer Implementierung eines ELMS-Interpreters – oftmals keine vergleichbar geeignete Alternative gibt.

Die eingeführten Formalismen von Umgebungen haben sich als geeignet und hilfreich erwiesen, Umgebungen und ihr Verhalten zu spezifizieren und gemäß dieser Spezifikation zu implementieren. In Kapitel 7.6 ist ein Beispiel gegeben, bei welchem Funktionalität sowohl natürlichsprachlich als auch mit den im Rahmen dieser Arbeit entwickelten Formalismen und schlussendlich in Java-Code beschrieben wird. Die formale Beschreibung ist dabei die nach Meinung des Autors zu bevorzugende Darstellung, da sie zum einen kompakt und verständlich, zum anderen präzise ist.

Darüber hinaus helfen die im Rahmen dieser Arbeit entwickelten Konzepte dabei, den Unterschied zwischen Gridwelten und der realen Welt zu verkleinern und Agenten somit eine realistischere Umgebung zu bieten. So entspricht dank Diagonalkorrektur die Entfernung zwischen zwei Gridzellen im Erwartungswert dem euklidischen Abstand, anstatt einer diagonal vom Standort eines Agenten befindlichen Zelle den Abstand 1 (falls Diagonalbewegungen ohne Diagonalkorrektur möglich sind) oder 2 (falls Diagonalbewegungen nicht möglich sind) zuzuweisen. Auch die auf sichtbehindernden Hindernissen, diskreten Sichtlinien und euklidischem Abstand basierende Wahrnehmung approximiert die Gegebenheiten der realen Welt über den bislang für Gridwelten üblichen Umfang hinaus. Gleiches gilt für die Kommunikation, die durch Interferenzen, Sicht sowie Sendestärke und Empfangsstärke der Agenten beschränkt werden kann und darüber hinaus wie in der realen Welt zwischen öffentlichem Sprechen und privaten Gesprächen unterscheidet.

Das Konzept von Kapazität und Kapazitätsbedarf führt ebenfalls zu einem realistischeren Verhalten von Gridwelten. So sind Zellen nicht wahlweise gar nicht, nur einmalig oder beliebig betretbar, sondern die Betretbarkeit von Zellen hängt von den Eigenschaften der Zelle, den Eigenschaften der bereits in der Zelle befindlichen Agenten und Objekte sowie einer Eigenschaft des Agenten oder Objekts ab, welcher bzw. welches die Zelle betreten

soll. Gleiches gilt für die Inventare von Agenten und Objekten und deren Kapazität. Auch die Möglichkeit, dass Agenten sich – Zustimmung des Empfängers vorausgesetzt – Gegenstände übergeben können, statt Gegenstände auf dem Grid abzulegen und wieder aufzunehmen, macht Gridwelten realistischer.

Gridwelten besitzen bei der Realisierung von Multiagentensystemen den Vorteil, dass sie die Verortung von Agenten und Objekten wesentlich vereinfachen. In einer dreidimensionalen und räumlich wie zeitlich kontinuierlichen Umgebung stünde der zusätzliche Aufwand bei der Realisierung der Umgebung und Agenten oftmals in keinem Verhältnis zum dadurch gewonnenen Nutzen. So stehen bspw. bei der praktischen Evaluierung von Konzepten für wissensbasierte Agenten in der Regel andere Aspekte im Vordergrund als die Umsetzung aufwändiger Verfahren, die es ermöglichen, dass Agenten sich überhaupt sinnvoll in der Welt bewegen können. Die Verwendung gridbasierter Welten erscheint deshalb sinnvoll und soll in Hinblick auf mögliche Erweiterungen im Anschluss an diese Arbeit beibehalten werden, wobei die Erweiterung des zweidimensionalen Grids um eine dritte Dimension durchaus erwägenswert sein könnte.

Im Rahmen dieser Arbeit wurde vor allem ein für eigene Erweiterungen offenes Framework konzipiert und implementiert. Dabei mussten viele Eigenschaften der realen Welt unberücksichtigt bleiben, so dass es für diese keine Entsprechung in von GridWorldSim realisierten Umgebungen gibt. Dazu zählt bspw. das Zusammenbauen von Objekten zu Objekten neuen Typs, über Nebel hinausgehende Naturereignisse, die Zerstörung von Agenten und Objekten durch Naturereignisse oder Agenten, die Anwendung eines Objekts auf ein anderes Objekt um dieses zu modifizieren, unterschiedliche räumliche Ausdehnung von Objekten, so dass diese mehr als eine Gridzelle belegen können oder auf mehr als den derzeit verwendeten Parametern basierenden Wahrnehmungen.

Wenngleich eine vollständige Abstraktion der realen Welt durch Gridwelten einen nicht zu leistenden Aufwand darstellt, so sind doch viele Erweiterungen denkbar, die sowohl sinnvoll als auch realisierbar sind und es den Entwicklern von Multiagentensystemen ermöglichen, ihre Agenten in für ihre Bedürfnisse geeigneteren Umgebungen zu testen.

[BDHK] Tristan Behrens, Jürgen Dix, Jomi Hübner, and Michael Köster. *Multi-Agent Programming Contest Protocol Description and Further Informations (2010 Edition)*. Fakultät für Mathematik/Informatik und Maschinenbau, TU Clausthal. `http://www.multiagentcontest.org/2010/` (abgerufen am 25. Oktober 2010). (Cited on page 10.)

[BKI03] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen*. Viewg, Braunschweig [u.a.], 2., überarb. u. erw. aufl. edition, 2003. (Cited on page 39.)

[BOdO+04] R. H. Bordini, F. Y. Okuyama, D. de Oliveira, G. Drehmer, and R. C. Krafta. The mas-soc approach to multi-agent based simulation. In Gabriela Lindemann, Daniel Moldt, and Mario Paolucci, editors, *RASTA*, volume 2934 of *Lecture Notes in Computer Science*, pages 70–91. Springer, 2004. (Cited on pages 3 and 7.)

[Bre65] Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. (Cited on page 74.)

[CFF+92] Hans Chalupsky, Tim Finin, Rich Fritzson, Don Mckay, Stu Shapiro, and Gio Wiederhold. An overview of KQML: A knowledge query and manipulation language. Technical report, KQML Advisory Group, Department of Computer Science, Stanford University, 1992. (Cited on page 124.)

[DDN06] Mehdi Dastani, Jürgen Dix, and Peter Novák. The second contest on multi-agent systems based on computational logic. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *CLIMA VII*, volume 4371 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2006. (Cited on pages 2 and 10.)

[FM96] Jacques Ferber and Jean-Pierre Müller. Influences and reaction: a model of situated multiagent systems. In *Proceedings of the Second International Conference*

on Multiagent Systems, pages 72–79, 1996. (Cited on page 21.)

[FN71] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971. (Cited on page 39.)

[Kru] Jack Krupansky. Foundations of software agent technology. Web site, `http://agtivity.com/def/multi_agent_system.htm` (abgerufen am 25. Oktober 2010). (Cited on page 1.)

[McC85] John McCarthy. Formalization of strips in situation calculus. Technical report, Formal Reasoning Group, Department of Computer Science, Stanford University, 1985. (Cited on page 39.)

[OBdRC04] Fabio Y. Okuyama, Rafael H. Bordini, and Antônio Carlos da Rocha Costa. ELMS: An environment description language for multi-agent simulation. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *E4MAS*, volume 3374 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2004. (Cited on pages 2, 7, and 8.)

[RN09] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 3. edition, 2009. (Cited on page 5.)

[Sid05] B. G. Sidharth. *The Universe of Fluctuations: The Architecture of Spacetime and the Universe*, volume 147 of *Fundamental Theories of Physics*. Springer, 2005. (Cited on page 14.)

[SS98] Josefina Sierra-Santibáñez. A declarative formalization of strips. In *Thirteenth European Conference on Artificial Intelligence, ECAI-98*, pages 509–513, 1998. (Cited on page 39.)

[WH02] Danny Weyns and Tom Holvoet. Look, talk and do: A synchronization scheme for situated multiagent systems. In Peter McBurney and Michael Wooldridge, editors, *Workshop Notes of UK Workshop on Multi-agent Systems*, 2002. (Cited on page 2.)

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995. (Cited on page 1.)

[WOO07] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007. (Cited on page 2.)

[WPM+04] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. Environments for multiagent systems state-of-the-art and research challenges. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *E4MAS*, volume 3374 of *Lecture Notes in Computer Science*, pages 1–47. Springer, 2004. (Cited on page 2.)

[WVH05] Danny Weyns, Giuseppe Vizzari, and Tom Holvoet. Environments for situated multi-agent systems: Beyond infrastructure. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *E4MAS*, volume 3830 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005. (Cited on page 1.)