Rescueanalyser -

Konzeption und Entwicklung eines Werkzeugs zur Analyse und Planerkennung im RoboCupRescue

Diplomarbeit
zur Erlangung des Grades einer Diplom-Informatikerin

vorgelegt von

Stefanie Burchert-Uhrmacher
lucy@uni-koblenz.de

Erstgutachter: Professor Doktor Ulrich Furbach
Zweitgutachter: Diplom-Informatiker Jan Murray

Koblenz, im März 2007

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

(Ort, Datum) (Unterschrift)

Vorwort

Die vorliegende Diplomarbeit entstand während meiner Zeit als Mitglied der Arbeitsgruppe Künstliche Intelligenz (AGKI) im Rahmen des Projektes *RoboLog* an der Universität Koblenz-Landau. Ich möchte mich an dieser Stelle bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben, und mir die Möglichkeit gaben, die Welt der KI in der Praxis zu erleben und Forscher auf diesem Gebiet kennen zu lernen.

Mein erster Dank gilt den wissenschaftlichen Betreuern, Prof. Dr. Ulrich Furbach und Dipl.-Inform. Jan Murray. Zunächst möchte ich Ihnen für Ihre Bereitschaft und Ihr unermüdliches Engagement danken, durch gemeinsame, konstruktive Diskussionen diese Arbeit zu formen.
Außerdem danke ich Ihnen für die Möglichkeit der Teilnahme an KI Konferenzen, Symposien, Messen, sowie der Teilnahme am RoboCup 2006, welches ein besonderes Erlebnis für mich darstellte.

Weiterhin danke ich Dr. Oliver Obst, der die Idee zu dieser Arbeit lieferte.

Auch verdient die Unterstützung durch Tomomi Kawarabayashi besondere Erwähnung. Sie hat Zwischenergebnisse dieser Arbeit mit mir diskutiert und ausgewertet. Ich danke Ihr für die investierte Zeit und das Interesse an meiner Arbeit.

Schließlich danke ich meinen Eltern für Ihre andauernde Unterstützung und die Finanzierung meines Studiums.

Stefanie Burchert-Uhrmacher

Kurzfassung

Im Forschungsgebiet der Künstlichen Intelligenz und Multiagenten-Systeme stellt die Erkennung von Plänen eine besondere Herausforderung dar. Es gilt aus einer Menge von Beobachtungen Agentenpläne effektiv, zuverlässig und eindeutig zu identifizieren. Im Bereich des RoboCup ist es von entscheidender Bedeutung, dass Agenten über die Pläne und Intensionen anderer Agenten schlussfolgern können, um gemeinsame Ziele zu erreichen.

Diese Diplomarbeit umfasst die Konzeption und Entwicklung eines Werkzeugs für die RoboCupRescue-Simulation, welches die Abläufe einer Rescue-Simulation analysiert und die beobachteten Aktionsfolgen Plänen einer vorspezifizierten Planbibliothek zuordnet.
Das entwickelte Analysewerkzeug zeichnet sich einmal durch seine Unabhängigkeit von den verschiedenen Rescue-Agententeams aus, sowie durch die Verwendung neuester Methoden zur effektiven Planerkennung.
Damit ermöglicht der entwickelte Rescueanalyser sowohl die Fehlersuche und Optimierung des eigenen Agententeams als auch die Erforschung und Auswertung unterschiedlicher Strategien der Rescue-Agenten konkurrierender Teams.

Neben der Visualisierung der Ergebnisse mittels eines optimierten Monitors des Rescue-Simulators, können die Resultate und Auswertungen der Planerkennung auch durch textbasierte Ausgaben detailliert eingesehen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung und Zielsetzung

1.1 Einführung und Rahmen

Diese Diplomarbeit wurde im Rahmen des Forschungsprojektes *RoboLog* an der Universität Koblenz-Landau innerhalb der Arbeitsgruppe *Künstliche Intelligenz* erstellt. Die Mitglieder des Projektes *RoboLog* beschäftigen sich mit den vielfältigen Aspekten von Multiagenten-Systemen in den *RoboCup*-Domänen *Robotic Soccer* und *Rescue Robotics* [1]. Der *RoboCup* stellt eine internationale Initiative zur Förderung der Entwicklung in den Bereichen Robotik und Künstliche Intelligenz dar [2]. Die Domäne *Robotic Soccer* beschäftigt sich, auf Basis der populären Sportart Fußball, mit interagierenden, intelligenten Agenten. In der Domäne *Rescue Robotics* - kurz RoboCupRescue - werden in virtuellen Katastrophensituationen Such- und Rettungsaufgaben simuliert. Beide Bereiche werden sowohl in der Simulation als auch mit echten Robotern betrieben und erforscht.
Das *RoboLog*-Team der Universität Koblenz-Landau nimmt seit 1999 an den RoboCup-Wettkämpfen in der Fußball Simulationsliga teil, und ist auch in anderen Bereichen der RoboCup-Initiative aktiv tätig. Exemplarisch seien an dieser Stelle die Organisation der RoboCup-Wettbewerbe und die Entwicklung des 3D Simulators für die Fußballsimulation genannt.
Seit Sommer 2005 hat sich das Interesse des *RoboLog*-Teams auch auf die Domäne RoboCupRescue ausgeweitet. Die Rescue-Simulation bietet, ebenso wie die Fußballsimulation, ein Anwendungsfeld für die praktische Erprobung der Methoden aus den Forschungsgebieten Künstliche Intelligenz sowie Multiagenten-Systeme. Darüber hinaus kommt der Rescue-Simulation, eine besondere gesellschaftliche Bedeutung im Rahmen des Katastrophenschutzes zu. [2]

1.2 Motivation

In der RoboCupRescue-Simulation werden intelligenten Agenten in einer simulierten Katastrophensituation Such- und Rettungsaufgaben übertragen. Die Simulation verfolgt das Ziel, den Gesamtschaden der virtuellen Welt möglichst gering zu halten. Die erfolg-

reiche Durchführung hängt daher in großem Maße von den Such- und Rettungsstrategien der agierenden Agententeams ab.
Der Erfolg einer Strategie wird maßgeblich von der Koordination der Handlungen und Effizienz der Kommunikation bestimmt. Durch Beobachtung der Handlungen einzelner Agenten oder Agentengruppen lassen sich Informationen über mögliche Strategien der Agenten erheben. Mittels einer geeigneten Analyse der erhobenen Daten können Muster abgeleitet werden, die eine Kategorisierung von Strategien ermöglichen. Aus der Beobachtung und Auswertung einer Vielzahl von abgearbeiteten Szenarien können Strategien und Planungsgrundlagen der agierenden Agenten analysiert werden.
Auf der *International Joint Conference on Artificial Intelligence* (*IJCAI*) wurde im Jahr 2005 die Arbeit von Dorit Avrahami-Zilberbrand und Gal A. Kaminka unter dem Titel *Fast and Complete Symbolic Plan Recognition* [3] veröffentlicht. Diese Veröffentlichung beschreibt die wesentlichen Konzepte eines neuartigen, effizienten Planerkennungsverfahren allerdings auf Basis der RoboCupSoccer-Simulation.

1.3 Zielsetzung und Vorgehensweise

Ziel dieser Diplomarbeit ist die Entwicklung eines Analysewerkzeugs für die RoboCupRescue-Simulation. Um eine Agententeam-unabhängige Vorgehensweise zu gewährleisten, werden die Daten für die Analyse aus der RoboCupRescue-Simulationsumgebung entnommen. Im Zuge dieser Arbeit werden mehrere Anwendungsgebiete abgedeckt. Zunächst werden die aus der Simulationsumgebung entnommenen Daten, teamunabhängig aufbereitet. Damit können diese Daten zur vergleichenden Analyse von Agententeams aber auch zum Debugging des eigenen Agententeams herangezogen werden. Weiterhin bilden die aufbereiteten Daten die Basis für das Planerkennungsverfahren, welches innerhalb des Analysewerkzeugs implementiert werden soll. Die Planerkennung bildet den Schwerpunkt dieser Arbeit, welche sich grob in zwei Teile untergliedert:

Im ersten Teil werden zunächst die themenrelevante informatikspezifische Basisterminologie und die für diese Arbeit erforderlichen theoretischen Grundlagen zusammengestellt. Dazu erfolgt in Kapitel 2 die Definition der grundlegenden Begriffe aus dem Bereich der Multiagenten-Systeme, sowie eine Einführung der verwendeten Methoden aus dem Forschungsgebiet der Künstlichen Intelligenz. Weiterhin werden die für diese Arbeit relevanten Konzepte aus dem wissenschaftlichen Beitrag *Fast and Complete Sym-*

bolic Plan Recognition [3] erläutert. In Kapitel 3 wird der RoboCupRescue-Bereich beschrieben, der für diese Arbeit als Forschungsgrundlage dient. Es erfolgt die Darstellung des Aufbaus und der Arbeitsweise des Rescue-Simulators, der die Basis des Analysewerkzeugs bildet. Der erste Teil schließt mit einer Zusammenfassung der für diese Arbeit relevanten anwendungsspezifischen Erkenntnisse.

Der zweite Teil beinhaltet die umfassende Dokumentation zur Entwicklung des generischen Analysewerkzeugs der RoboCupRescue-Simulation. Dazu werden in Kapitel 4 zunächst die Anforderungen an das Werkzeug definiert. Darüber hinaus wird eine Abgrenzung zu bereits vorhandenen Entwicklungen im Bereich der RoboCupRescue-Simulation geschaffen. Anschließend wird in Kapitel 5 die resultierende Architektur und die Arbeitsweise des Rescue-Analysewerkzeugs erörtert. Die detaillierte Beschreibung der relevanten Komponenten und deren Umsetzung sind in Kapitel 6 dokumentiert. Kapitel 7 beinhaltet die Darstellung der durchgeführten Experimente und die erzielten Prüfergebnisse.

Diese Arbeit schließt mit einer Zusammenfassung und einem Ausblick zu möglichen Erweiterungen des entwickelten Werkzeugs.

I Theoretische Grundlagen, Anwendungsgebiete und Ausgangsbasis

Dieser Teil der Diplomarbeit stellt eine Einführung in die Grundlagen des ausgearbeiteten Themas dar. Es wird ein Überblick zum Forschungs- und Anwendungsgebiet als Basis dieser Arbeit gegeben.

Kapitel 2 liefert einen Einblick in das zugrunde liegende Forschungsgebiet, der Künstlichen Intelligenz und insbesondere der Multiagenten-Systeme. Das Anwendungsgebiet - die RoboCupRescue-Simulation - in welches die aus dieser Diplomarbeit hervorgehende Entwicklung eingebettet ist, wird in Kapitel 3 vorgestellt. Abschließend erfolgt eine Zusammenfassung der relevanten Grundlagen, die dann im zweiten Teil dieser Diplomarbeit, im Zuge der Entwicklung eines Analysewerkzeugs, verwendet werden.

2 Theoretische Grundlagen: Multiagenten-Systeme und Methoden der KI

Um eine Grundlage zum Verständnis des Forschungsumfeldes dieser Arbeit zu schaffen, werden in diesem Kapitel zunächst die grundlegenden Begriffe aus den Bereichen Künstliche Intelligenz und Multiagenten-Systeme dargestellt.

2.1 Agenten und Multiagenten-Systeme

Ein *Agent* ist ein Computer-System, das seine Umgebung mittels Sensoren wahrnimmt und darauf autonom und rational mittels Effektoren (re)agiert (vgl. Abbildung 1). [4]

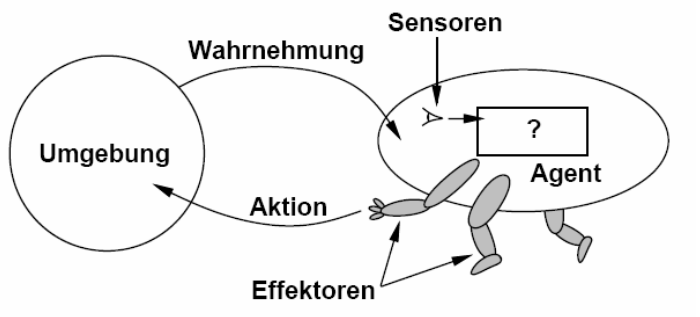


Abbildung 1: Abstrakte Darstellung eines Agenten [4]

Ein *intelligenter Agent* besitzt dabei folgende Eigenschaften [5]:

1. Er ist reaktiv: Er besitzt die Fähigkeit auf Änderungen der Umgebung in Zeitintervallen zu reagieren, um Ziele zu erreichen;
2. Er ist pro-aktiv: Er kann Initiative übernehmen und sich zielgerichtet verhalten;
3. Er ist sozial: Er tritt in Interaktion mit anderen Agenten, um gemeinsame Ziele zu erreichen.

Arbeiten mehrere autonom agierende Entitäten (Agenten) zusammen, um ein gegebenes Ziel zu erreichen, so handelt es sich definitionsgemäß um ein *Multiagenten-System*, kurz *MAS*. [4]

Die interne Struktur eines Agenten setzt sich zusammen aus der *Architektur* und dem *Programm* [4]:

Agent = Architektur + Programm

Unter der *Architektur* versteht man den inneren Aufbau eines Agenten. Folgende grundlegende Architekturen werden unterschieden [5]:

- BDI Agenten
- Reaktive Agenten
- Mehrschichtige Agenten

BDI Agenten - diese Abkürzung steht für Überzeugungen (*Beliefs*), Wünsche (*Desires*) und Intentionen (*Intentions*) - verfügen über ein Planungs- und Schlussfolgerungssystem, welches den Entscheidungsprozess durch Überzeugungen, Wünsche und Intentionen beeinflusst.
Bei reaktiven Agenten fehlen diese aus der Künstlichen Intelligenz bekannten Methoden. Die Entscheidungen werden unmittelbar (reaktiv) durch die sich ändernde Umgebung ausgelöst. Reaktive Agenten haben im Gegensatz zu BDI Agenten keine interne Struktur, sie wählen lediglich eine auf das Ereignis passende Regel aus einer Regelbasis aus. Der Vorteil dieser Architektur liegt in den kurzen Antwortzeiten.
Mehrschichtige Agenten vereinigen die Architektur von BDI und Reaktiven Agenten. Sie besitzen eine interne Struktur, die sich aus einem Planungsmodul und einem reaktiven Modul zusammensetzt. [5]

Die Architektur liefert die Basis für das *Programm* eines Agenten. Mit ihm werden Abbildungsregeln realisiert, welche Wahrnehmungen auszuführenden Aktionen zuordnen [4].
Ebenso wie bei der Architektur der Agenten, lassen sich anhand der Programme unterschiedliche Typen von Agenten klassifizieren [5]:

- einfache reaktive Agenten
- Agenten mit internem Zustand
- zielgerichtete Agenten
- nützlichkeits-orientierte Agenten

Einfache reaktive Agenten reagieren lediglich auf Reflexe. Sie wählen ihre Aktionen aufgrund der aktuellen Wahrnehmung. Das Verhalten des Agenten ist direkt an die Umgebung gebunden.
Agenten, die einen *internen Zustand* verwalten, berücksichtigen die durch ihre Aktionen ausgelösten Änderungen der Umwelt. Die Wahrnehmungen beeinflussen über eine *Zustandsaktualisierungsfunktion* den internen Zustand des Agenten. Im einfachsten Fall werden die Wahrnehmungen in den Zustandsspeicher hinzugefügt. Auch komplexere Aktualisierungsfunktionen sind möglich, so können beispielsweise veraltetes Wissen gelöscht oder Widersprüche erkannt und behandelt werden.
Zielgerichtete Agenten verwalten Ziele, um wünschenswerte Situationen zu erreichen. Ein zielorientierter Agent wählt immer solche Aktionen aus, mit denen er entweder ein Ziel erreicht, oder zumindest dem anvisierten Ziel näher kommt.
Nützlichkeits-orientierte Agenten beurteilen die Auswirkung ihrer Handlung nicht bezogen auf einen bestimmten Wunschzustand, sondern versuchen, eine *Nutzenfunktion* zu maximieren. Die Nutzenfunktion bildet verschiedene Umweltzustände auf einer Skala ab, welche Präferenzen hinsichtlich verschiedener Zustände beschreibt. So ist der Agent in die Lage, auch Zielkonflikte zwischen mehreren wünschenswerten Zuständen zu beurteilen. [22]

Die Architektur einzelner Agenten stellt den Forschungsschwerpunkt der Multiagentensysteme dar. Hingegen beschäftigt sich die Künstliche Intelligenz damit, Agenten-Programme zu entwerfen. [6]

2.2 Planung in der Künstlichen Intelligenz

Intelligente Agenten sind in der Regel zielorientiert. Zur Erreichung eines Ziels wird ein intelligenter Agent einen Plan verfolgen. Ein *Plan* besteht aus einer Sequenz von Aktionen derart, dass die sequentielle Ausführung der Aktionen zur Zielerreichung des Agenten führt. [4]
In einem Multiagenten-System, in dem konkurrierende Agenten verschiedene oder gleiche Ziele verfolgen, ist es wichtig die Pläne und Ziele einzelner Agenten oder von Agentengruppen zu erkennen. Als *Planerkennung* wird die Aufgabe bezeichnet, anhand einer Menge von beobachteten Aktionen eines intelligenten Agenten auf dessen Ziel oder dessen nächste mögliche Aktion zu schließen. [7]

Innerhalb der Planerkennung wird zwischen Agenten unterschieden, welche ein vollständiges und exaktes Wissen über ihre Umwelt haben, und solchen Agenten die nur über ein unvollständiges Wissen verfügen. [8]
In dieser Diplomarbeit wird insbesondere der letzte Fall von Bedeutung sein.

Die für den Planerkennungsprozess innerhalb dieser Arbeit verwendeten Methoden basieren auf dem im Rahmen der *International Joint Conference on Artificial Intelligence* (*IJCAI*) 2005 erschienenen Artikel von Dorith Avrahami-Zilberbrand und Gal A. Kaminka *Fast and Complete Symbolic Plan Recognition* [3].
In diesem Artikel werden neue effiziente Methoden vorgestellt, um einen Planerkennungsprozess bei modernen Roboteranwendungen zu implementieren. Als Testfeld dieser Methoden werden fortschrittliche Fußballroboterteams, wie sie bei der RoboCupSoccer-Simulation zu finden sind, eingesetzt. Für die heutigen Roboteranwendungen sind bisher existierende Ansätze zur Planerkennung in ihren Leistungen nicht mehr zufrieden stellend. Klassische Planerkennungsverfahren gehen meist davon aus, dass die zu beobachtenden Aktionen diskret und unverzüglich und nur eine zu einem Zeitpunkt ausgeführt werden. [30] Ebenso werden die möglichen Auswirkungen der Aktionen auf die Umwelt oder den Agenten selbst vernachlässigt. Daher ist es nicht möglich innerhalb des Planerkennungsprozesses die Reaktionen auf sich ändernde Faktoren in der Umwelt zu erkennen. [28] [29]
Es existieren zwar Ansätze, die sich mit dieser Problematik beschäftigen. Dabei sind Mengen von Beobachtungen zugelassen, die aus weiteren Eigenschaften, als nur den Aktionen der Agenten bestehen - so genannte *multi-featured observations.* So gehen beispielsweise der von Tambe und Rosenbloom 1995 entwickelte Ansatz *RESC* (*Real-time Situated Commitments*) [9] oder *RESL* (*Real-time Situated Least-commitments*) [10] diese Probleme an. Beide Algorithmen ignorieren jedoch die Historie der Beobachtungen, die in dem behandelten Ansatz von Avrahami-Zilberbrand und Kaminka in den Planerkennungsprozess miteinbezogen wird. In diesem Ansatz wird nicht nur die Beobachtungsmenge zu einem Zeitpunkt betrachtet, sondern es werden zusätzlich die Beobachtungen zu früheren sowie späteren Zeitpunkten in den Planerkennungsprozess mit aufgenommen. In Erweiterung zu bisherigen Methoden wird durch dieses Verfahren die Mehrdeutigkeit der erkannten Pläne reduziert. [3] [28]
Mit den meisten Algorithmen zur Planerkennung hat der Ansatz von Avrahami-Zilberbrand und Kaminka gemeinsam, dass eine Planbibliothek Verwendung findet. Typischerweise ist auch hier die Planbibliothek als vollständige, aber minimale Sammlung aller möglichen Pläne dargestellt, die ein beobachteter Agent ausführen kann.

Diese Pläne sind hierarchisch angeordnet. Während des Planerkennungsprozesses werden die Beobachtungen mit Teilplänen oder Planschritten der Planbibliothek abgeglichen. In der Regel gibt es keine eindeutige Zuordnung der Beobachtungen zu einem Teilplan bzw. Planschritt. Daher besteht die Herausforderung darin, trotz verschiedener Möglichkeiten den richtigen Teilplan bzw. Planschritt zu erkennen. [7]
Ein Ansatz zur Lösung des Problems besteht in der Zuordnung von Wahrscheinlichkeiten zu Planschritten oder Aktionen. Dieses Vorgehen wird als *probabilistischer* Ansatz bezeichnet. Eine weitere Möglichkeit besteht im Abgleichen mehrerer aufeinander folgender Beobachtungen. [7] [29]

Da es sich bei den Beobachtungen um Mengen handelt, kann das Abgleichen aller Elemente der Beobachtungsmenge mit allen möglichen Planschritten der Planbibliothek sehr aufwändig sein. Eine neu eingeführte Datenstruktur ermöglicht es, diese Beobachtungsmengen strukturiert zu analysieren und damit den Aufwand zu reduzieren. Diese Datenstruktur wird als Feature Decision Tree (FDT) bezeichnet. Der FDT ist ein Entscheidungsbaum der beim Abgleichen von Elementen der Beobachtungsmenge mit Planschritten aus der Planbibliothek die Anzahl der Vergleichsoperationen reduziert. Ohne Verwendung des FDT kann im schlechtesten Fall der Aufwand exponentiell groß werden. Durch den Einsatz des FDT beträgt der Aufwand maximal lineare Größe - in Abhängigkeit von der Tiefe des Baumes. [3] [28]

Da sich der Einsatz der in den Artikeln [3], [28] vorgestellten Datenstrukturen, wie Planbibliothek und FDT, auf die Domäne fortschrittlicher Fußballroboter bezieht, wird in diesem Kapitel von einer tiefer gehenden Beschreibung der verwendeten Methoden abgesehen. Stattdessen werden im zweiten Teil dieser Ausarbeitung die im Bereich der RoboCupRescue-Simulation entwickelten Methoden zur Planerkennung unter Verwendung der hier vorgestellten Repräsentationen ausführlich erläutert.

3 RobocupRescue-Simulation

In diesem Kapitel wird das innerhalb der Diplomarbeit analysierte Anwendungsgebiet - der RoboCup und insbesondere die RoboCupRescue-Simulation - beschrieben.

3.1 Allgemeine Einführung in den RoboCup

Der *RoboCup* ist eine internationale Initiative zur Förderung der Forschung und der Entwicklung in den Bereichen Künstliche Intelligenz und Robotik. Die Initiative verfolgt die Idee, Teams aus der ganzen Welt in einer standardisierten Umgebung gegeneinander unter den gleichen Rahmenbedingungen Roboterfußball spielen zu lassen.
Seit 1997 finden jedes Jahr RoboCup-Wettkämpfe statt. Auf diese Weise wird den Wissenschaftlern die Möglichkeit geboten, die erreichten Forschungs- und Entwicklungsergebnisse dem internationalen Vergleich zu stellen und so gemeinsam Fortschritte in den einzelnen Fachdisziplinen zu erzielen. [26]
Dabei wird ein klares Ziel verfolgt [11]:

> "*By 2050, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official FIFA rules, against the winner of the most recent World Cup of Human Soccer.*"

Im RoboCup haben sich über die Jahre drei generelle Bereiche entwickelt: *RoboCupSoccer*, *RoboCupRescue* und *RoboCupJunior*. Diese Bereiche unterteilen sich jeweils in unterschiedliche Ligen, die sich durch die Robotergröße, die Spieleranzahl, die Spielfeldgröße und das Anforderungsprofil an die Roboter unterscheiden. Eines ist allerdings allen Spielern gemein - sie agieren selbstständig und ohne menschliche Hilfe beim Spiel. [26]

3.1.1 RoboCupSoccer

RoboCupSoccer ist der älteste Bereich und die Realisierung der ursprünglichen Idee. Durch die hohe Dynamik und die Vielzahl zu bewältigender Teilaufgaben bietet der RoboCupSoccer-Bereich ein interessantes Forschungsgebiet für unterschiedliche Dis-

ziplinen in der Informatik und der Robotik. Um die unterschiedlichen Anforderungsprofile an die Roboter zu realisieren, existieren verschiedene Ligen:

- *Small-Size* Liga
- *Middle-Size* Liga
- *Four-Legged* Liga
- *Humanoid* Liga

In der *Small-Size* Liga spielen zwei Teams mit je fünf Roboter auf einem ca. 6 Meter langen und 4 Meter breiten Spielfeld. Die maximale Größe der Roboter beträgt im Durchmesser 18 cm und in der Höhe 15 cm. Der Antrieb der Roboter erfolgt mittels integrierter Stromversorgung und Elektromotoren auf Rollen. Auf Grund der geringen Größe, besteht die Herausforderung in dieser Liga in der Minimalisierung der Elektronik. Daher sind auch nur wenige Sensoren direkt auf dem Roboter angebracht. Stattdessen erhalten die Roboter die Informationen über ihre Umgebung anhand einer über dem Spielfeld global angebrachten Kamera. Externe Computer verarbeiten diese Informationen in Echtzeit und senden die berechneten Befehle per Funk an die Roboter. Alle Roboter - auch die des Gegners - werden anhand von Farbmarkierungen, die oben auf den Robotern angebracht sind, unterschieden. [12][18]

In der *Middle-Size* Liga bestehen die beiden Teams in der Regel aus vier Robotern auf einem Spielfeld von 12 Metern Länge und 8 Metern Breite. Alle vorhandenen Objekte - andere Roboter und Landmarken - werden auch hier anhand von Farben unterschieden. Die Kommunikation zwischen den Robotern erfolgt per WLAN. Die Roboter haben eine Größe von max. 2500 cm² Grundfläche und wiegen bis zu 40 kg. Wie auch in der *Small-Size* Liga bewegen sich die Roboter auf Rollen über das Spielfeld. Allerdings müssen in dieser Liga alle Sensoren am Roboter angebracht werden. In der Regel verfügt jeder Roboter über ein Laptop oder ein Subnotebook, welches die Informationen der Sensoren erfasst, mittels Bildverarbeitung interpretiert, und mit dem verfügbaren Wissen entscheidet welche Aktionen der Roboter ausführen soll. [12][18]

Bei der *Four-Legged* Liga handelt es sich um zwei Teams mit je vier Robotern der Marke SONY AIBO. Das Spielfeld hat eine Länge von 6 Metern und eine Breite von 4 Metern. Die agierenden Roboter sind einem vierbeinigen Hund nachempfunden und verfügen über 20 Freiheitsgrade in ihren Bewegungsabläufen. Zudem sind die Roboter mit allen erforderlichen Sensoren ausgestattet. Dazu zählen eine Kamera, sowie Distanz-

und Berührungssensoren. Eine besondere Herausforderung in dieser Liga besteht darin, dass durch die Bewegung der Gliedmaßen, die Kamera extrem verwackelte Bilder liefert, die durch entsprechende Bildverarbeitungstechniken realitätsnah zu interpretieren sind.
Diese konventionell erwerbbaren Roboter ermöglichen einen günstigen Einstieg in den RobocupSoccer und stellen das Bindeglied zwischen den Radangetriebenen Robotern in der *Small-* und *Middle-Size* Liga und den zweibeinigen Humanoiden Robotern da. [12] [18]

Die *Humanoid* Liga mit menschenähnlichen Robotern hat sich erst 2002 etabliert. Nach anfänglich einfacheren Wettbewerben im Elfmeterschießen und Wettläufen, beispielsweise um Hindernisse, hat es bei der Weltmeisterschaft in Osaka 2005 erstmals Fußballspiele mit zwei Spielern pro Mannschaft gegeben. Die humanoiden Fußballroboter sind dabei vollständig autonom. Lediglich untereinander dürfen sie über WLAN kommunizieren. Wegen der geringen Tragfähigkeit der Roboter nutzen viele Teams Pocket PCs für Bildverarbeitung, Selbstlokalisierung, Verhaltenssteuerung und Bewegungsplanung. Nach dem Anpfiff läuft das Spiel ohne äußere Einflüsse und Hilfe. Die Komplexität der Bewegungssteuerung beim Laufen mit zwei Beinen und die Wahrung des Gleichgewichts, beispielsweise beim Schießen des Balles, stellen eine besondere Herausforderung dar. Falls die Roboter während des Spiels zu Boden gehen, müssen sie in der Lage sein ohne fremde Hilfe wieder auf die Beine zu kommen.
Die Roboter werden in zwei Größenklassen eingeteilt: KidSize mit einer Größenbeschränkung von 30-60 cm und TeenSize mit 65-130 cm. Neben diesen Größenvorgaben, existieren Bestimmungen über den physischen Aufbau eines humanoiden Roboters, die sicherstellen, dass die Füße nicht zu groß sind, der Schwerpunkt nicht zu niedrig liegt und die Arme und Beine nicht zu lang oder zu kurz sind. [12][18]

Neben den Ligen, in denen sich echte Roboter auf realen Spielfeldern bewegen, existiert die *Simulationsliga*, bei der die Spieler aus virtuellen Agenten bestehen. Die Agenten agieren in zwei Teams aus jeweils 11 Spielern innerhalb einer Computersimulation, die ein virtuelles Spielfeld in der Größe eines originalen Fußballfeldes darstellt. Jeder Agent ist als ein separater Prozess implementiert. In diesem Umfeld lassen sich Strategien, Taktiken und Gruppenverhalten von Agenten erproben, um diese dann auf humanoide Roboter zu übertragen. Die Simulationsliga existiert in einer 2D- und 3D-Version. [12][18]

3.1.2 RoboCupJunior

Seit 2000 existieren bei den RoboCup-Weltmeisterschaften auch eigene Wettbewerbe für Junioren, die innerhalb des RoboCupJunior-Bereichs ausgetragen werden. Dieser Wettbewerb ist speziell für Schüler ausgerichtet und bietet jungen Nachwuchswissenschaftlern die Möglichkeit mit einfachen Programmiersprachen und Baukästen, zum Beispiel von LEGO oder Fischertechnik, fantasiereiche Roboter zu konstruieren. Die Wettkämpfe werden in drei verschiedenen Ligen ausgetragen. Neben den bekannten Ligen *Soccer* und *Rescue* existiert die *RoboCupDance* Liga, bei der sich die selbstgebauten Roboter in einer programmierten Choreographie zu Musik bewegen bzw. tanzen. Zum Teil sind die Roboter dabei "verkleidet", "erzählen" Geschichten oder die Programmiererinnen und Programmierer tanzen dazu oder ergänzen ihre Vorführung durch ein selbst gestaltetes Bühnenbild. Entscheidend ist der Gesamteindruck der Vorführung, der von einer Jury bewertet wird. [13]

3.1.3 RoboCupRescue

Der Bereich RoboCupRescue wurde von japanischen Forschern angeregt, nachdem 1995 durch ein Erdbeben in der japanischen Stadt Kobe mehr als 6.500 Menschen darunter auch zahlreiche Rettungskräfte ihr Leben lassen mussten. Der Schaden der gesamten Infrastruktur wurde auf mehr als eine Milliarde US Dollar geschätzt. [14]
Derzeit gibt es zwei Projekte im Bereich RoboCupRescue, die *Simulationsliga* und die *RealRobot* Liga. Ziel dieser Projekte ist die Anwendung der Erkenntnisse aus den Forschungsgebieten Künstliche Intelligenz und Robotik auf das Gebiet von Rettungsaktionen bei Naturkatastrophen. Die Herausforderung besteht darin, verschiedenartige Einsatzkräfte in einer feindseligen Umgebung zu koordinieren.
Das Einsatzgebiet in der *RealRobot* Liga ist auf eine Arena in Zimmergröße beschränkt. Innerhalb dieser so genannten Testarena existieren unterschiedliche Schwierigkeitszonen, die Katastrophenszenarien wie etwa nach einem Erdbeben darstellen. Die gelbe Zone stellt eine leicht beschädigte Büroumgebung dar, in der Papier und einige umgefallene Gegenstände herumliegen. In der orangen Zone müssen bereits Rampen und Treppen bewältigt werden. Die rote Zone zeichnet sich durch instabile Trümmerberge aus und umfasst schlecht zugängliche Hohlräume unterschiedlicher Größe. Innerhalb dieser Testarena müssen die Roboter nach Opfern suchen und deren Position an einen Einsatzleiter weiter geben. Um das Gebiet systematisch zu erkunden und die genauen

Positionen gefundener Opfer zu melden, müssen die Roboter zunächst eine Karte der Umgebung erstellen.
In dieser Liga werden die höchsten Anforderungen an die motorischen und sensorischen Fähigkeiten der Roboter gestellt. Auf den Robotern sind daher leistungsfähige Rechner und eine Vielzahl von Sensoren angebracht, die ihnen vollkommen autonomes Handeln erlauben. [12][18]

Die Simulationsliga besteht hingegen aus mehreren Software-Agenten, die Such- und Rettungsaktionen in einem simulierten Katastrophenszenario durchführen. Ein RoboCupRescue-Simulationsszenario besteht aus der Simulation einer Stadt nach einer Erdbebensituation. Ein Rettungsteam aus Feuerwehrleuten, Polizisten und Ambulanzen muss innerhalb dieser simulierten Welt Such- und Rettungsaktionen durchführen, um den Gesamtschaden der Welt möglichst gering zu halten. Eine große Herausforderung besteht darin, dass die Einsätze der Rettungskräfte in Echtzeit geplant und koordiniert werden müssen. Die Katastrophensituation ist dargestellt durch brennende Gebäude, blockierte Straßen, sowie einer Vielzahl verschütteter und verletzter Zivilisten. Die Aufgabe eines Rettungsteams ist es nun, diese Katastrophensituation einzudämmen, und dabei möglichst viele Zivilisten zu retten. Der Erfolg einer Rettungsaktion wird anhand eines Punktestands bestimmt, der sich aus der Anzahl geretteter Zivilisten und nicht verbrannter Häuser errechnet. Eine Rettungsaktion in der Rescue-Simulation dauert 300 Runden, wobei eine Runde einer Sekunde in Echtzeit entspricht. [15]

In den RoboCup-Wettbewerben variiert ein Rescue-Simulationsszenario in seinem Schwierigkeitsgrad. Dabei wird auch die Anzahl der einzelnen Rettungskräfte festgelegt. Zu Beginn der Wettkämpfe ist die Katastrophensituation noch überschaubar in einer kleinen Stadt dargestellt. Die Zivilisten können sich teilweise noch selbst retten und die Brände sowie verschüttete Gebäude sind meist zugänglich. Jedoch werden beim Fortschreiten der Wettbewerbe die Katastrophenszenarien immer tückischer: große Gebäudeflächen stehen bereits in Flammen, und viele Zivilisten sind tief in eingestürzten Gebäuden verschüttet, zudem sind die Straßen größtenteils blockiert. In derart schwierigeren Szenarien kommt die Schlüsselrolle für einen erfolgreichen Einsatz der Koordination der Rettungsteams zu: Ist beispielsweise die Straße durch eingestürzte Gebäude oder fliehende Zivilisten blockiert, müssen Polizisten die Durchfahrt für Feuerwehr und Ambulanz sicherstellen.

3.2 RoboCupRescue-Simulationssystem

Die RoboCupRescue-Simulation besteht aus zwei Komponenten, einem *Simulator*, der die virtuelle Welt und somit das Einsatzszenario zur Verfügung stellt und den *Rescue-Agenten*, welche in dieser simulierten Welt agieren. Beide Komponenten bilden das *RoboCupRescue-Simulationssystem* (RCRSS).

Der Rescue-Simulator wurde in seiner ersten Version im Juni 1999 veröffentlicht. Es handelte sich um einen Prototyp, der seither immer weiterentwickelt wurde und gegenwärtig in der Version 0.49 zur Verfügung steht. [16]
Um den Entwicklungsprozess voranzutreiben, wird innerhalb der jährlichen RoboCup-Wettbewerbe auch eine Auszeichnung für die Entwicklung der besten Simulatorsoftware vergeben. In den letzten zwei Jahren ging dieser Preis an ein Team aus Freiburg für die Entwicklung eines Feuersimulators, 3D-Betrachters und Zivilistensimulators. [17]
Gerade die Entwicklung von Komponenten für die Simulationssysteme ist von besonderem Interesse für die Öffentlichkeit: Die Feuerwehr in Los Angeles verwendet Teile der aktuell bestehenden RoboCup-Software zur Simulation von Feuer und zur 3D-Visualisierung für das Training von Einsatzleitern. [18]

3.2.1 Architektur des RCRSS

Der architektonische Aufbau des RoboCupRescue-Simulationssystem ist in Abbildung 2 dargestellt. Der Rescue-Simulator (Rescue Simulator) wird in Abbildung 2 schwarz angezeigt. Dieser interagiert mit den verschiedenen Rescue-Agenten (Rescue Agents), die grau abgebildet sind.

Der Rescue-Simulator setzt sich aus folgenden Modulen zusammen [20]:
Kernel, Viewer, Geographical Information System (GIS), Civilian und den Katastrophensimulatoren, bestehend aus Fire Simulator, Collapse Simulator, Blockage Simulator, Traffic Simulator und Miscellaneous Simulator.

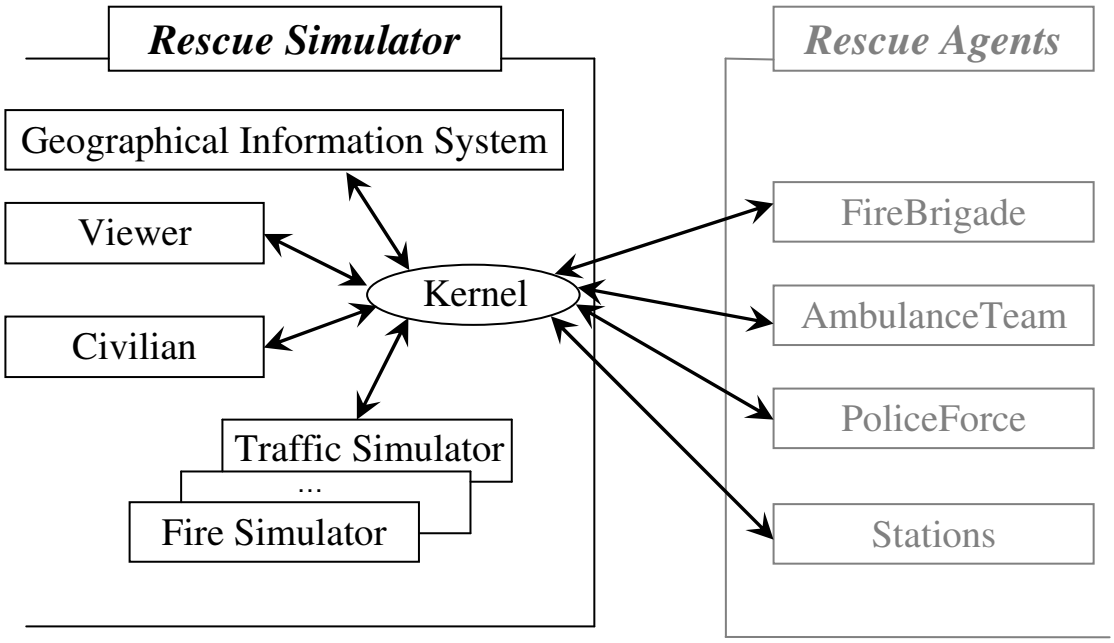


Abbildung 2: Aufbau des RoboCupRescue-Simulationssystem (basierend auf [20])

Das zentrale Element des Rescue-Simulators ist die Kernel-Komponente (Kernel). Der Kernel koordiniert, verwaltet und überwacht den Informationsaustausch zwischen den Simulatorkomponenten. Weiterhin bildet der Kernel die Schnittstelle zur Kommunikation mit den Rescue-Agenten (Rescue Agents).

Die Rescue-Agenten sind heterogene intelligente Agenten, die in der simulierten Welt abhängig von den gegebenen Weltumständen eigene Entscheidungen über ihre Handlungen treffen. Diese sind gesteuert durch Algorithmen in den Programmen, die die Entwickler eines Rescue-Agententeams entwerfen. Die Programme der Agententeamentwickler stellen den Entscheidungs- und Handlungsrahmen für die Rescue-Agenten in der Simulation zur Verfügung. Die Rescue-Agenten werden durch die Client Programme des RoboCupRescue-Simulationssystems dargestellt. Der Kernel des Simulators bildet das Server Programm zu diesen Clients. Er kontrolliert die Handlungen der Agenten, indem zu spät gesendete oder unzulässige Befehle der Agenten verworfen werden. Zu den Rescue-Agenten gehören Feuerwehr- (FireBrigade), Polizei- (PoliceForce) und Ambulanzagenten (AmbulanceTeam) sowie die zugehörigen Stationen (Stations). [20]

Zivilisten gehören nicht zur Gruppe der Rescue-Agenten, da sie vom Simulator gesteuert werden. Ein Agententeamentwickler kann daher keinen Einfluss auf die Verhaltensweisen der Zivilisten nehmen.

Zivilisten werden durch das Civilian-Modul als spezielle Agenten dargestellt. In der virtuellen Welt werden mehrere gleichartige Zivilisten simuliert, die von den Rescue-Agenten gerettet werden sollen, falls sie sich nicht mehr eigenständig fortbewegen können. Solange ein Zivilist in der Lage ist, sich selbstständig zu bewegen, sucht er spezielle Gebäude auf, die in der virtuellen Welt als Schutzgebäude dienen. Diese speziellen Gebäude werden als *Refuges* bezeichnet.

Das GIS (Geographical Information System) Modul liefert die initiale Konfiguration der Welt. Diese Konfiguration enthält die Platzierungen der einzelnen Gebäude und Straßen in der virtuellen Welt, sowie die Positionen aller Agenten und Zivilisten zu Beginn der Rescue-Simulation. Das GIS dokumentiert zudem alle Informationen einer Rescue-Simulation in den Logdateien *rescue.log* und *action.log*.

Für die Auswirkungen der Katastrophensituation auf die Zivilisten ist der Miscellaneous Simulator zuständig. Er sorgt beispielsweise dafür, dass der Gesundheitszustand eines Zivilisten in einem brennenden oder eingestürzten Gebäude entsprechend angepasst wird.

Für die Darstellung der Katastrophenzustände, wie beispielsweise brennende Gebäude oder blockierte Straßen, sind die verschiedenen Katastrophensimulatoren zuständig.

Der Feuersimulator (Fire Simulator) simuliert Brände in Gebäuden, die sich im Laufe der Simulation auf die benachbarten Gebäude ausbreiten und, bei ausbleibender Löschaktivität, die Gebäude völlig zerstören können.

Der Grad der Zerstörung und die Beschaffenheit der einzelnen Gebäude wie beispielsweise Grundfläche, Etagenanzahl, Ausgänge und Bauart werden durch den Kollapssimulator dargestellt. Je nach Beschaffenheit eines Gebäudes ist es robuster oder anfälliger für die verschiedenen Katastropheneinwirkungen. Besteht die Bauart eines Gebäudes beispielsweise aus einem einfachen Holzhaus, ist dieses Gebäude wesentlich schneller vom Feuer zerstört als ein modernes Stahlgebäude.

Ebenso wie bei den Gebäuden existieren auch bei den Straßen unterschiedliche Beschaffenheiten. Der Verkehrssimulator (Traffic Simulator) simuliert daher die Breite der Straße, die Anzahl der befahrbaren Spuren pro Richtung, sowie die Anzahl der Fußgängerspuren in jeder Richtung. Durch diese Beschaffenheiten sind Straßen ebenfalls unterschiedlich anfällig für Zerstörung von herunterstürzenden Gebäudeteilen, oder sich ausbreitendem Feuer.

Durch den Blockadesimulator werden Blockaden auf den zerstörten Straßen verteilt, so dass diese je nach Breite und Anzahl Spuren teils eingeschränkt oder gar nicht befahrbar sind.

Neben den beschriebenen Katastrophensimulatoren sollen zukünftig weitere Simulatoren entwickelt werden. Angedacht sind ein Windsimulator zur Simulation von Windstärke und –Richtung, sowie ein Menschenmassensimulator (*Crowd Simulator*), zur Darstellung einer Menschenmenge aus Zivilisten mit typischen Verhaltensweisen innerhalb von Gruppen wie beispielsweise Massenpanik. Die simulierten Menschenmassen sollen vorrangig an brennenden oder eingestürzten Gebäuden auftauchen, um die Einsätze der Rettungskräfte zu blockieren. [21]
Diese Erweiterungen können ein mögliches Katastrophenszenario noch realistischer darstellen.

Für die Visualisierung der Rescue-Simulation ist das Viewer-Modul zuständig. Der Viewer entspricht einem von T. Morimoto entwickeltem Monitor [24], welcher mittels Computergrafik eine 2D Karte der virtuellen Welt anzeigt. Zudem werden auch die Rescue-Agenten und Zivilisten mit Ihren Bewegungen und sichtbaren Handlungen dargestellt.
Der Monitor kann sowohl *Online* als auch *Offline* betrieben werden. Im *Online*-Modus erhält der Monitor Informationen über die Bewegungen und Handlungen der Agenten vom Kernel und stellt diese in Echtzeit dar.
Im *Offline*-Modus benutzt der Monitor die vom GIS erzeugte Logdatei *rescue.log*, wobei lediglich die Bewegungen der Agenten als Änderungen der Positionen zwischen zwei Zeitschritten visualisiert werden.
Abbildung 3 zeigt die Darstellung des im Simulator der Version 0.48 enthaltenen Monitors im *Online*-Modus.

Abbildung 3: Monitor der Simulatorversion 0.48 im *Online*-Modus

Die Rescue-Agenten werden durch ausgefüllte Kreise kenntlich gemacht. Grüne Kreise stellen Zivilisten dar, rote Kreise stehen für Feuerwehragenten, blaue Kreise entsprechen Polizeiagenten und weiße Kreise stellen Ambulanzagenten dar. Je dunkler die Farbschattierungen der Kreise, desto schlechter ist der Gesundheitszustand der Agenten bzw. Zivilisten.
Die einzelnen Aktionen werden veranschaulicht, indem die ausführenden Agenten durch einen Ring markiert werden (vgl. Abbildung 3 (a)). Der Löschvorgang eines Agenten wird durch eine blaue Linie angezeigt, die vom ausführenden Agenten zum gelöschten Gebäude verläuft (vgl. Abbildung 3 (b)).

Neben des im Simulator enthaltenem Monitors existieren weitere Entwicklungen, die den Ablauf einer Rescue-Simulation veranschaulichen können. Beispielsweise wurde 2005 ein 3D-Monitor von dem Freiburger Team *ResQ* entwickelt, welcher neben einer 3D-Visualisierung weitere Spezialeffekte, wie beispielsweise die Darstellung der Rauchentwicklung eines brennenden Gebäudes, enthält. [27]
Die Visualisierung des 3D-Monitors ist in Abbildung 4 dargestellt.

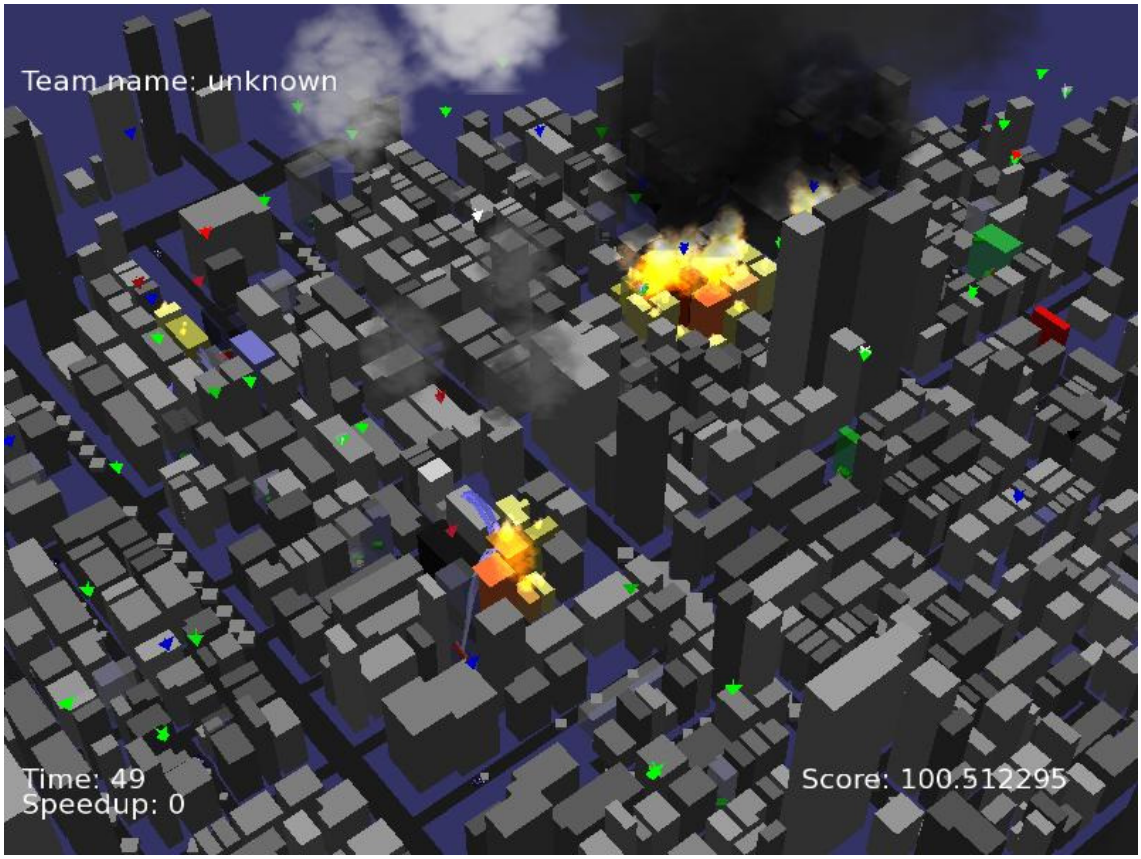


Abbildung 4: 3D-Monitor des Freiburger Teams *ResQ*

3.2.2 Ablauf der Simulation

In einer Initialphase vor einer Rescue-Simulation werden zunächst alle Module in das RCRSS wie folgt integriert (vgl. Abbildung 5) [15]:

1. Der Kernel verbindet sich mit dem GIS Modul (1.1), welches dem Kernel die initiale Konfiguration der virtuellen Welt bereitstellt (1.2).
2. Die Katastrophensimulatoren sowie vorhandene Monitore verbinden sich mit dem Kernel (2.1), der seinerseits die initiale Konfiguration als Anfangsbedingungen an alle Module weiterleitet (2.2).
3. Die RoboCupRescue-Agenten (RCR-Agenten) sowie die Zivilisten verbinden sich mit dem Kernel (3.1). Je nach Möglichkeit der Wahrnehmung der unterschiedlichen Agententypen (vgl. Kapitel 3.3) erhalten diese ebenfalls die Informationen über den Anfangszustand der virtuellen Welt (3.2).

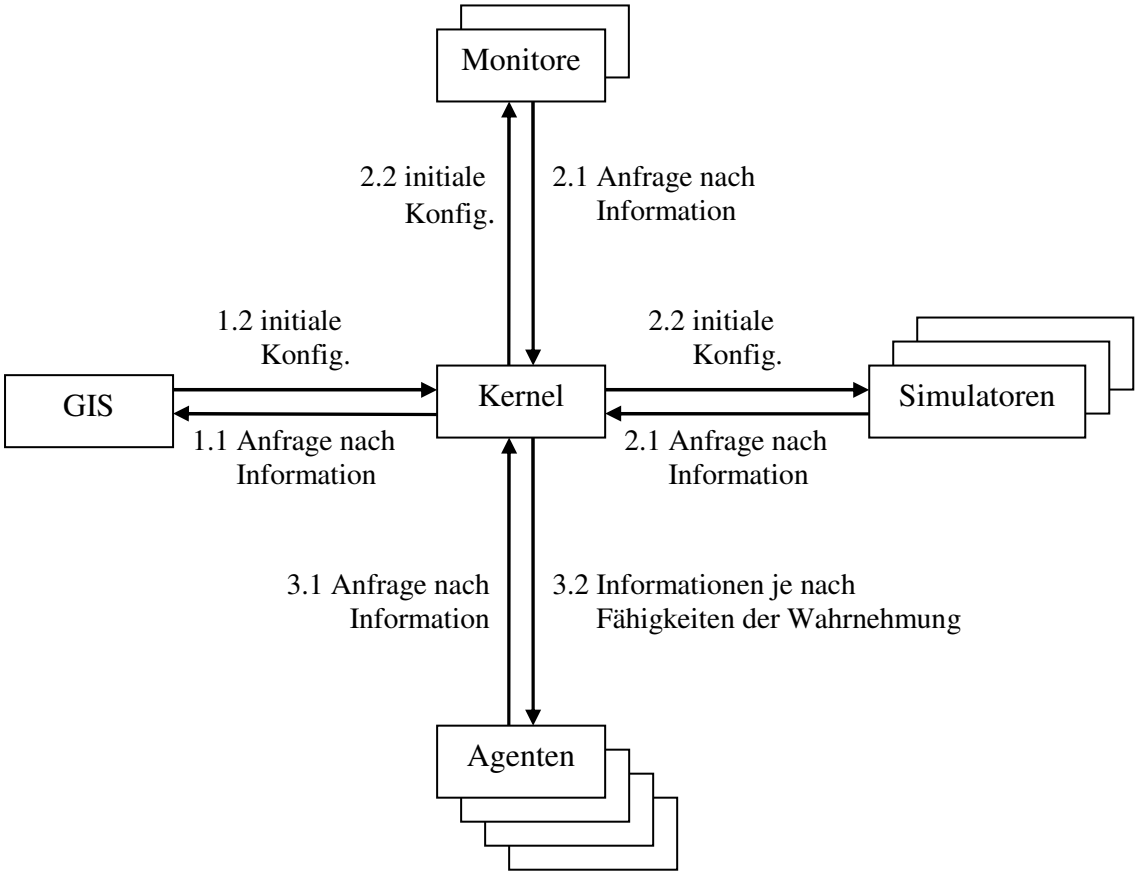


Abbildung 5: Initialisierung der Rescue-Simulation (basierend auf [15])

Nachdem alle Agenten - RCR-Agenten und Zivilisten - erfolgreich im RCRSS integriert sind, beginnt die Simulation. Der Simulationsverlauf geschieht durch Wiederholung des nachfolgend aufgeführten Zyklus (vgl. Abbildung 6). In der ersten Runde der Simulation werden Schritt 1. und 2. übersprungen. [15]

1. Der Kernel übermittelt jedem Agenten die individuellen, visuellen Informationen.
2. Jeder Agent antwortet dem Kernel mit einem individuellen Aktionsbefehl.
3. Der Kernel leitet die gesammelten Aktionsbefehle der Agenten an alle Simulatoren weiter.
4. Diese aktualisieren ihre Daten entsprechend den Auswirkungen der Aktionen der einzelnen Agenten und übermitteln dem Kernel ihre aktuellen Daten.
5. Der Kernel fasst die erhaltenen Daten zusammen und sendet sie an das GIS und die integrierten Monitore.
6. Der Kernel erhöht die Simulationszeit um eine Runde.

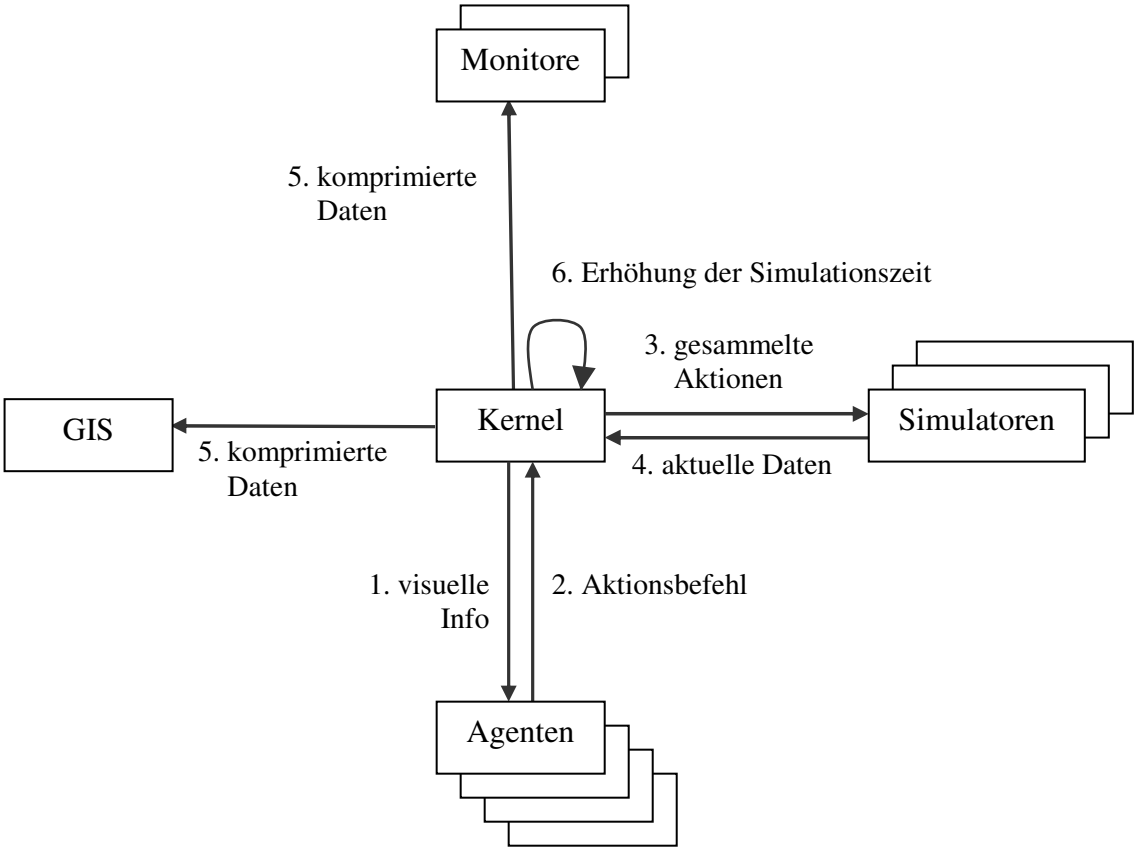


Abbildung 6: Prozess der Rescue-Simulation (basierend auf [15])

Eine Runde der Simulation entspricht einer Sekunde in Echtzeit. Der Kernel wartet jeweils eine halbe Sekunde auf die Aktionsbefehle der Agenten (Schritt 2) und die aktualisierten Daten der Katastrophensimulatoren (Schritt 4). Das bedeutet, dass alle Agenten ihre Entscheidung über ihre Handlungen innerhalb einer halben Sekunde treffen müssen. [15]

3.2.3 Bestandteile der Welt und Eigenschaften

Die simulierte Welt besteht aus Objekten, die durch eine neunstellige Identifikationsnummer (Id) eindeutig identifiziert werden können. Weiterhin besitzen die Objekte zweidimensionale Positionskoordinaten (x,y). Innerhalb der simulierten Welt existieren bewegliche (MovingObject) und unbewegliche Objekte (MotionlessObject) (vgl. Abbildung 7).

Zu den unbeweglichen Objekten zählen:

- Gebäude (Building),
- Straße (Road),
- Knotenpunkt (Node),
- Fluss (River),
- Flussknoten (RiverNode).

Die beweglichen Objekte bestehen aus:

- Zivilist (Civilian),
- Ambulanz (AmbulanceTeam),
- Feuerwehr (FireBrigade),
- Polizei (PoliceForce),
- Auto (Car).

Bisherige Simulatorversionen verwenden allerdings keine Objekte der Klasse River und RiverNode. Des Weiteren fehlen auch Objekte der Klasse Car. Diese Klassen sind seit der ersten Simulatorversion vorhanden, wurden aber bisher noch nicht benutzt. Möglicherweise werden sie in zukünftigen Entwicklungen neuer Simulatoren entweder benutzt oder endgültig entfernt.
Aufgrund dieser Tatsache werden die genannten Objekte in weiteren Betrachtungen nicht mehr aufgeführt.

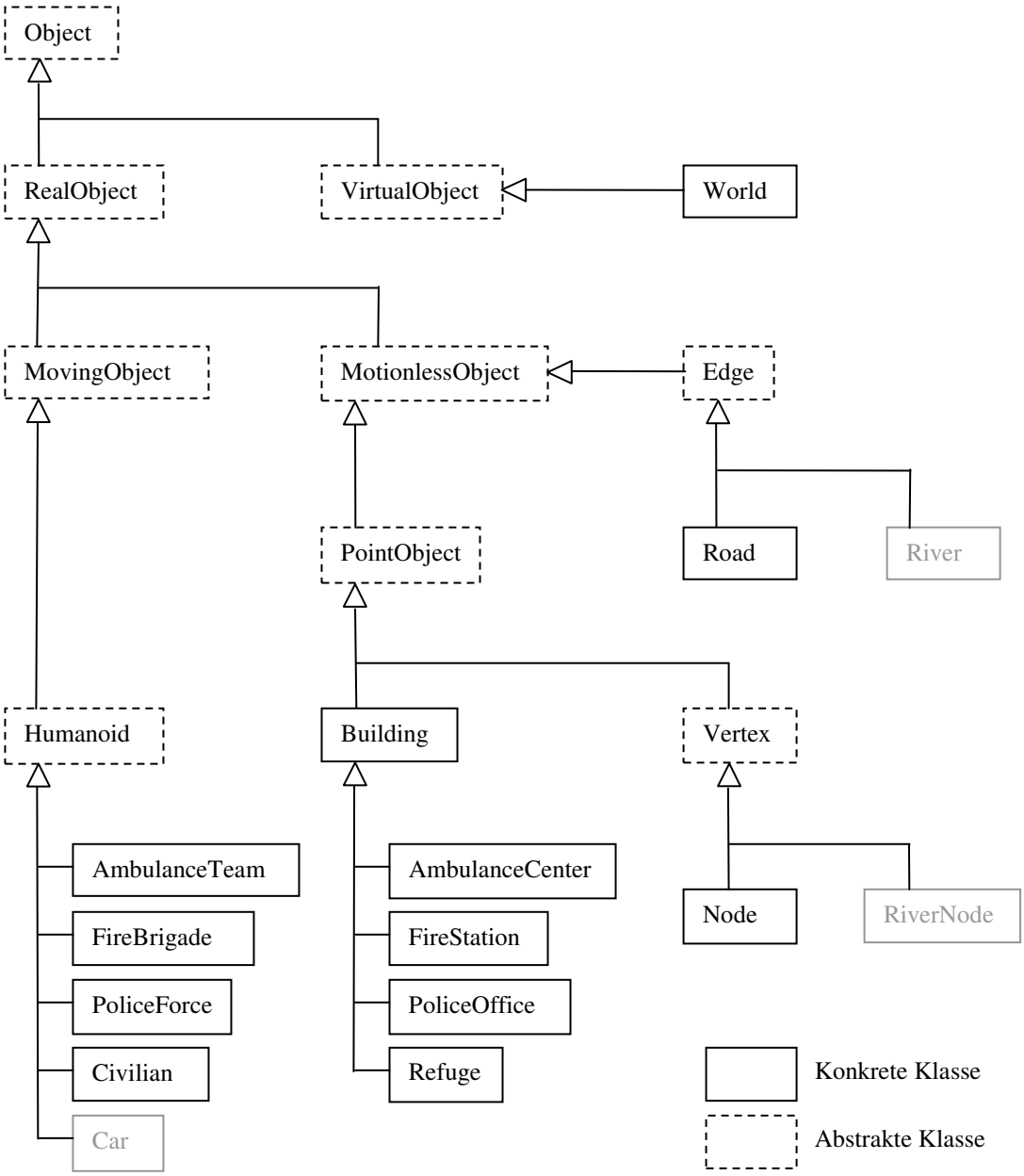


Abbildung 7: Klassenhierarchie der Weltobjekte

Alle Weltobjekte sind charakterisiert durch objektspezifische Eigenschaften. Zum einen hat jedes unbewegliche Objekt statische, geometrische Eigenschaften, wie beispielsweise Größe und Form, sowie dynamische Eigenschaften, die sich im Laufe der Simulation verändern. Die beweglichen Objekte besitzen nur dynamische Eigenschaften. Tabelle 1 zeigt die dynamischen Eigenschaften aller Objekte der Rescue-Simulation.

	Objekte	**Dynamische Eigenschaften**
Unbewegliche Objekte	Gebäude	Ausmaß des Feuers (*Fieryness*) Ausmaß der Zerstörung (*Brokeness*)
	Straße	Grad der Blockiertheit (*Block*)
	Knoten	- keine-
Bewegliche Objekte	Zivilist	Gesundheitszustand (*HealthPoints*, *HP*) Grad der Verschüttung (*Buriedness*)
	Ambulanz	Gesundheitszustand (*HealthPoints*, *HP*) Grad der Verschüttung (*Buriedness*)
	Feuerwehr	Gesundheitszustand (*HealthPoints*, *HP*) Grad der Verschüttung (*Buriedness*) Wasservorrat (*WaterQuantity*)
	Polizei	Gesundheitszustand (*HealthPoints*, *HP*) Grad der Verschüttung (*Buriedness*)

Tabelle 1: Dynamische Eigenschaften der Weltobjekte (basierend auf [20])

Die Werte der dynamischen Eigenschaften werden durch Integerzahlen repräsentiert. Einen Überblick über die verwendeten Wertebereiche der letzten RoboCupRescue-Wettbewerbe zeigt Tabelle 2.

Dyn. Eigen-schaften	Werte-bereiche	Bedeutung
Fieryness	0-5	Ausmaße des Feuers eines Gebäudes 0: Gebäude brennt nicht 1-4: leichtes bis starkes Brennen, aber löschbar 5: Gebäude abgebrannt und total zerstört
Brokeness	0-100	Ausmaß der Zerstörung eines Gebäudes 0: kein Schaden 25: teilweise beschädigt 50: halb zerstört 100: total zerstört
Block		Breite der Straße, die nicht befahrbar ist, Einheit: Mm
Healthpoints	0-10000	Notwendigkeit zur medizinischen Hilfe eines Agenten 10000: Agent ist gesund 9999-1: Agent hat wenige bis erhebliche Verletzungen 0: Agent ist tot
Buriedness	0-60	Tiefe der Verschüttung eines Agenten (z.B. in einem eingestürzten Gebäude) 0: nicht verschüttet 60: es sind 60 Agenten notwendig, um den Agenten in einem Zyklus auszugraben, oder ein Agent benötigt 60 Zyklen, um den verschütteten Agenten auszugraben
WaterQuantity	0-15000	Wasservorrat eines Feuerwehragenten 0: Wasservorrat aufgebraucht 15000: maximaler Wasservorrat

Tabelle 2: Wertebereiche der dynamischen Eigenschaften (basierend auf [20])

3.3 Fähigkeiten der Rescue-Agenten

Das gemeinsame Ziel der RoboCupRescue-Agenten ist es, möglichst viele Zivilisten zu retten und brennende Gebäude zu löschen. Um dieses Ziel zu erreichen, haben Rescue-Agenten unterschiedliche *Fähigkeiten*, die sich aus *Wahrnehmung* und *Handlung* zusammensetzen.
Die Aufgabe der Polizeiagenten ist die Räumung von Blockaden auf den Straßen, um die Passierbarkeit für Ambulanz- und Feuerwehragenten zu gewährleisten. Die Feuerwehragenten haben die Funktion brennenden Gebäude zu löschen, sobald sie sich innerhalb eines definierten Umkreises um den Brandherd befinden. Die Ambulanzagenten sind in der Lage verletzte Zivilisten zu bergen und diese in die *Refuges* zu befördern. *Refuges* sind vergleichbar mit Hospitälern, hier werden verletzte Zivilisten und auch verletzte Agenten geheilt. Die *Refuges* stellen ebenfalls den Wasservorrat für die Feuerwehragenten bereit, die nur eine begrenzte Kapazität an Wasser zum Löschen speichern können und somit regelmäßig die *Refuges* zum Nachtanken aufsuchen müssen.
Zu den Rescue-Agenten gehören auch die entsprechenden Stationen. Die Stationen dienen zur Unterstützung der Kommunikation und Koordination der einzelnen Rescue-Agenten. Sie werden wie auch *Refuges* in Form spezieller Gebäude dargestellt.
Durch die heterogene Architektur der Rescue-Agenten (vgl. Kapitel 3.2.1) existieren verschiedene *Typen* von Agenten, die unterschiedliche Fähigkeiten besitzen.
Die verschiedenen *Agententypen* eines Rescue-Agententeams, sowie deren Fähigkeiten sind in Tabelle 3 dargestellt.
Anhand ihrer Wahrnehmungen müssen die Rescue-Agenten entscheiden, welche Handlungen sie ausführen. Dieser Entscheidungsprozess muss in Echtzeit ausgeführt werden. In einer Runde der Rescue Simulation, die eine Sekunde in Echtzeit beträgt, hat ein Rescue-Agent nur eine halbe Sekunde Zeit (vgl. Kapitel 3.2.2), um auf die wahrgenommene Katastrophensituation zu reagieren.
Die Zivilisten besitzen ebenfalls die Fähigkeit zur Wahrnehmung (*Sense, Hear*) und Handlung (*Say, Move, Rest*). Allerdings werden diese vom Simulator gesteuert und können nicht durch die Programmierung eines Agententeams verändert werden (vgl. Kapitel 3.2.1).
Die möglichen Handlungen der verschiedenen Agententypen bestehen aus einzelnen *Aktionen*. Diese untergliedern sich in *Kommunikationsaktionen* und *Handlungsaktionen*. Die Bedeutung dieser Aktionen ergibt sich aus Tabelle 4.
In einer Runde der Rescue-Simulation kann ein Rescue-Agent jeweils nur eine Handlungsaktion sowie eine beschränkte Anzahl von Kommunikationsaktionen (vgl. Kapitel

3.4) ausführen. Je nach Agententyp können nur bestimmte Aktionen ausgewählt werden (vgl. Tabelle 3).

Agententyp	**Fähigkeiten**	
	Wahrnehmung	**Handlung**
AmbulanceTeam	*Sense, Hear*	*Say, Tell, Move, Rescue, Load, Unload, Rest*
FireBrigade	*Sense, Hear*	*Say, Tell, Move, Extinguish, Rest*
PoliceForce	*Sense, Hear*	*Say, Tell, Move, Clear, Rest*
AmbulanceCenter	*Sense, Hear*	*Tell, Rest*
FireStation	*Sense, Hear*	*Tell, Rest*
PoliceOffice	*Sense, Hear*	*Tell, Rest*

Tabelle 3: Fähigkeiten der Rescue-Agenten [15]

	Aktionen	**Bedeutung**
Kommunikations-aktionen	*Say*	Auditive Informationsübertragung
	Tell	Funkübertragung von Nachrichten
Handlungs-aktionen	*Rest*	Verweilen auf der aktuellen Position
	Move	Bewegung zu einer anderen Position
	Rescue	Ausgraben eines Zivilisten
	Load	Aufladen eines Zivilisten
	Unload	Abladen eines Zivilisten
	Extinguish	Löschen eines brennenden Gebäudes
	Clear	Freiräumen einer blockierten Straße

Tabelle 4: Bedeutung der Aktionen

3.4 Kommunikation der Agenten

Für einen erfolgreichen Einsatz in einem Rescue-Simulationsszenario ist eine effiziente Koordination der einzelnen Agententypen erforderlich. Aufgrund der beschränkten visuellen Wahrnehmung auf ein kleines Umfeld der virtuellen Welt von 10 Metern, kann eine Wissensteilung über weite Distanzen nur über Kommunikation erfolgen. Die Kommunikation der Agenten unterliegt allerdings einschränkenden Rahmenbedingungen. Innerhalb eines Agententeams können Agenten desselben Typs direkt miteinander kommunizieren, während typenübergreifend keine direkte Kommunikation möglich ist. In diesem Fall kann eine Kommunikation nur über die Stationen erfolgen. Im Regelfall gibt es zu jedem Agententyp eine Station, die die Nachrichten dieses Agententyps empfangen kann. Weiterhin empfängt jede Station die Nachrichten aller anderen Stationen. Das variierende Rescue-Simulationsszenario kann, wie es in der Realität ebenfalls möglich ist, den Ausfall einer, mehrer oder sogar aller Stationen simulieren, und damit die Kommunikation zwischen den unterschiedlichen Agententypen unterbrechen. Auf Grund solcher tief greifender Einschränkung der Kommunikation sind Maßnahmen bzw. Mechanismen erforderlich die es Agenten ermöglichen, trotz widriger Rahmenbedingungen die Intentionen anderer Agenten erkennen zu können.

Für die Kommunikation existieren die Kommunikationsaktionen *say* und *tell*, die zum Nachrichtenaustausch zwischen Agenten sowie Stationen vorgesehen sind. Die Kommunikation unterliegt dabei folgenden Regeln.

Sendevorgänge [15]:

- Alle Nachrichten sind Broadcasts.
- Die Länge der mit den Kommunikationsbefehlen *say* und *tell* gesendeten Nachrichten ist auf maximal 256 Byte pro Nachricht beschränkt.
- Der Befehl *tell* kann nicht von Zivilisten gesendet werden.
- Alle übrigen Agententypen dürfen in einer Runde vier *tell* Nachrichten verschicken.
- Alle Stationen dürfen in einer Runde zweimal so viele *tell* Nachrichten verschicken wie Agenten ihres Typs in der Simulation vorhanden sind.
- Der Befehl *say* kann von jedem Agenten und jeder Station in einer Runde einer Rescue-Simulation nur einmal gesendet werden.

Empfangsvorgänge [15]:

- Alle Nachrichten kommen erst mit einer Runde Verzögerung beim Empfänger an.
- Die Auswahl der Nachrichten kann nur anhand der Id des Absenders erfolgen ohne den Betreff der Nachricht zu kennen.
- Zivilisten können keine *tell* Nachrichten empfangen.
- Alle Agenten, außer Zivilisten, können in einer Runde vier *tell* Nachrichten empfangen.
- Alle Stationen können zweimal so viele *tell* Nachrichten empfangen wie Agenten ihres Typs in der Simulation vorhanden sind.
- Eine von einem Agenten gesendete *tell* Nachricht kann nur von Agenten desselben Typs empfangen werden, sowie von der zugehörigen Station.
- Eine von einer Station versendete *tell* Nachricht kann von allen Stationen sowie von den zugehörigen Agenten empfangen werden.
- Der Befehl *say* ist auf einen Hörradius von 30 Metern beschränkt, das heißt, nur Agenten und Stationen, die sich innerhalb dieses Radius befindet, können eine mit *say* verfasste Nachricht empfangen.

Teil I: Zusammenfassung

Im ersten Teil dieser Arbeit wurden einführend das Forschungsgebiet und der Anwendungsbereich dieser Diplomarbeit dargestellt.

Die vorliegende Arbeit ist in das Forschungsgebiet der Künstlichen Intelligenz einzuordnen. Daher wurden zunächst in Kapitel 2 die themenrelevanten Begriffe definiert, die im zweiten Teil der Arbeit im Umfeld des zu entwickelnden Werkzeugs verwendet werden. Eine wesentliche Rolle wird dabei den Begriffen *Plan* und *Planerkennung* zukommen.
Weiterhin wurde das Konzept zur Planerkennung von Avrahami-Zilberbrand und Kaminka vorgestellt, welches die grundlegende Idee zu dieser Diplomarbeit lieferte. Dieses Konzept basiert auf der Verwendung zweier Datenstrukturen, einer *Planbibliothek* und eines Entscheidungsbaums (*Feature Decision Tree*), die das Verfahren der Planerkennung gegenüber bisheriger Konzepte erleichtern. Die verwendeten Datenstrukturen werden auch im zweiten Teil wieder aufgegriffen, da diese die Grundlage für die Entwicklung des Analysewerkzeugs innerhalb der Rescue-Simulation bilden.

Die RobocupRescue-Simulation stellt das Anwendungsgebiet dieser Diplomarbeit dar. Eine ausführliche Beschreibung erfolgte innerhalb des dritten Kapitels. Zu Beginn wurde ein Überblick zum Aufbau der RobocupRescue-Simulation gegeben. Weiterhin wurden die innerhalb der Rescue-Simulation agierenden Einheiten (Agenten) mit ihren Fähigkeiten vorgestellt, sowie ein Überblick über den Ablauf einer Rescue-Simulation gegeben. Es wurden die einschränkenden Rahmenbedingungen erörtert, die durch das vorhandene Kommunikationssystem innerhalb der Rescue-Simulation auftreten.

Der erste Teil liefert damit die Ausgangsbasis zu den weiteren Ausführungen, die sich mit der Definition und Umsetzung der Anforderungen an das zu entwickelnde Werkzeug beschäftigen.

II Anforderungsdefinition und Realisierung

Der zweite Teil der Arbeit beschäftigt sich mit der Entwicklung und Realisierung des Analysewerkzeugs für die RoboCupRescue-Simulation. Dabei werden in Kapitel 4 zunächst die Anforderungen an das Analysewerkzeug definiert. Weiter werden zur Abgrenzung gegenüber bisheriger Entwicklungen die Stärken und Vorteile des Analysewerkzeugs erörtert. Darauf aufbauend wird die Architektur des Werkzeugs dargestellt. Anschließend wird innerhalb des fünften Kapitels die Arbeitsweise der beinhalteten Komponenten detailliert beschrieben.
Die umgesetzten Methoden zur Planerkennung sind in Kapitel 6 ausführlich dargestellt. Es folgt die Vorstellung von ausgeführten Experimenten und deren Ergebnisse innerhalb von Kapitel 7.
Abschließend wird der zweite Teil zusammengefasst.

4 Definition und Architektur

In diesem Kapitel werden aus Sicht der angestrebten Zielsetzung der Analyse und Planerkennung zunächst die Anforderungen an das Analysewerkzeug - im Folgenden als *Rescueanalyser* bezeichnet - definiert. Eine Abgrenzung gegenüber bisherigen Werkzeugentwicklungen soll die Neuartigkeit des Ansatzes dieser Arbeit belegen. Ergänzend wird die gewählte Architektur des Analysewerkzeugs dargestellt.

4.1 Anforderungen und Ausgangsbasis

Im Rahmen dieser Diplomarbeit soll ein Analysewerkzeug für die RoboCupRescue-Simulation entwickelt werden. Ausgehend von der Zielsetzung einer Analyse und Planerkennung muss das Werkzeug die nachfolgend definierten Anforderungen erfüllen:
Das Analysewerkzeug soll Pläne von Agententeams erkennen. Dabei soll es unabhängig von dem jeweils agierenden Agententeam arbeiten. Dazu muss zunächst analysiert werden, welche Pläne in der Rescue-Simulation existieren. Diese Pläne sollen in einer Planbibliothek archiviert werden.
Die Planerkennung soll transparent in mehreren Schritten ablaufen.
Die zugrunde liegenden Daten sollen verständlich aufbereitet und ausgegeben werden. Damit können diese Daten zur vergleichenden Analyse von Agententeams aber auch zum Debugging des eigenen Agententeams herangezogen werden.
Weiterhin bilden die aufbereiteten Daten die Basis für den Planerkennungsalgorithmus. Dieser Algorithmus soll sich an dem Konzept von Dorit Avrahami-Zilberbrand und Gal A. Kaminka [3] orientieren, sofern es die Gegebenheiten in der RoboCupRescue-Simulation erlauben.

Der in dieser Diplomarbeit umgesetzte Lösungsansatz basiert auf dem Rescue-Simulator der Version 0.48. Der Simulator erzeugt Logdateien, die unabhängig vom agierenden Agententeam immer im gleichen Format gespeichert werden. Die Logdateien enthalten alle Informationen, die während des Ablaufs einer Rescue-Simulation zwischen dem Simulator und den Rescue-Agenten ausgetauscht werden (vgl. Kapitel 3.2.2).

4.2 Bisherige Entwicklungen

Das im Rahmen dieser Diplomarbeit entwickelte Analysewerkzeug zeichnet sich durch seine Agententeam-unabhängige Anwendbarkeit aus. Dies bedeutet zunächst, dass das Werkzeug für jedes Agententeam eingesetzt werden kann, welches mit dem Rescue-Simulator Version 0.48 läuft. Agententeam bedingte Anpassungen sind nicht erforderlich. Weiterhin liefert das Werkzeug Daten in einem vom Agententeam unabhängigen Format. Dies hat den Vorteil, dass Daten verschiedener agierender Teams direkt und ohne weitere Analyse verglichen werden können.

Für die Analyse im Bereich der RoboCupRescue-Simulation wurden bislang nur agententeamspezifische Werkzeuge entwickelt, die auf Basis der vom Agententeam erzeugten Logdateien arbeiten (vgl. [17]). Solche Daten haben keine vergleichbare Struktur.
Eine Verwendung dieser Werkzeuge für verschiedene Agententeams ist entweder ausgeschlossen oder erfordert einen sehr hohen Anpassungsaufwand. Eine vergleichende Analyse ist auf Basis solcher heterogener Daten nahezu unmöglich.

Neben der Analyse ist die Planerkennung eine der wesentlichen Aufgaben des zu entwickelnden Werkzeugs. Zur Planerkennung gibt es bis dato keine Entwicklungen im Bereich der RoboCupRescue-Simulation.

4.3 Architektur und Arbeitsweise des Rescueanalyser

Der Rescueanalyser besteht aus mehreren Komponenten, die verschiedene Aufgabenbereiche abdecken. Die Architektur des Rescueanalysers ist in Abbildung 8 dargestellt.

Als Eingabe benötigt der Rescueanalyser die Logdateien rescue.log und action.log eines erfolgreichen Rescue-Simulationsdurchlaufs. Diese Dateien werden unabhängig von den Rescue-Agententeams (Rescue-Agenten) erzeugt. Die Analysekomponente (Analyser) ist verantwortlich für die Informationsgewinnung aus den Logdateien rescue.log und action.log. Sie besteht aus zwei weiteren Modulen, einem Parser und einem Generator. Der Parser bildet die Schnittstelle zum Rescue-Simulator, über welche die Logdateien der Rescue-Simulation gelesen und bearbeitet werden. Der Generator erzeugt eine graphische (grafische Ausgabe) sowie eine textbasierte (textbasierte Ausgabe) Repräsentation der Information aus den Logdateien.

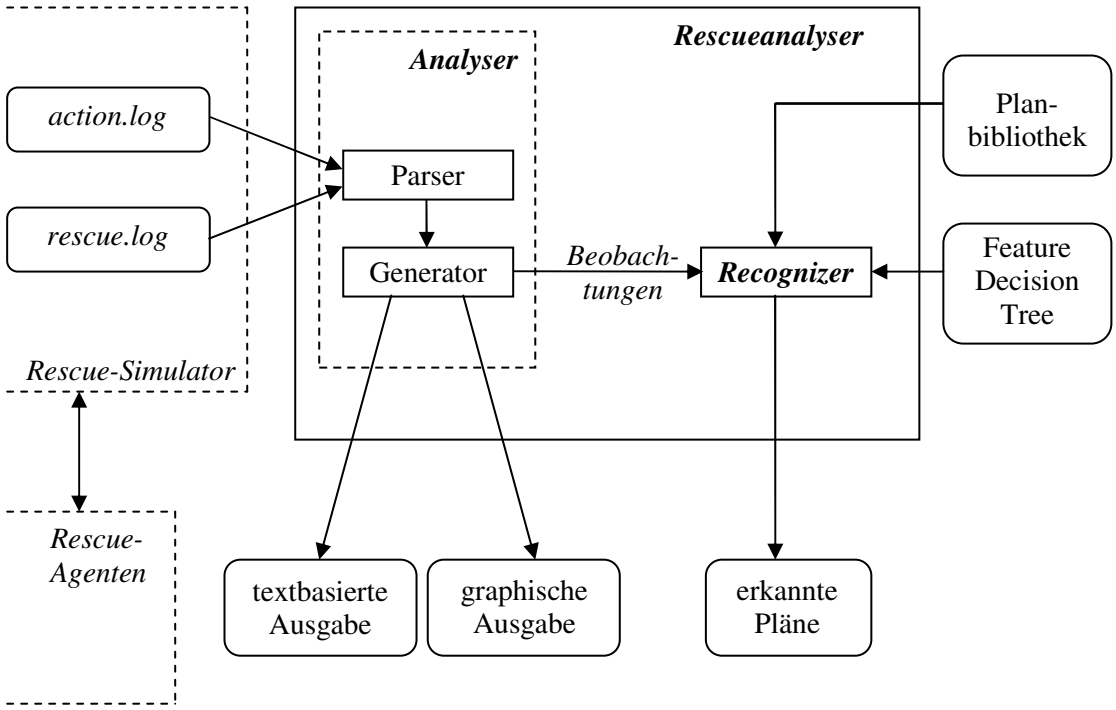


Abbildung 8: Architektur des Rescueanalysers

Weiterhin liefert der Generator die Eingabe für eine weitere Komponente, den Recognizer. Der Recognizer ist für den Planerkennungsprozess verantwortlich. Die Eingabe besteht aus den Informationen (Beobachtungen), welche bei der Analyse der Logdateien erzeugt werden.

Um eine Planerkennung mittels dieser Informationen durchführen zu können, benötigt die Recognizer-Komponente zwei weitere Eingaben: die Planbibliothek und den Feature Decision Tree. Die Planbibliothek enthält bekannte Pläne, die in der Rescue-Simulation auftreten können. Diese Pläne, welche im Wesentlichen aus möglichen Aktionssequenzen der Agenten bestehen, werden einmal spezifiziert, können aber bei Bedarf erweitert werden. Die Spezifikation der Planbibliothek wird ausführlich in Kapitel 6.2 erläutert.
Der Feature Decision Tree erlaubt die effiziente Zuordnung der Informationen aus den Logdateien zu Planschritten der Planbibliothek. Diese Datenstruktur wird in Kapitel 6.3 detailliert erörtert.

Die Aufgabe des Recognizer-Moduls ist es nun, die aus der Rescue-Simulation durch die Analysekomponente aufbereiteten Daten (Beobachtungen) mit Plänen der Planbib-

liothek unter Verwendung des Feature Decision Tree zu vergleichen, um zuverlässig Pläne zu erkennen.
Die Ausgabe des Recognizer-Moduls sind erkannte Pläne der Planbibliothek. Eine ausführliche Beschreibung der Ausgabe erfolgt in Kapitel 6.5.

5 Detaillierte Konzeption

In diesem Kapitel werden die Eingaben, Ausgaben, sowie die Komponenten des Analyser-Moduls und ihre Arbeitsweisen beschrieben.

5.1 Analyse der Eingabe-Dateien

Der Simulator erzeugt während eines Rescue-Simulationsdurchlaufs die Logdateien *rescue.log* und *action.log*. Sie dienen unter anderem dem Simulator zur Wiederholung der Simulation in einem *Offline*-Modus des Viewer-Moduls (vgl. Kapitel 3.2.1). Die Logdateien beinhalten alle Informationen, die während eines *Online*-Simulationsdurchlaufs zwischen den Simulatorkomponenten ausgetauscht werden (vgl. Kapitel 3.2.2), um eine Wiederholung der kompletten Simulation ohne die entsprechenden Simulatorkomponenten zu ermöglichen. Zur Verwendung dieser Dateien als Grundlage für den Analyse- und Planerkennungsprozess ist zunächst eine genaue Betrachtung der beinhalteten Informationen aus beiden Dateien notwendig.

Die Datei *rescue.log* enthält allgemeine Informationen (General Info) über einen Rescue-Simulationsdurchlauf, beispielsweise die Länge der Simulation, das heißt die Anzahl der durchlaufenen Zyklen, oder die Größe des Monitors. Weiterhin werden spezielle Parameter der Rescue-Objekte, wie beispielsweise die Löschkraft und der maximale Wasservorrat eines Feuerwehragenten, dokumentiert. Diese Parameter ändern sich im Laufe eines Simulationsdurchlaufs nicht und werden daher nur einmal zu Beginn der Simulation gespeichert.
Darüber hinaus enthält die Datei *rescue.log* alle Daten über die in der Welt vorhandenen Objekte. Diese Objektdaten können sich im Laufe der Simulation verändern, und werden daher für jeden Zeitschritt (TimeStep) der Simulation erneut abgespeichert.
Eine abstrakte Darstellung der gespeicherten Informationen der Datei *rescue.log* ist in Abbildung 9 veranschaulicht.

Während die Datei *rescue.log* die allgemeinen Informationen der Welt sowie die Daten aller Objekte beinhaltet, werden in der Datei *action.log* nur Informationen gespeichert, die die gesendeten Aktionen der Agenten betreffen. Die Informationen der Datei *action.log* bestehen aus dem Aktionstyp (ActionType), der Id des ausführenden Agenten (AgentId), sowie Informationen über die von einer Aktion betroffenen Objekte. Diese

Objektinformationen bestehen aus den eindeutigen Id-Werten der betroffenen Objekte (Obj.Id).
In jedem Zeitschritt der Simulation werden neue Aktionen gesendet. Daher erfolgt die Speicherung der Daten in der Datei *action.log* nach Zeitschritten (TimeStep) der Simulation.
Eine abstrakte Darstellung der innerhalb der Datei *action.log* gespeicherten Informationen zeigt Abbildung 10.

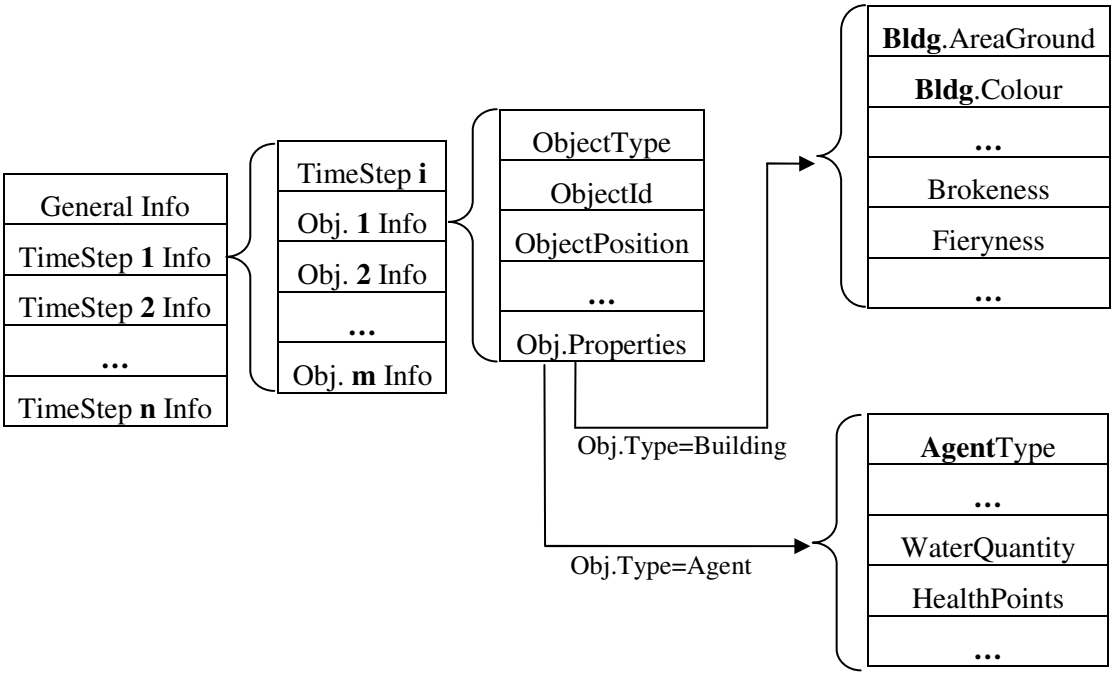


Abbildung 9: Abstrakte Darstellung des Inhalts der Datei *rescue.log*

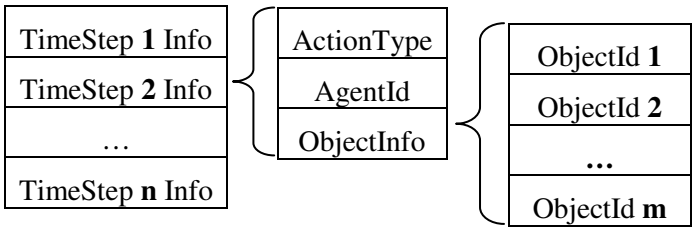


Abbildung 10: Abstrakte Darstellung des Inhalts der Datei *action.log*

Der Rescue-Simulator speichert die Daten der Logdateien *rescue.log* und *action.log* in einem für den Menschen nicht lesbaren Maschinencode.
Maschinencode (Maschinensprache) besteht aus binär kodierten, Prozessor-spezifischen Befehlen. [23]
Die Logdateien werden innerhalb des Rescue-Simulators nur vom *Viewer*-Modul zur Wiederholung von Simulationsdurchläufen im *Offline*-Modus benutzt, daher ist es innerhalb des Rescue-Simulationssystems nicht notwendig, die Dateien in einem für Menschen lesbarem Format darzustellen. Die Speicherung der Logdateien in einem Maschinenformat ermöglicht die unmittelbare Verarbeitung der Daten durch den *Viewer*.

5.2 Funktionsweise des Parsers

Nachdem die in den Logdateien *rescue.log* und *action.log* enthaltenen Strukturen und Informationen bekannt sind, können die relevanten Daten extrahiert werden. Für diese Informationsgewinnung aus den Logdateien ist die Parser-Komponente des Analyser-Moduls zuständig. Mittels des Parsers werden die Daten der kodierten Logdateien als Datenstrom eingelesen. Innerhalb des Datenstroms einer Logdatei werden Sequenzen erkannt, die definierten Werten entsprechen. Dies ermöglicht die Rekonstruktion der ursprünglich gesendeten Befehle der Simulatorkomponenten. Mittels Zuordnungstabellen, die innerhalb des Rescueanalysers realisiert sind, können die erkannten Werte korrekt interpretiert werden. Tabelle 5 zeigt die verwendeten Zuordnungen für Aktionsbefehle. Dargestellt sind die Hexadezimalwerte, die den möglichen Aktionen der Rescue-Agenten entsprechen, die in der RoboCupRescue-Simulation ausgeführt werden können (vgl. Tabelle 2). Wird beispielsweise im Datenstrom der Datei *action.log* eine Sequenz gefunden, die dem Hexadezimalwert „0x80“ entspricht, ist damit die Aktion *Move* (AK_MOVE) kodiert, die von einem Agenten an den Kernel (AK) geschickt wurde (vgl. Kapitel 3.2.2).

Eine weitere Zuordnungstabelle enthält die Eigenschaften aller Weltobjekte, die innerhalb der Datei *rescue.log* kodiert sind. Hier entsprechen die Bytesequenzen vordefinierten Integerwerten, mit denen die Objekteigenschaften repräsentiert werden (vgl. Tabelle 6).

Aktionen	Hexadezimalwerte
AK_REST	0x80
AK_MOVE	0x81
AK_LOAD	0x82
AK_UNLOAD	0x83
AK_SAY	0x84
AK_TELL	0x85
AK_EXTINGUISH	0x86
AK_RESCUE	0x87
AK_CLEAR	0x89

Tabelle 5: Zuordnungstabelle für Agentenaktionen

Eigenschaften	Integerwerte
PROPERTY_OBJECT_TYPE	12
PROPERTY_POSITION	6
PROPERTY_X	3
PROPERTY_Y	4
PROPERTY_HP	10
PROPERTY_BURIEDNESS	23
PROPERTY_WATER_QUANTITY	25
PROPERTY_FIERYNESS	16
PROPERTY_BUILDING_AREA_GROUND	51
PROPERTY_BROKENNESS	17

Tabelle 6: Ausschnitt der Zuordnungstabelle für Objekteigenschaften

Weitere Zuordnungen der Datei *rescue.log* betreffen die Objekttypen, die wie die Objekteigenschaften auf Integerwerte abgebildet werden. Eine Darstellung dieser Zuweisungen zeigt Tabelle 7.

Objekttypen	**Integerwerte**
TYPE_ROAD	168
TYPE_BUILDING	176
TYPE_REFUGE	184
TYPE_FIRE_STATION	185
TYPE_AMBULANCE_CENTER	186
TYPE_POLICE_OFFICE	187
TYPE_NODE	200
TYPE_WORLD	208
TYPE_CIVILIAN	232
TYPE_FIRE_BRIGADE	233
TYPE_AMBULANCE_TEAM	234
TYPE_POLICE_FORCE	235

Tabelle 7: Zuordnungstabelle für Objekttypen

Nachdem der Prozess zur Datenrekonstruktion aus den kodierten Logdateien abgeschlossen ist, erfolgt eine Aufbereitung der ausgelesenen Informationen. Nicht alle Daten, die in den Logdateien gespeichert sind, werden zur Analyse und Planerkennung der Rescue-Simulation benötigt. Daher werden die nicht relevanten Informationen von der Generator-Komponente ausgesondert. Eine Beschreibung dieses Prozesses erfolgt im folgenden Kapitel 5.3.

5.3 Funktionsweise des Generators

Der Generator filtert jene Informationen, welche zur Analyse eines Agententeams sowie für die Planerkennung relevant sind und strukturiert diese in einem übersichtlichen Format, so dass eine informative Ausgabe über die gespeicherten Informationen dargeboten werden kann. Die Filterung der Informationen auf Basis der Dateien *rescue.log* und *action.log* erfolgt in zwei Schritten:
Im ersten Schritt werden die Aktionen aus der Datei *action.log* gefiltert.
Im zweiten Schritt werden die zugehörigen relevanten Objektinformationen aus der Datei *rescue.log* extrahiert.

Die Datei *action.log* beinhaltet zu jeder Aktion die Identifikationsnummer (Id) des ausführenden Agenten sowie die Id-Werte aller betroffenen Objekte (vgl. Kapitel 5.1).
Im einfachsten Fall ist von der ausgeführten Aktion nur ein einzelnes Objekt betroffen, und die zu einer Aktion gespeicherten Daten bestehen neben der Aktion und der Sender-Id aus einem Id-Wert, der das betroffene Objekt eindeutig beschreibt. Dieser Fall gilt für die Aktionen *Rescue*, *Load*, *Unload*, *Clear* und *Extinguish*. Die aktionsspezifischen Zielobjekte sind in Tabelle 8 aufgeführt.

Aktionen	Zielobjekte	Interpretation
Rescue	Id eines Zivilisten	Ein verschütteter Ziv. soll „ausgegraben“ werden
Load	Id eines Zivilisten	Ein ausgegrabener Ziv. soll „aufgeladen“ werden
Unload	Id eines Zivilisten	Ein ausgegrabener Ziv. soll „abgeladen“ werden
Clear	Id einer Straße	Eine Straße soll von einer Blockade „bereinigt“ werden
Extinguish	Id eines Gebäudes	Ein brennendes Gebäude soll „gelöscht“ werden
Rest	-kein Ziel-	Ein Agent soll sich „erholen“ oder Wasser „auftanken“

Tabelle 8: Zielobjekte der Aktionen

Wie aus Tabelle 8 ersichtlich wird enthält die Aktion *Rest* innerhalb der Datei *action.log* kein Zielobjekt. Führt ein Agent die Aktion *Rest* aus, ist kein weiteres Objekt von dieser Aktion betroffen. Diese Aktion hat lediglich Auswirkungen auf die agentenspezifischen Eigenschaften: Falls sich der Agent bei der Ausführung der Aktion *Rest* in einem *Refuge* befindet, so wird sein Gesundheitszustand (*HealthPoints*) verbessert oder

im Falle eines Feuerwehragenten der Wassertank (*WaterQuantity*) nachgefüllt (vgl. Kapitel 3.3).

Im komplexeren Fall der Aktion *Move* sind mehrere Zielobjekte betroffen. Hier bestehen die gespeicherten Daten neben der Aktion und der Sender-Id aus Informationen über eine vom Agenten berechneten Route. Diese besteht aus allen Id-Werten der zu passierenden Objekte, wie Straßen und Kreuzungen. Nach erfolgreicher Ausführung der Aktion *Move* besteht die Position des Agenten in der Regel aus der letzten Objekt-Id der berechneten Route. Dieser Sachverhalt legt die Interpretation eines geplanten Ziels des Agenten nahe, welches aus dem letzten Objekt der in der Datei *action.log* hinterlegten Route besteht. In Ausnahmefällen wird der letzte Punkt der Route nicht erreicht, wenn beispielsweise die berechnete Route von unpassierbaren Straßen unterbrochen ist. Allerdings lässt sich in solchen Situationen - auch mit dem Wissen über die gesamte Route - keine Aussage darüber machen, wo sich der Agent im nächsten Zeitpunkt der Simulation befinden wird. Das vom Agenten erreichte Objekt lässt sich erst im nächsten Zeitschritt anhand der aktuellen Position des Agenten bestimmen. Aus der berechneten Route einer Aktion *Move* wird daher nur die letzte Objekt-Id in die Ausgabe des Generators übernommen. Somit enthält auch die durch den Generator aufbereitete Aktion *Move* nur noch einen Id-Wert eines Zielobjektes. Dieser Wert kann unterschiedliche Objekttypen beschreiben (vgl. Tabelle 9).

Aktion	**Zielobjekt-Id**	**Interpretation**
Move	Id einer Straße	Ziel ist eine Straße
	Id einer Kreuzung	Ziel ist eine Kreuzung
	Id eines Gebäudes	Ziel ist ein Gebäude
	Id eines Zivilisten	Ziel ist die Position eines Zivilisten

Tabelle 9: Zielobjekte der *Move*-Aktion

Der zweite Arbeitsschritt betrifft die Filterung und Zuweisung der in Kapitel 3.2.3 beschriebenen Eigenschaften der Objekte. Diese Objektinformationen sind in der Datei *rescue.log* kodiert. Die dynamischen Eigenschaften der Objekte sind interessant und nützlich bei der Analyse einer Rescue-Simulation, da diese durch Aktionen der Agenten verändert werden können. Daher werden diese Daten zur weiteren Verwendung aufbereitet. Die statischen Eigenschaften, wie beispielsweise die Grundfläche (*Buildin-*

gAreaGround) oder Farbe (*BuildingColour*) eines Gebäudes (vgl. Abbildung 9), sind sowohl für die Analyse eines Rescue-Agententeams als auch für die Planerkennung unerheblich und werden daher vom Generator nicht weiterverarbeitet.
Die aus der Datei *rescue.log* ermittelten Eigenschaften sind abhängig von den gefilterten Aktionsdaten aus Schritt 1. Zu der im ersten Schritt ermittelten Agenten- und Zielobjekt-Id werden im zweiten Schritt detaillierte Informationen zugeordnet. Diese aus der Datei *rescue.log* ausgelesenen Informationen sind in Tabelle 10 dargestellt.

Informationen der Datei *action.log*	**Informationen der Datei *rescue.log***
• Zeitpunkt der Simulation • Aktion	
• Id-Wert des ausführenden Agenten	• Agententyp • Agentenposition • Gesundheitszustand des Agenten (*HealthPoints*) • Wasservorrat des Agenten (*WaterQuantity*) (falls Agententyp = Feuerwehr)
• Id-Wert des Zielobjektes	• Objekttyp • Objektposition • Ausmaß des Feuers des Geb. (*Fieryness*) (falls Objekttyp = Gebäude) • Gesundheitszustand des Ziv. (*HealthPoints*) • Grad der Verschüttung des Ziv. (*Buriedness*) (falls Objekttyp = Zivilist) • Grad der Blockiertheit der Str. (*Block*) (falls Objekttyp = Straße)

Tabelle 10: ermittelte Daten der Dateien *rescue.log* und *action.log*

Die extrahierten Daten der Datei *rescue.log* bestehen bezüglich des Agenten aus dem Agententyp, der Agentenposition, dem Gesundheitszustand des Agenten (*HealthPoints*), sowie im Falle eines Feuerwehragenten aus dem Wasservorrat des Agenten (*WaterQuantity*). In Bezug auf das Zielobjekt einer Aktion werden der Objekttyp, die Objektposition, und bei einem Gebäudeobjekt beispielsweise das Ausmaß des Feuers (*Fieryness*), aus der Datei *rescue.log* bestimmt.

Nachdem die Daten der Logdateien geparst und gefiltert wurden, erfolgt eine Gliederung der relevanten Informationen in einem übersichtlichen Format. Das Ergebnis der Datenaufbereitung ist in Kapitel 5.4 dargestellt.

5.4 Ausgabe der Daten

Als Ergebnis liefert der Generator die aufbereiteten Informationen aus den Logdateien in unterschiedlichen Ausgabeformaten. Zum einen werden die aufbereiteten Informationen der Logdateien textbasiert in einer HTML-Datei abgelegt, die mit jedem herkömmlichen Web-Browser eingesehen werden kann. Zum anderen erfolgt eine Darstellung der Daten mittels einer graphischen Ausgabe, die auf dem von T. Morimoto entwickeltem Monitor [24] basiert (vgl. Kapitel 3.2.1).

5.4.1 Textbasierte Ausgabe

Aufbauend auf den in Kapitel 5.3 generierten Daten der Logdateien *rescue.log* und *action.log* wird eine Textdatei erzeugt, die alle relevanten Informationen über die Agenten, deren Aktionen und die davon betroffenen Objekte in aufbereiteter Form beinhaltet. Die Gliederung dieser Ausgabeform erfolgt nach Zeitschritten der Simulation. Für jeden Zeitschritt wird eine Auflistung aller agierenden Agenten mit den entsprechenden Daten dargeboten.
Die textbasierten Daten sind in der Datei *Ausgabe.html* gespeichert. Durch die HTML Formatierung können die Daten mittels eines Browsers in einer tabellarischen und somit übersichtlichen Form angesehen werden. Tabelle 11 zeigt einen Ausschnitt der aufbereiteten Informationen im HTML Format.

Zu jedem Zeitpunkt (Time) werden alle Agenten (Agent) aufgelistet, die zu diesem Zeitpunkt eine Aktion ausgeführt haben. Die Agenten werden dargestellt durch Angabe ihres Agententyps sowie der eindeutigen Identifikationsnummer. Zu jedem Agenten werden die Position (Position) sowie der Grad der Gesundheit (HP=HealthPoints) angegeben. Die Darstellung der Agentenposition erfolgt durch den Typ und die Id des Objektes, auf dem oder in dem sich der Agent zu diesem Zeitschritt befindet. Der Gesundheitszustand der Agenten wird repräsentiert durch einen Zahlenwert, der zwischen *max_hp* - in den

bisherigen Rescue-Simulationswettkämpfen betrug der Wert von *max_hp* 10000 - und 0 variieren kann.

Time	Agent	Position	HP	WQ	ActionType	ActionTarget	ActionData
…	…	…	…	…	…	…	…
166	AmbulanceTeam (Id:92036883)	Road (Id:264896022)	10000	null	MOVE	Civilian (Id:263546484)	Civ.Buriedness: 0, Civ.HP: 3526
166	AmbulanceTeam (Id:161412197)	Building (Id:263546484)	9985	null	LOAD	Civilian (Id:200142917)	Civ.HP: 3526
169	FireBrigade Id:259863813)	Node (Id:175841598)	10000	10000	EXTINGU-ISH	Building (Id:267882706)	Bldg.Fieryness: 3
169	AmbulanceTeam (Id:198082971)	Building (Id:117266151)	9422	null	RESCUE	Civilian (Id:134187089)	Civ.Buriedness: 19, Civ.HP: 5541
169	PoliceForce (Id:268349708)	Road (Id:201145751)	10000	null	CLEAR	Road (Id:201145751)	Rd.Block: 2000
…	…	…	…	…	…	…	…
170	AmbulanceTeam (Id:161412197)	Road (Id:268338687)	9985	null	MOVE	Refuge (Id:209458916)	null
170	PoliceForce (Id:251375364)	Road (Id:215028307)	10000	null	MOVE	BlockedRoad (Id:117031798)	Rd.Block: 2000
170	FireBrigade (Id:134139611)	Node (Id:175841598)	10000	5000	EXTINGU-ISH	Building (Id:267882706)	Bldg.Fieryness: 3
170	AmbulanceTeam (Id:198082971)	Building (Id:117266151)	9415	null	RESCUE	Civilian (Id:134187089)	Civ.Buriedness: 18, Civ.HP: 5496
170	FireBrigade (Id:133017882)	Refuge (Id:209458916)	10000	8000	REST	null	null
…	…	…	…	…	…	…	…

Tabelle 11: Ausschnitt der Datei *Ausgabe.html*

Handelt es sich bei einem Agenten um einen Feuerwehragenten (FireBrigade), so wird zusätzlich zu den Angaben über Position und Gesundheit auch die Wassermenge (WQ=WaterQantity) angegeben, die sich im Tank der Agenten befindet. Dieser Wert wird nur bei Feuerwehragenten gesetzt und beträgt bei allen anderen Agenten „null“, da diese keinen Wassertank besitzen (vgl. Kapitel 3.2.3).

Die Aktionen der Agenten werden in der Spalte ActionType, unter Angabe der Aktionstypen, aufgelistet. Die Werte aus der Spalte ActionTarget beschreiben das Zielobjekt, auf welches die Aktion gerichtet ist, und dessen Eigenschaften gegebenenfalls durch die Aktion beeinflusst wird. Je nach Aktionstyp können unterschiedliche Zielobjekte auftreten, die durch den Objekttyp und die eindeutige Id aufgeführt sind (vgl. Tabelle 8 und 9).
Besitzt ein Zielobjekt Eigenschaften, die von den Agenten durch Aktionen beeinflusst werden können, werden diese Eigenschaften unter ActionData aufgeführt. Die Eigenschaften der unterschiedlichen Objekte sind in Tabelle 10 (Kapitel 5.3) dargestellt.

Alle hier beschriebenen Informationen liegen für jeden Zeitschritt, das heißt für jede der insgesamt 300 Runden, die in einer Simulation durchlaufen werden, vor. Auf dieser Basis können alle getätigten Aktionen der einzelnen Agenten in Reaktion auf die gegebenen Weltumstände systematisch verfolgt werden.

5.4.2 Graphische Ausgabe

Der im Rescue-Simulator Version 0.48 verwendete Monitor dient zur Darstellung eines Rescue-Simulationsdurchlaufs im *Online*- sowie im *Offline*-Modus (vgl. Kapitel 3.2.1).
Im *Offline*-Modus erfolgt die Visualisierung lediglich durch Darstellung der Positionsänderungen, ohne Berücksichtigung der von den Agenten ausgeführten Aktionen. Somit ist der im Simulator verwendete Monitor für Fehlersuche bzw. -behebung oder Analysezwecke eines Agententeams nicht hinreichend von Nutzen. Wichtiger ist es die Handlungen in Reaktion auf die entsprechenden Weltumstände zu kennen, um mögliches Fehlverhalten oder unterschiedliche Verhaltensweisen der Agenten zu analysieren.
Zu diesem Zweck wurde der Monitor von T. Morimoto [24] für die Verwendung im Rescueanalyser weiterentwickelt. Der resultierende Monitor ist in Lage sowohl alle Positionsänderungen, als auch alle möglichen Aktionen der Agenten während eines Simulationsdurchlaufs darzustellen.

Die visuelle Wiedergabe des im Rescueanalyser eingebundenen Monitors bietet folglich dieselbe Illustration wie auch ein *Online*-Durchlauf einer Rescue-Simulation im Monitor des Rescue-Simulators (vgl. Kapitel 3.2.1). Abbildung 11 zeigt die Visualisierung des entwickelten Monitors. Im Gegensatz zur *Offline*-Darstellung des herkömmlichen Monitors sind nun die Aktionen der einzelnen Agenten sichtbar. Man kann zum einen

Ambulanzagenten bei der Bergung von Zivilisten erkennen (1), zum anderen sind Polizisten bei der Räumung von Straßen sichtbar (2). Außerdem sind die Löschaktionen der Feuerwehragenten visualisiert (3).

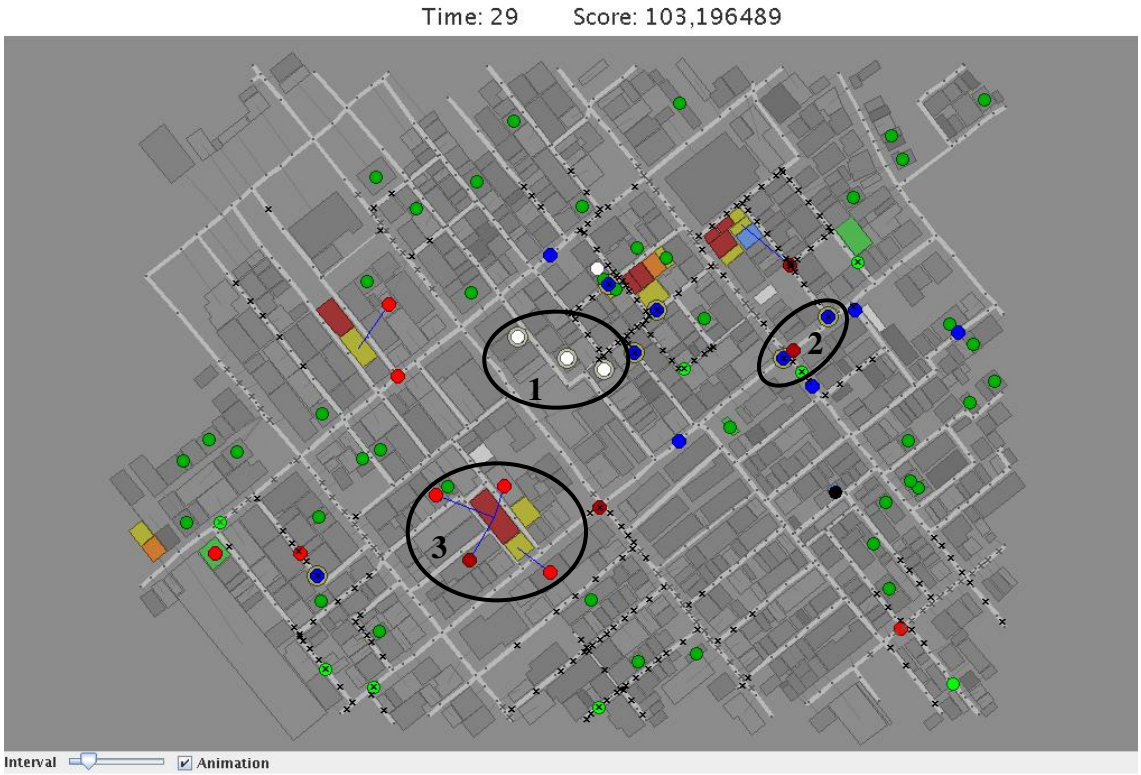


Abbildung 11: Visualisierung des Rescueanalyser-Monitors

Der Monitor ist innerhalb der Analyser-Komponente des Rescueanalysers verwirklicht. Dieser kann auf jeder Linux Plattform - auch ohne existierenden Rescue-Simulator, das heißt ohne Vorhandensein eines Kernels oder des GIS - verwendet werden, da alle benötigten Informationen aus der Simulatorumgebung, innerhalb des Monitors eingebunden sind. Einzige Voraussetzung ist, dass die Dateien *rescue.log* und *action.log* aus einem Simulationsdurchlauf des Rescue-Simulators der Version 0.48 vorliegen.

Die zur Verfügung stehenden Ausgabemöglichkeiten der Analyser-Komponente des Rescueanalysers liefern alle relevanten Informationen eines Agententeams, die zu Debugging- und Analysezwecken genutzt werden können. Die grafische Ausgabe im Monitor ermöglicht das Abspielen der kompletten Simulation mit der Darstellung aller Aktionen der Agenten. Zur Analyse sowie zur Fehlerbehebung und -beseitigung eines Agententeams ist die textbasierte Repräsentation der Daten von Vorteil. Hier können die

aufgelisteten Daten der kompletten Simulation methodisch verfolgt werden, während bei der graphischen Ausgabe ein stetiger Prozess über die Zeit voranschreitet. Dieser ist wiederholbar und in der Geschwindigkeit veränderbar, aber alle Agenten führen ihre Aktionen zeitgleich und fortlaufend aus, was die Nachverfolgbarkeit erschwert.
Während die graphische Ausgabe zur Beobachtung der Teamarbeit, sowie zur Verfolgung von ausgeführten Strategien gut geeignet ist, stellt die textbasierte Ausgabe ein besseres Werkzeug für die Betrachtung einzelner Aktionen und deren Auswirkungen dar.

5.5 Analyse der Kommunikationsbefehle

Im ersten Teil dieser Arbeit wurde in Kapitel 3.4 die mögliche Kommunikation innerhalb der Rescue-Simulation erläutert. Die zur Verfügung stehenden Aktionen sind *say* und *tell*.
Innerhalb der textbasierten Ausgabe, die in Kapitel 5.4.1 beschrieben wurde, traten diese Aktionen allerdings nicht auf. Die Ursache für die fehlenden Kommunikationsbefehle liegt im Kernel des Simulators begründet. Wie alle anderen Aktionsbefehle (vgl. Tabelle 2) werden auch *say* und *tell* von den Agenten an den Kernel gesendet (vgl. Kapitel 3.2.2) und von diesem weiter verarbeitet. Allerdings werden die Befehle *say* und *tell* vom Simulator nicht in die Logdateien mit aufgenommen. Somit ist es also nicht möglich, aus den vom Simulator der Version 0.48 erzeugten Logdateien, die von den Agenten gesendeten Kommunikationsbefehle der Analyse zuzuführen und in der textbasierten Ausgabe darzustellen.

Die Möglichkeit kann durch Modifikation des Quellcodes der Kernel-Komponente des Rescue-Simulators der Version 0.48 geschaffen werden, wodurch die Kommunikationsbefehle in die Logdateien mit aufgenommen werden können. Die benötigten Änderungen am Quellcode der Kernel-Komponente werden in Anhang A ausführlich beschrieben, so dass sie bei Bedarf selbst durchgeführt werden können. Werden die mittels des modifizierten Simulators erzeugten Logdateien an den Rescueanalyser übergeben, enthält die generierte Datei *Ausgabe.html* alle gesendeten Kommunikationsbefehle *say* und *tell* der Agenten. In der HTML-Ausgabe werden die Kommunikationsbefehle als Aktionen der Agenten angezeigt.
Wie in Tabelle 12 zu erkennen ist, wird bei einer Aktion vom Typ *say* oder *tell* nur der Aktionstyp (ActionType) dargestellt. Die Aktionen *say* und *tell* enthalten zwar weitere

Daten, die mit ihnen versendet werden (ActionData), nämlich die eigentliche Nachricht. Allerdings fordert die Vorgabe für solche Nachrichten lediglich eine Längenbeschränkung auf maximal 256 Byte (vgl. Kapitel 3.4). Für den Inhalt einer Nachricht gibt es keinerlei Vorgaben oder Beschränkungen. Um möglichst viele Informationen in eine Nachricht zu packen bedienen sich die meisten Teams der Zuhilfenahme einer Datenkompression, die es dem Simulator unmöglich macht, die gesendeten Nachrichten einzusehen.

Time	**Agent**	**Position**	**HP**	**WQ**	**Action Type**	**ActionTarget**	**ActionData**
…	…	…	…	…	…	…	…
166	FireBrigade (Id:259104353)	Road (Id:266070153)	10000	15000	TELL	null	null
166	Civilian (Id:842675145)	Building (Id:268894114)	6879	null	SAY	null	null
…	…	…	…	…	…	…	…

Tabelle 12: Darstellung der Kommunikationsbefehle *say* und *tell*

Neben der Information über die mittels *say* und *tell* gesendeten Nachrichten ist für die Kommunikation auch die Annahme der Nachrichten von Bedeutung. In einer Runde der Rescue-Simulation kann nur eine bestimmte Anzahl von Nachrichten gesendet und empfangen werden (vgl. Kapitel 3.4). Für die Betrachtung der tatsächlich stattfindenden Kommunikation ist es von Bedeutung, welche dieser Nachrichten auch wirklich von den Agenten angenommen werden. Denn nur die angenommenen Nachrichten sind Bestandteil eines Kommunikationsprozesses der Agenten untereinander. Alle nicht angenommenen Nachrichten werden verworfen und spielen keine Rolle für die Kommunikation.

Die Analyse der Kernel-Komponente, welche als Kommunikationsschnittstelle zwischen Simulator und Agententeam fungiert (vgl. Kapitel 3.2.1) zeigt, dass keine der ausgetauschten Parameter die Annahme einer Nachricht dokumentieren. Das bedeutet, dass auf Simulatorebene nicht nachvollziehbar ist, welche Nachrichten von den Agenten angenommen und welche verworfen werden. Daher ist es dem Simulator nicht möglich die zulässige Anzahl der angenommenen Nachrichten zu überprüfen.
Diese aufgedeckte Schwachstelle des Simulators verhindert die Vervollständigung der Logdateien durch Empfangsbestätigungen. Dieser Sachverhalt stellt derzeit eine erheb-

liche Einschränkung in der Analyse sowie der Planerkennung dar, und verhindert somit die Verfolgung des Kommunikationsprozesses eines Agententeams.

6 Umsetzung der Methoden zur Planerkennung

In den vorangegangenen Kapiteln wurde das Analysewerkzeug für die Rescue-Simulation mit Aufbau und Arbeitsweise, sowie das Umfeld des Rescue-Simulators vorgestellt. Nun folgt die detaillierte Beschreibung des Planerkennungsprozess innerhalb der Recognizer-Komponente. Zunächst wird die notwendige Eingabe zum Planerkennungsalgorithmus beschrieben. Nachfolgend werden die weiteren erforderlichen Repräsentationen - Planbibliothek, Feature Decision Tree und Zeitstempel - erörtert. Anschließend werden die einzelnen Prozessschritte zur Planerkennung in Form eines Algorithmus spezifiziert. Abschließend wird die Ausgabe des Planerkennungsprozesses - die erkannten Plänen der Rescue-Agenten - beschrieben.

6.1 Eingabe zur Planerkennung

Als Eingabe dienen der Recognizer-Komponente die durch den Generator gefilterten Daten, deren Inhalt in Tabelle 11 exemplarisch dargestellt wurde.

Die Repräsentation der Daten erfolgt durch die Beschreibung von Weltzuständen (*States*).
Es seien

M_1 = Menge aller Zeitschritte der Rescue-Simulation (*Time*)
M_2 = Menge aller Rescue-Agenten (*Agent*),
M_3 = Menge aller möglichen Positionen eines Rescue-Agenten (*Position*),
M_4 = Menge aller möglichen Gesundheitswerte eines Rescue-Agenten (*HealthPoints*),
M_5 = Menge aller möglichen Werte für den Wasservorrat eines Feuerwehragenten (*WaterQuantity*),
M_6 = Menge aller möglichen Aktionen der Rescue-Agenten (*ActionType*),
M_7 = Menge aller möglichen Aktionsziele einer Aktion (*ActionTarget*),
M_8 = Menge aller möglichen Aktionsdaten zu einer Aktion (*ActionData*).

Dann ist

$$States = M_1 \times M_2 \times \ldots \times M_8$$

die Menge aller möglichen Weltzustände.

Innerhalb der Rescue-Simulation können allerdings nicht alle möglichen Weltzustände erreicht werden. Beispielsweise beschreibt

> (166, AmbulanceTeam(ID:92036883), Road(ID:264896022), 10000, null,
> EXTINGUISH, Civilian(ID:263546484), Bldg.Fieryness=5)

einen unmöglichen Weltzustand, der innerhalb der Rescue-Simulation nicht auftreten kann, da durch die Regeln des Rescue-Simulators festgelegt ist, dass ein Agent vom Typ *AmbulanceTeam* nicht in der Lage ist die Aktion *Extinguish* auszuführen.
Die Menge aller in der Rescue-Simulation möglichen Weltzustände wird bezeichnet als *RescueStates*, welche eine Teilmenge von *States* bildet:

$$RescueStates \subseteq States = M_1 \times M_2 \times \ldots \times M_8$$

Die Mengen M_1 bis M_8 repräsentieren die *Attribute*, nach denen die Daten der Generatorausgabe sortiert sind. Die möglichen Werte der einzelnen Attribute wurden bereits in Kapitel 5.4.1 exemplarisch dargestellt. Tabelle 13 beschreibt die Wertebereiche der Attribute.

Attribute	**Wertebereich**
Time	{0,…, S*imulatingTime*}
Agent	{*FireBrigade, AmbulanceTeam, PoliceForce*} × Id
Position	{*Building, Road, Refuge, Node*} × Id
HealthPoints	{0, ..., *max_hp*}
WaterQuantity	{0, …, *tank_quantity_maximum*}
ActionType	{*Move, Rest, Rescue, Load, Unload, Extinguish, Clear*}
ActionTarget	{*Building, Road, Refuge, Node, Civilian*}× Id
ActionData	{(x,y) | x∈ {*Bldg.Fieryness, Rd.Block*} ∧ y ∈N_0 } ∪ {(x,y,z) | x∈ Civ.Info ∧ y, z ∈ N_0 } mit Civ.Info = { *Civ.Buriedness, Civ.HP* }

Tabelle 13: Wertebereiche der Attribute

Eine Beobachtung (*Obs*) ist eine Abbildung der Simulationszeit auf die möglichen Weltzustände der Rescue-Simulation (*RescueStates*), dabei beschreibt *dom*(x) den Wertebereich von x:

$$Obs: dom(Time) \rightarrow 2^{RescueStates}$$

Die Beobachtungen zu einem Zeitpunkt der Rescue-Simulation werden beschrieben durch:

$$Obs(t) = \{(t, m_2, \ldots., m_8) \in RescueStates)\}$$

Abbildung 12 stellt exemplarisch entsprechend den Daten aus Tabelle 11 die Menge der Beobachtungen zum Zeitpunkt t = 166 einer Rescue-Simulation dar.

***Obs*(166) = { (**166, AmbulanceTeam(ID:92036883),
Road(ID:264896022), 10000, null,

MOVE, Civilian(ID:263546484),
Civ.Buriedness: 0, Civ.HP: 3526**),**

(166, AmbulanceTeam(ID:161412197),
Builing(ID:263546484), 9985, null,

Abbildung 12: Menge der Beobachtungen zum Zeitpunkt t = 166

Die Sequenz der Beobachtungen eines (im Allgemeinen kompletten) Durchlaufs einer Rescue-Simulation, das heißt die Folge von Beobachtungen

$$Obs(t_i) \text{ mit } 1 \leq i \leq SimulatingTime,\ 1 \leq t_i \leq SimulatingTime \text{ und } t_{i+1} = t_i+1$$

dient als Eingabe für den Planerkennungsprozess.

Neben den Beobachtungen verwendet der Planerkennungsalgorithmus weitere Datenstrukturen als Eingabe. Diese bestehen aus einer Planbibliothek und einem Entscheidungsbaum. Während des Planerkennungsprozesses werden die Beobachtungen unter Zuhilfenahme des Entscheidungsbaumes mit der Planbibliothek verglichen, um Pläne in den Beobachtungen zu erkennen.

6.2 Planbibliothek

Zu Beginn dieses Kapitels wird die Spezifikation der Planbibliothek beschrieben. Anschließend wird exemplarisch der Ablauf des Planerkennungsprozess auf Basis der Planbibliothek skizziert.

6.2.1 Spezifikation der Planbibliothek

Die Planbibliothek umfasst alle möglichen Pläne die Rescue-Agenten ausführen können. Die einzelnen Pläne basieren auf den Aktionen der verschiedenen Rescue-Agenten (vgl. Kapitel 3.3) und der sinnvollen Ausführungsreihenfolge dieser Aktionen. Die Planbibliothek ist statisch und unabhängig von den Beobachtungen der verschiedenen Agententeams.
Die Planbibliothek wurde auf Grundlage der RoboCup-Rescue Manuals [15], [20] und verfügbaren Teambeschreibungen [17], [19], [25] entwickelt. Die Manuals enthalten alle in Kapitel 3.2 und 3.3 beschrieben Aktionen und Eigenschaften der Agenten die vom Rescue-Simulator akzeptiert werden, das heißt zur Grundmenge der gültigen Befehle der Rescue-Simulation gehören. Die Teambeschreibungen müssen von allen Agententeamentwicklern, die am RoboCup-Wettbewerb teilnehmen möchten, verfasst werden. Diese enthalten unter anderem Erläuterungen der programmierten Verhaltensweisen der Agenten. Die Verwendung von Teambeschreibungen und Manuals als Basis zur Festlegung der Planbibliothek gewährleistet, dass die zu erkennenden Pläne auch tatsächlich als Verhaltensweisen der Agenten in der Rescue-Simulation ausgeführt werden und somit idealerweise eindeutig erkannt werden.

Die Darstellung der Planbibliothek erfolgt in Anlehnung an [3]. Die Repräsentation der Pläne besteht aus einem *gerichteten Graphen* G = (V, E) mit einer endlichen Menge von *Knoten* (V), und einer endlichen Menge von *Kanten* (E).
Die Kanten des Graphen können von zwei unterschiedlichen Typen sein:

- *zerlegende Kanten* - diese unterteilen die Pläne solange in Planschritte, bis ein Blatt erreicht wird - und
- *sequentielle Kanten,* welche die Ausführungsreihenfolge festlegen.

Die Kantenmenge E teilt sich daher in die Menge der zerlegenden Kanten (E_z) und die Menge der sequentiellen Kanten (E_s), dass heißt $E = E_z \cup E_s$ mit $E_z \cap E_s = \varnothing$.

Ein spezieller Knoten *root*(G)∈ V, welcher nur ausgehende Kanten besitzt, stellt den *Wurzelknoten* des Graphen dar. Alle Knoten, die aus dem Wurzelknoten durch zerlegende Kanten hervorgehen, also alle v∈ V mit (*root*(G), v) ∈ E_z stellen *Pläne* dar. Die Menge dieser Knoten repräsentiert die Menge aller Pläne der Planbibliothek:
Plans(G) = {v∈ V | (root(G),v) ∈ E_z}.

Die *Blätter* des Graphen entsprechen den Aktionen der Agenten. Alle inneren Knoten, die keine Pläne sind, werden als *Planschritte* bezeichnet.

Sind v_i, v_{i+1} ∈ V durch eine Kante verbunden, so repräsentiert W = (v_i, v_{i+1}) einen *Weg* in G. Existiert in G ein Weg W = (v_1, …, v_n) mit v_i ∈ V und gilt für alle Paare (v_i, v_{i+1}) ∈ E_s [bzw. (v_i, v_{i+1}) ∈ E_z] mit $1 \leq i < n$, so beschreibt W einen *sequentiellen [bzw. zerlegenden] Weg* in G.
Gilt für einen sequentiellen [bzw. zerlegenden] Weg W = (v_1, …, v_n) in G, dass W *maximal* ist, dass heißt es existiert in G kein W' = (v, v_1, …, v_n) bzw. W'' = (v_1, …, v_n, v), so stellt W einen *sequentiellen [bzw. zerlegenden] Pfad* dar.
Falls in W = (v_1, …, v_n) Start- und Endknoten identisch sind, dass heißt, falls $v_1 = v_n$ gilt, so beschreibt W einen *Zyklus*. Während in G Zyklen der Art W = (v, v) mit v ∈ E_s möglich sind, existieren keine Zyklen bei zerlegenden Kanten.

Ein Ausschnitt aus der verwendeten Planbibliothek ist in Abbildung 13 dargestellt. Die vollständige Planbibliothek, welche im Rahmen dieser Diplomarbeit spezifiziert wurde, befindet sich in Anhang B.

In Abbildung 13 sind exemplarisch die Pläne *HealMe* und *RescueBuriedCivilian* dargestellt. Die Kästen repräsentieren die Knoten der Planbibliothek, während die Pfeile die Kanten darstellen. Gestrichelte Pfeile entsprechen sequentiellen Kanten, und durchgehende Pfeile repräsentieren zerlegende Kanten. Ein Plan wird solange durch zerlegende Kanten in Planschritte aufgeteilt, bis an den Blättern nur noch Aktionen vorhanden sind.
Der Plan *HealMe* besteht aus dem sequentiellen Pfad, welcher sich aus dem Planschritt *MoveToRefuge* gefolgt von dem Planschritt *Recover* zusammensetzt. Der Planschritt *MoveToRefuge* untergliedert sich in eine Folge von mindestens einer Aktion *Move*. Der Planschritt *Recover* zerlegt sich in eine Folge von mindestens einer Aktion *Rest*.
Der Plan *RescueBuriedCivilian* besteht aus den Planschritten *MoveToCivilian*, *RescueCivilian* und *TransportCivilian*, welche ebenso einen sequentiellen Pfad darstellen. Der Planschritt *MoveToCivilian* besteht aus einer Folge von mindestens einem Move. Der

Planschritt *RescueCivilian* besteht aus einer Folge von mindestens einem *Rescue*. Der Planschritt *TransportCivilian* setzt sich aus dem sequentiellen Pfad der Planschritte *LoadCivilian*, *MoveToRefuge* und *UnloadCivilian* zusammen. Der Planschritt *LoadCivilian* zerlegt sich in ein Blatt mit der Aktion *Load*. Der Planschritt *MoveToRefuge* untergliedert sich in eine Folge von mindestens einer Aktion *Move*. Der Planschritt *UnloadCivilian* zerlegt sich in ein Blatt mit der Aktion *Unload*.

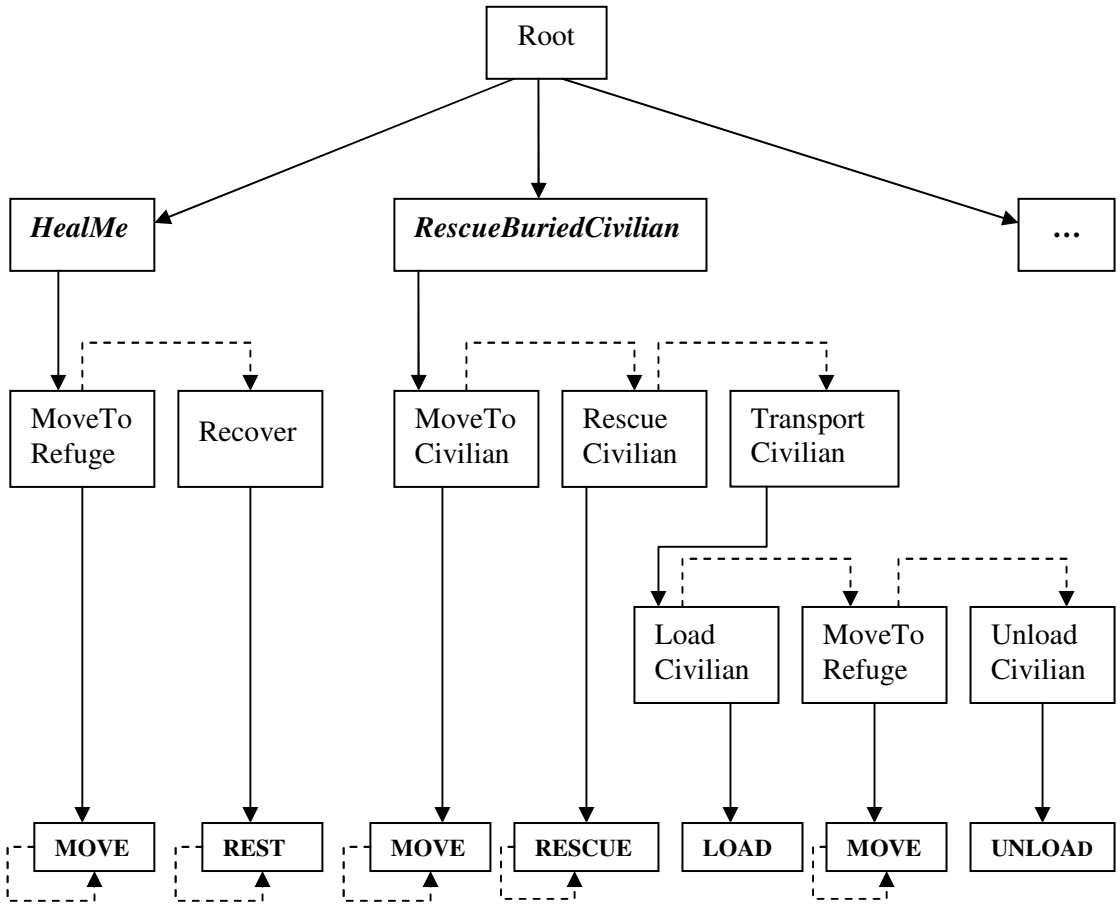


Abbildung 13: Ausschnitt der verwendeten Planbibliothek

6.2.2 Ansatz zur Planerkennung mittels der Planbibliothek

Der Ansatz zum Planerkennungsprozess startet in der Planbibliothek bei den Blättern. Die Blätter repräsentieren jene Aktionen, welche auch in den Beobachtungen im Attribut *ActionType* wieder zu finden sind (vgl. Abbildung 12). Durch Vergleichen des Attributes *ActionType* aus einer Beobachtung mit den Blättern der Planbibliothek werden

nun jene Planschritte näher betrachtet, welche die positiv verglichene Aktion enthalten. Beispielsweise wird die Beobachtung

b={(166, AmbulanceTeam(Id:92036883), Road(Id:264896022), 10000, null,
MOVE, Civilian (Id:263546484), Civ.Buriedness: 0 Civ.HP: 3526)}

den Blättern mit der Aktionsbezeichnung Move aus Abbildung 13 zugeordnet, welche im Rahmen der Planschritte *MoveToRefuge*(1), *MoveToCivilian*(2) oder *MoveToRefuge*(3) ausgeführt werden (vgl. Abbildung 14).

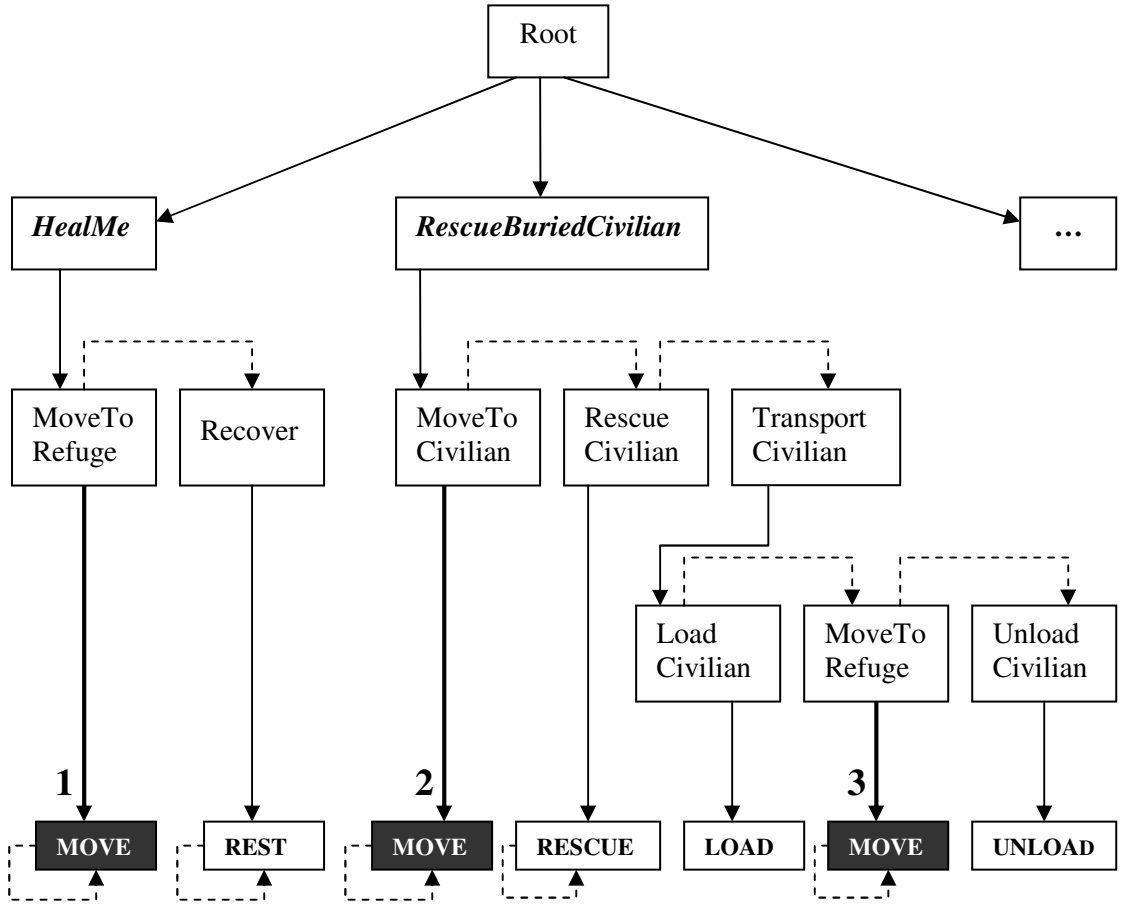


Abbildung 14: Vergleichen der Aktion einer Beobachtung mit Blättern der Planbibliothek

Durch die Bestimmung eines Blattes der Planbibliothek kann folglich keine eindeutige Zuweisung der Beobachtung zu einem Planschritt erfolgen, da gleiche Aktionen - wie im vorstehenden Beispiel beschrieben - in unterschiedlichen Planschritten vorkommen können.

Die eindeutige Bestimmung des ausgeführten Planschrittes erfolgt anhand weiterer Attribute der Beobachtungen. Wird im Fall der Beobachtung

b={(166, AmbulanceTeam(Id:92036883), Road(Id:264896022), 10000, null,
MOVE, Civilian (Id:263546484), Civ.Buriedness: 0 Civ.HP: 3526)}

neben dem Attribut *ActionType* mit dem Wert *Move* der Wert des Attributs *ActionTarget* betrachtet, kann eindeutig der durchgeführte Planschritt bestimmt werden. In der Beobachtung b entspricht der Attributwert von *ActionTarget* dem Objekt *Civilian*. Damit kann die Beobachtung b eindeutig dem Planschritt *MoveToCivilian* zugeordnet werden.
Leider ist das Attribut *ActionTarget* nicht für alle Aktionen als Entscheidungskriterium zur Identifikation von Planschritten geeignet. Betrachtet man beispielsweise die Aktion *Rest*, die ebenfalls mehrfach in unterschiedlichen Planschritten bzw. Plänen der Planbibliothek wieder zu finden ist (vgl. Anhang B), so ist der Wert von *ActionTarget* immer gleich „null“ (vgl. Tabelle 8), und es kann keine zusätzliche Information aus diesem Attribut gewonnen werden. In diesem Fall muss ein anderes Attribut zur Erkennung des Planschritts ausgewählt werden.

Dieses Beispiel verdeutlicht, dass für verschiedene Aktionen unterschiedliche Attribute für die Zuweisung einer Beobachtung zu einem Planschritt ausschlaggebend sind. Um nicht für jede Beobachtung alle Attribute testen zu müssen, wird zur Bestimmung der entscheidenden Attribute eine weitere Datenstruktur entworfen, welche die zu testenden Attribute in Abhängigkeit der beobachteten Aktionen vorgibt. Diese Datenstruktur entspricht einem Entscheidungsbaum und wird als *Feature Decision Tree* bezeichnet.

6.3 Feature Decision Tree

Zur Bestimmung der zu testenden Attribute bezüglich der beobachteten Aktion wird der *Feature Decision Tree* (FDT) entworfen. Anhand dieser Datenstruktur kann mit wenig Aufwand effektiv eine Aussage über den zu einer Beobachtung passenden Planschritt getätigt werden [3]. Die Daten des Feature Decision Tree sind statisch und werden ebenso wie die Inhalte der Planbibliothek nur einmal generiert.

6.3.1 Spezifikation des FDT

Die Repräsentation des FDT erfolgt in Anlehnung an [3]. Die zugrunde liegende Datenstruktur entspricht einem *Entscheidungsbaum.* Die Struktur des Baumes besteht aus einem *zusammenhängenden azyklischen gerichteten Graphen* G = (V, E) mit einer endlichen Menge von *Knoten* (V) und einer endlichen Menge von *Kanten* (E). Weiterhin existiert in G genau ein Knoten *w*, welcher die *Wurzel* des Baumes darstellt. Die Wurzel enthält nur ausgehende Kanten, es existieren keine Kanten, die in der Wurzel enden. Die Knoten, die lediglich eingehende Kanten besitzen, stellen die *Blätter* des Baumes dar. Im Feature Decision Tree repräsentieren die Knoten die zu testenden Attribute der Beobachtungen. Die Kanten eines Knotens enthalten jede mögliche Wertebelegung des getesteten Attributs (vgl. Tabelle 13). Die Blätter entsprechen Planschritten der Planbibliothek.

Die Konstruktion des FDT erfolgt, nach dem *Top-Down-Prinzip*. Zunächst wird die Wurzel des Baumes bestimmt. Das Attribut *ActionType* wird als Wurzel des FDT festgelegt, da anhand dieses Attributes bereits eine eindeutige Zuweisung der Beobachtung zu einem Blatt der Planbibliothek erfolgen kann (vgl. Kapitel 6.2). Vom Wurzelknoten gehen sieben Kanten aus. Jede dieser Kanten repräsentiert eine mögliche Aktion eines Agenten. Die Werte dieser Kanten lauten (vgl. Tabelle 13): *Rest*, *Move*, *Extinguish*, *Clear*, *Rescue*, *Load* und *Unload*.
Nach der Bestimmung des Wurzelknotens, erfolgt die Generierung des Baumes, indem ein weiteres Attribut gewählt wird, welches die Daten der Beobachtungen mit dem größten Informationsgewinn bezüglich der Zuordnung zu Planschritten aufteilt.

Dieses Vorgehen wird am Beispiel der Aktion *Move* verdeutlicht: Es existieren die fünf nachfolgend aufgelisteten, unterschiedliche Planschritte, welche die Aktion *Move* enthalten können:

- *MoveToRefuge* (MTR),
- *MoveToBurningBuilding* (MTBB),
- *MoveToBlockedRoad* (MTBR),
- *MoveToCivilian* (MTC) und
- *DistrictExploration* (DE).

Ziel ist es nun, ein weiteres Attribut aus den Beobachtungen zu wählen, welches die Identifikation des zugehörigen Planschrittes am effektivsten ermöglicht. Dazu werden alle Attribute einer Beobachtung bezüglich ihrer Aussagekraft über die Zuordnung zu einem Planschritt bewertet.

Tabelle 14 zeigt exemplarisch die Bewertung der Attribute *ActionTarget* und *Agent* im Bezug auf den Planschritt *MoveToRefuge* (MTR).

a)

ActionTarget	**MTR ?**
Refuge	Ja
Bldg≠Refuge	nein
Road	nein
Civilian	nein
Node	nein

b)

Agent	**MTR ?**
FireBrigade	?
PoliceForce	?
AmbulanceTeam	?

Tabelle 14: Aussagekraft von *ActionTarget* und *Agent* bezüglich *MoveToRefuge*

In Tabelle 14 a) und b) stellt jeweils die linke Spalte die Attribute *ActionTarget* und *Agent* mit ihren möglichen Wertebelegungen dar. Die rechte Spalte der Tabellen 14 a) und b) repräsentiert eine Aussage über die Ausführbarkeit des Planschritts *MoveToRefuge*(MTR) in Abhängigkeit der Attributwerte von *ActionTarget* bzw. *Agent*. Im Falle von *ActionTarget* (vgl. Tabelle 14 a)) kann zu jeder möglichen Wertebelegung eine eindeutige Aussage über die Ausführbarkeit von MTR getroffen werden. Entspricht der Wert von *ActionTarget* einem *Refuge*, so kann direkt der Planschritt *MoveToRefuge*

abgeleitet werden. Alle anderen Belegungen von *ActionTarget* widersprechen der Ausführung des Planschritts *MoveToRefuge*. Im Gegensatz dazu kann zu keiner Wertebelegung von *Agent* eine Aussage über die Ausführbarkeit von MTR getroffen werden (vgl. Tabelle 14b)).
Um eine endgültige Auswahl eines Kandidaten für den FDT zu treffen, muss allerdings auch die Ausführbarkeit der Attribute bezüglich aller anderen Planschritte getestet werden.

Tabelle 15 beinhaltet alle Aussagen, die bezüglich der Wertebelegungen von *ActionTarget* über die möglichen Planschritte getroffen werden können.

ActionTarget	**MTR**	**MTBB**	**MTBR**	**MTC**	**DE**
Refuge	ja	nein	nein	nein	?
Bldg≠Refuge	nein	?	nein	nein	?
Road	nein	nein	?	nein	?
Civilian	nein	nein	nein	ja	?
Node	nein	nein	nein	nein	nein

Tabelle 15: Aussagekraft von *ActionTarget* bezüglich aller möglichen Planschritte

Insgesamt existieren 25 Aussagen, von denen 19 eine eindeutige Aussage über die Ausführbarkeit des entsprechenden Planschritts darbieten. Das entspricht einer 76-prozentigen Informationsgewinnung durch das Attribut *ActionTarget*. Durch kein anderes Attribut kann dieser Wert erreicht oder übertroffen werden. Daher wird das Attribut *ActionTarget* für die Verwendung im FDT ausgewählt.
Das Verfahren der Attributauswahl wird solange wiederholt, bis an den Blättern des Baumes nur noch Planschritte der Planbibliothek auftauchen.

Abbildung 15 zeigt den Aufbau des verwendeten FDT für das Attribut *ActionType* mit dem Wert *Move*. Die Attribute werden durch Kästen dargestellt. Die Pfeile repräsentieren jede mögliche Wertebelegung eines Attributs. Die Blätter des FDT entsprechen den Planschritten der Planbibliothek. Der komplette FDT, welcher für die Implementation des Rescueanalysers verwendet wurde, befindet sich in Anhang C.

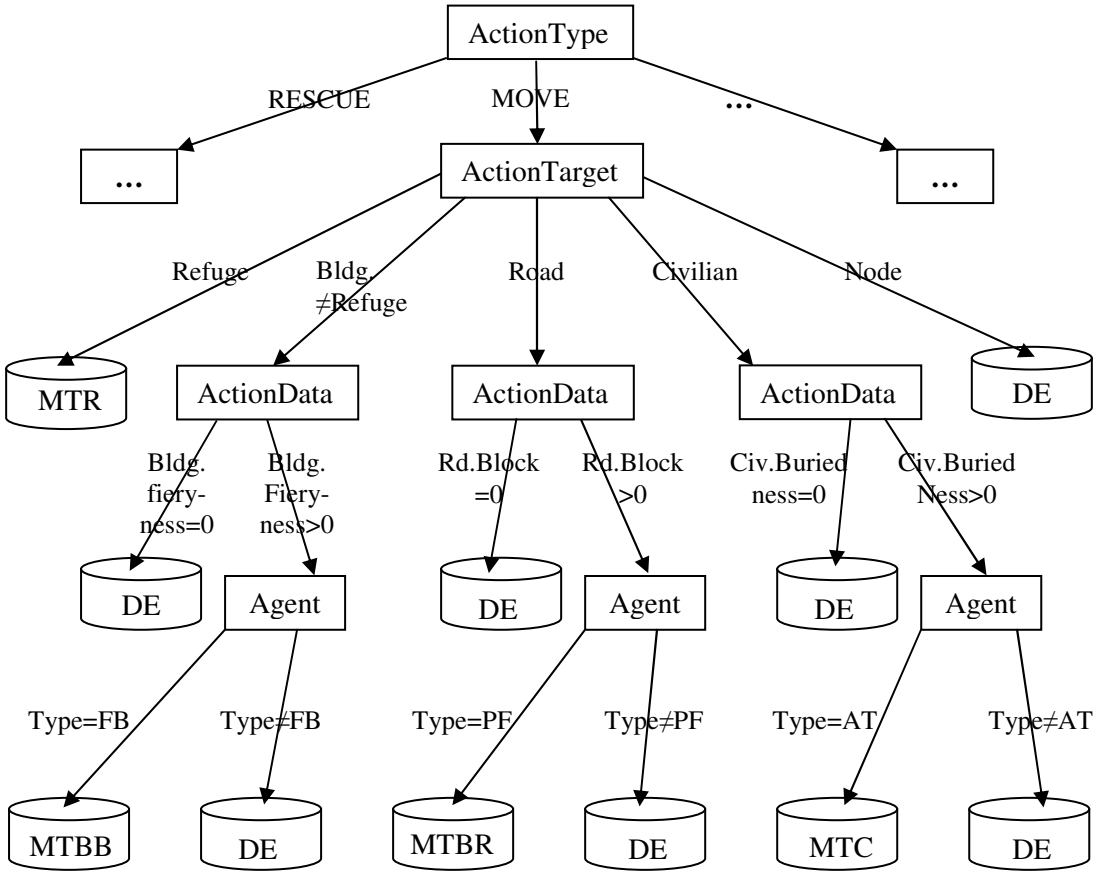


Abbildung 15: Auszug des Feature Decision Tree

6.3.2 Ansatz zur Planerkennung mit dem FDT

Die Identifikation eines Planschrittes mit Hilfe des FDT beginnt im Wurzelknoten. Zur Entscheidung, welchen Planschritt der beobachtete Agent gerade ausführt, wird der FDT abwärts durchlaufen (*top-down traversing*). Zuerst wird das Attribut *ActionType* betrachtet, welches eine eindeutige Zuweisung der Beobachtung zu einem Blatt der Planbibliothek zulässt und somit den größtmöglichen Informationsgehalt für alle Daten darstellt.

Exemplarisch wird an dieser Stelle anhand des Wertes *Move* für *ActionType* (vgl. Abbildung 15) die weitere Vorgehensweise erläutert. Gemäß dem Aufbau des FDT für *ActionType=Move* wird im nächsten Schritt das Attribut *ActionTarget* getestet. Entspricht der Wert von *ActionTarget* dem Objekt *Refuge*, so müssen keine weiteren Attri-

bute getestet werden, der Agent führt auf jeden Fall den Planschritt *MoveToRefuge* (MTR) aus. Ist das Ziel der Aktion Move ein Gebäude, welches keinem *Refuge* entspricht (Building≠Refuge), so muss weiter getestet werden, ob es sich um ein brennendes Gebäude handelt (BldgFieryness>0) oder nicht (BldgFieryness=0). Brennt das Gebäude nicht, wird angenommen das der Agent gerade die Gegend erforscht, was dem Planschritt *DistrictExploration* (DE) entspricht. Handelt es sich jedoch um ein brennendes Gebäude, könnte die Ausführung von *MoveToBurningBuilding* (MTBB) die Intention des Agenten sein, was Bestandteil des Plans *ExtinguishBurningBuilding* (EBB) eines Feuerwehragenten ist. Handelt es sich bei dem Agenten nicht um einen Feuerwehragenten (AgentType≠FB) kommt die Ausführung von *MoveToBurningBuilding* (MTBB) nicht in Frage, da der Agent nicht in der Lage wäre das Feuer zu löschen, und dies keinem sinnvollen Plan entspräche. Stattdessen wird angenommen, dass eine Ambulanz oder Polizei *DistrictExploration* (DE) ausführt, und das brennende Gebäude Teil der zu erforschenden Route ist.
Ebenso wird bei den anderen Zielmöglichkeiten *Road* und *Civilian* verfahren, die in Abhängigkeit von weiteren Attributen entweder zu den Planschritten *MoveToBlockedRoad* (MTBR) bzw. *MoveToCivilian* (MTC) führen oder, wenn keine dieser Eigenschaften zutrifft wieder zu *DistrictExploration* (DE) hinzielen. Entspricht der Zielpunkt der Aktion einem Straßenknoten (Node), kommt keine andere Möglichkeit als *DistrictExploration* (DE) in Frage.

Durch die Bestimmung der Planschritte mittels des FDT werden die Beobachtungen eindeutig Planschritten der Planbibliothek zugeordnet. Durch diese Zuordnung kann der übergeordnete Plan allerdings noch nicht eindeutig bestimmt werden. Zur Bestimmung des zugehörigen Plans werden daher *Zeitstempel* in der Planbibliothek eingeführt.

6.4 Einführung von Zeitstempeln

Bei der Bestimmung eines Plans aus Planschritten existieren Mehrdeutigkeiten in der Planbibliothek, wie das folgende Beispiel verdeutlicht.

Abbildung 16 zeigt die Pläne *HealMe* und *RescueBuriedCivilian* (weiße Kästen) mit ihren Planschritten (graue Kästen). Die Planschritte sind durch sequentielle Kanten verbunden (gestrichelte Pfeile). Durch die Markierung anhand der schwarzen Kreise wird ersichtlich, dass der Planschritt *MoveToRefuge* Bestandteil beider dargestellten Pläne ist.

Der Unterschied besteht lediglich in der zeitlichen Abfolge, in der der Planschritt *MoveToRefuge* in den verschiedenen Plänen auftaucht. Innerhalb des Plans *HealMe* ist der Planschritt *MoveToRefuge* der erste ausgeführte Planschritt. Hingegen ist für den Plan *RescueBuriedCivilian* festgelegt, dass erst dann ein *MoveToRefuge* erfolgen kann, wenn zuvor der Planschritt *LoadCivilian* ausgeführt wurde. Weiterhin ist durch sequentielle Wege festgelegt, welcher Planschritt nach der Ausführung von *MoveToRefuge* erfolgen muss, damit ein gültiger Plan erkannt wird.

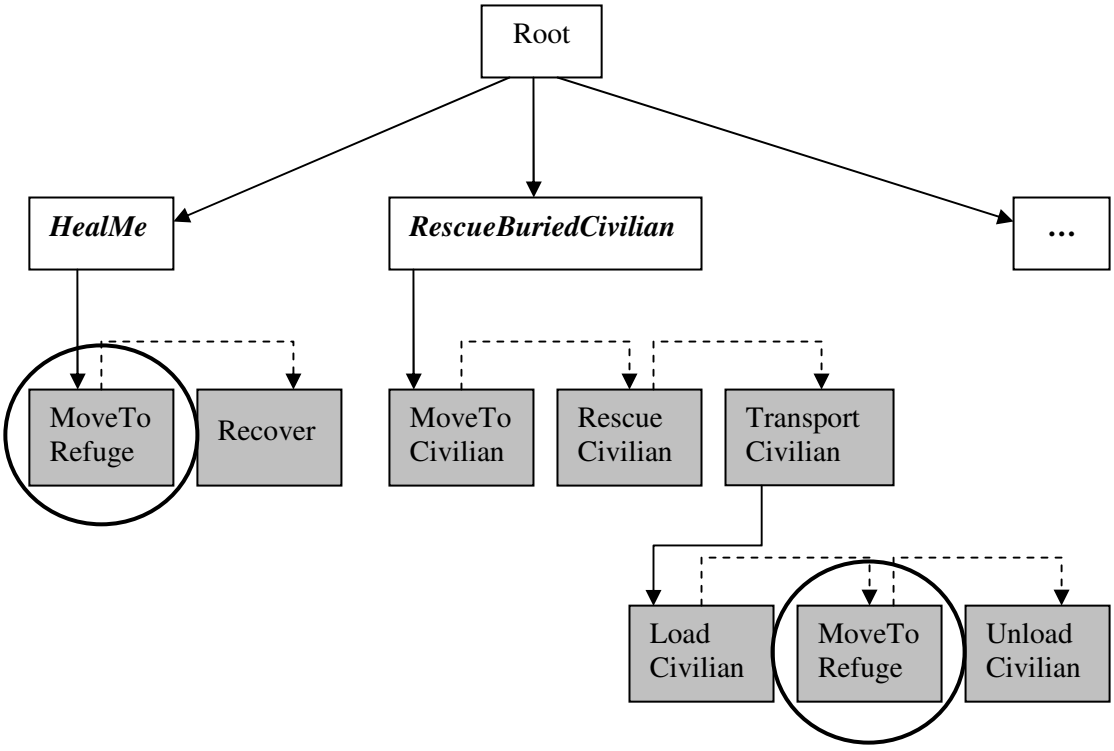


Abbildung 16: Ausschnitt der Planbibliothek mit mehrfach auftretendem Planschritt

6.4.1 Festlegung von Zeitstempeln

Zur Berücksichtigung der zeitliche Reihenfolge in den erkannten Planschritten, werden bei den Zuordnungen der Beobachtungen zu Planschritten der Planbibliothek, die erkannten Planschritte jeweils mit einem *Zeitstempel* markiert. Der Zeitstempel enthält den Zeitpunkt der Rescue-Simulation, der in den Beobachtungen unter dem Attribut *Time* gespeichert ist. (vgl. Kapitel 6.1)
Das Vorgehen lässt sich am Bespiel in Abbildung 17 weiter verdeutlichen.

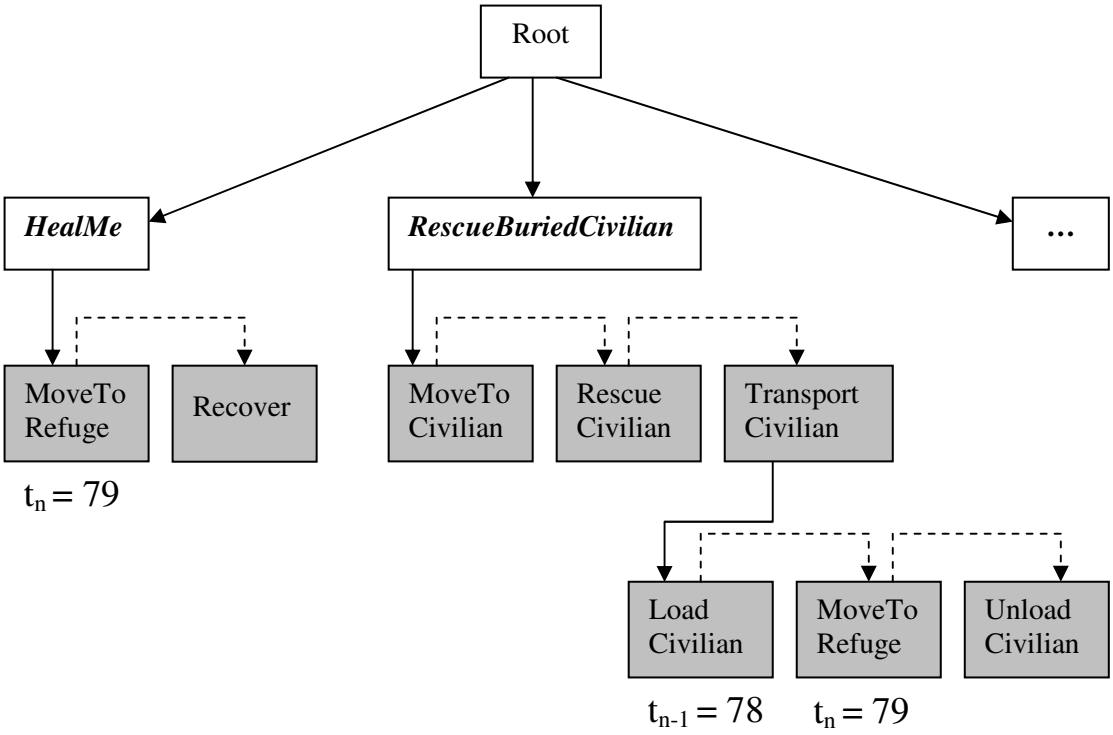


Abbildung 17: Markierung des Planschrittes *MoveToRefuge* mit dem Zeitstempel t_n

Zum Zeitpunkt t_n (t_n = 79) wird ein Planschritt *MoveToRefuge* (MTR) erkannt. Daraufhin wird in der Planbibliothek jedes Vorkommen von MTR unter Berücksichtigung der folgenden Bedingungen mit dem Zeitpunkt t_n markiert:

- Es existiert kein Knoten X mit (X, MTR) ist sequentielle Kante der Planbibliothek, das heißt MTR stellt den ersten Knoten eines sequentiellen Pfades dar.
- Existiert ein Knoten X mit (X, MTR) ist sequentielle Kante der Planbibliothek, dann muss X einen Zeitstempel t_{n-1} besitzen.

Diese Bedingungen gewährleisten, dass Pläne nur dann erkannt werden, wenn die Planschritte in lückenloser zeitlicher Reihenfolge ausgeführt werden.

6.4.2 Ansatz zur Planerkennung mittels Zeitstempel

Der Prozess zur Erkennung eines Plans aus Plansachritten beginnt, nachdem die Zeitstempel t_n eines erfolgreich erkannten Planschritts in der Planbibliothek gesetzt wurden. Zum selben Zeitpunkt t_n wird jeder mit t_n markierte Planschritt betrachtet. Dabei können drei Fälle auftreten:

a) Der markierte Knoten beschreibt den ersten Planschritt eines sequentiellen Pfades.
b) Der markierte Knoten v_n ist Bestandteil eines sequentiellen Pfades derart, dass sowohl v_{n-1} als auch v_{n+1} Bestandteil desselben sequentiellen Pfades sind.
c) Der markierte Knoten stellt den letzten Knoten eines sequentiellen Pfades dar.

In den ersten beiden Fällen wird nichts weiter unternommen, da die erkannten Planschritte noch keinen Plan bilden. Es wird mit dem nächsten Zeitpunkt der Simulation fortgefahren.
Tritt Fall c) ein, so existiert in der Planbibliothek ein sequentieller Pfad, indem alle Knoten mit Zeitstempeln markiert sind, die einen lückenlosen zeitlichen Ablauf bilden. In diesem Fall werden die Zeitstempel des sequentiellen Pfades entlang der zerlegenden Kante zum Elternknoten weitergeleitet und alle Markierungen entlang des sequentiellen Pfades gelöscht. Entspricht die Bezeichnung des Elternknotens einem Plan, so wird dieser als erkannter Plan mit den entsprechenden Start- und Endzeitpunkten an die Recognizer-Ausgabe übertragen. Handelt es sich bei dem Elternknoten um einen weiteren Planschritt, bleibt die Markierung solange vorhanden, bis der zugehörige sequentielle Pfad komplett markiert ist, und die Zeitstempel weitergegeben werden können oder bis die Simulation abgelaufen ist.
Abbildung 18 stellt die Weiterleitung der Zeitstempel dar. Zum Zeitpunkt $t_0 = 79$ wird ein *MoveToRefuge* eines Agenten beobachtet. In der Planbibliothek werden alle konsistenten Instanzen von *MoveToRefuge* mit $t_0 = 79$ markiert (1. $t_0 = 79$). Im nachfolgenden Zeitschritt $t_1 = 80$ wird der Planschritt *UnloadCivilian* vom selben Agenten ausgeführt und folglich wird in der Planbibliothek *UnloadCivilian* mit $t_1 = 80$ (2. $t_1 = 80$) markiert.

Zu jedem Zeitpunkt wird getestet, ob schon ein sequentieller Pfad mit aufeinander folgenden Zeitstempeln markiert ist. Dies ist in Abbildung 18 zum Zeitpunkt $t_1 = 80$ der Fall. Daher werden die Markierungen des sequentiellen Pfades zum Elternknoten geschickt, welcher hier durch *TransportCivilian* dargestellt ist, und alle Markierungen entlang des sequentiellen Pfades werden gelöscht. Es erfolgt die Markierung von *TransportCivilian* mit $t_{-1/1} = 79\text{-}80$ (3. $t_{-1/1} = 79\text{-}80$). Zum selben Zeitpunkt ($t_1 = 80$) wird festgestellt, dass ein weiterer sequentieller Pfad komplett markiert ist. Dieser besteht aus den Planschritten *MoveToCivilian*, *RescueCivilian* und *TransportCivilian*. Somit werden die Zeitstempel dieses sequentiellen Pfades ebenfalls an den Elternknoten weitergeleitet (4. $t_{-3/2} = 76\text{-}80$) und die Markierungen des sequentiellen Pfades werden gelöscht. In diesem Fall entspricht der Knoten einem Plan, dem Plan *RescueBuriedCivilian*. Dieser wird mit den entsprechenden Zeitangaben an die Ausgabe des Recognizer übergeben.

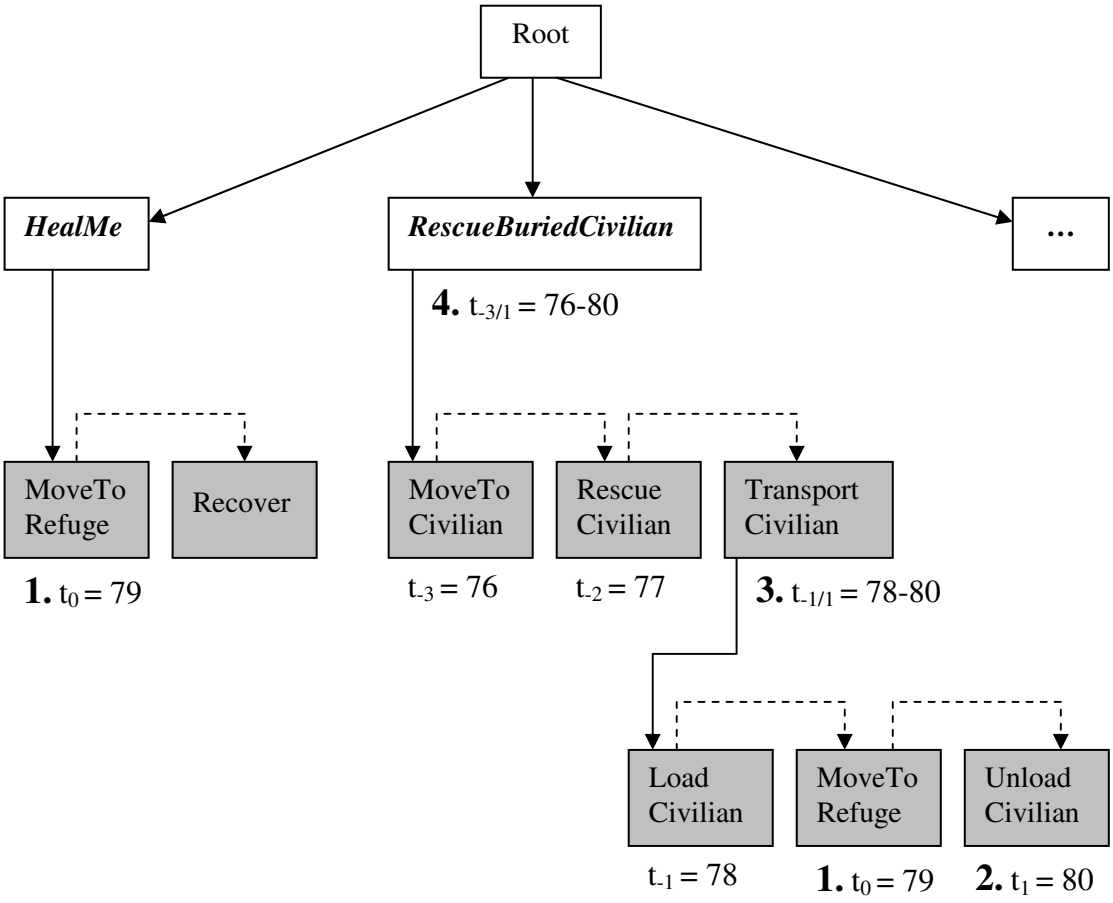


Abbildung 18: Ablauf der Weitergabe von Zeitstempeln

6.5 Planerkennungsalgorithmus

Zusammenfassend werden in diesem Unterkapitel die Überlegungen zur Planerkennung mittels der vorgestellten Eingaben - den *Beobachtungen* (vgl. Kapitel 6.1) - sowie der eingeführten Repräsentationen - wie *Planbibliothek* (vgl. Kapitel 6.2)*, Feature Decision Tree* (vgl. Kapitel 6.3) und *Zeitstempel* (vgl. Kapitel 6.4) - als Algorithmus formuliert.

Der Algorithmus zur Planerkennung wird in fünf inkrementellen Phasen durchlaufen:

1. Abgleichen der Beobachtungen mit den Blättern der Planbibliothek
2. Erkennen der Planschritte mit Hilfe des Feature Decision Tree
3. Markieren erkannter und zeitlich konsistenter Planschritte der Planbibliothek mit Zeitstempeln
4. Weiterleiten der Zeitstempel eines sequentiellen Pfades entlang der zerlegenden Kanten
5. Übertragung der erkannten Pläne an die Recognizer-Ausgabe.

Der Algorithmus wird für jeden Agenten durchlaufen. Die Anzahl der Agenten in einer Rescue-Simulation ist variabel (vgl. Kapitel 3.3). Daher wird zu Beginn des Algorithmus die Anzahl der tatsächlich an der Simulation beteiligten Agenten ermittelt.
Weiter wird der Algorithmus für jeden Zeitschritt durchlaufen. Die Anzahl der Zeitschritte ist für die Rescue-Simulation zwar konstant auf 300 Runden festgelegt (vgl. Kapitel. 3.2.2), allerdings kann diese Größe für den Planerkennungsalgorithmus variiert werden.
Nun wird der erste Schritt des Algorithmus durchgeführt und das Attribut *ActionType* aus einer Beobachtung wird mit den Blättern der Planbibliothek verglichen (vgl. Kapitel 6.2). Mehrdeutigkeiten werden durch den zweiten Schritt des Algorithmus eliminiert. Dazu werden weitere Attribute der Beobachtung laut Vorgabe des Feature Decision Tree getestet bis eindeutig ein Planschritt erkannt wird (vgl. Kapitel 6.3). Nun können mehrere gleiche Planschritte zu verschieden Plänen gehören. Daher werden im nächsten Schritt die erkannten Planschritte auf zeitliche Konsistenz untersucht und gegebenenfalls mit Zeitstempeln versehen (vgl. Kapitel 6.4). Im vierten Schritt wird die Planbibliothek auf markierte sequentielle Pfade getestet und die Zeitstempel eines sequentiellen Pfades werden entlang der zerlegenden Kante des sequentiellen Pfades weitergeleitet (vgl. Kapitel 6.4.2). Entspricht der Ausgangsknoten der zerlegenden Kante einem Plan,

so werden im letzten Schritt der erkannte Plan sowie die Zeitstempel an die Recognizer-Ausgabe übertragen.

In Abbildung 19 ist der Planerkennungsalgorithmus im Pseudo-Code dargestellt.

Planerkennungsalgorithmus (Beobachtung b, Planbibliothek G, FDT):

For all ag_j (j=0 to j=*AnzahlAgenten*) **do**

For all t_i (i=0 to i=S*imulatingTime*) **do**

- Abgleichen der Beobachtung b_{ji} mit Blättern der Planbibliothek
- Erkennen der Planschritte durch *top-down traversing* des *FDT*
- Markieren aller zeitlich konsistenter Planschritte (w_0, …, w_n) mit Zeitstempel t_i

For all w_k (k=0 to k=n) **do**

If („Es existiert ein *seqentiellerPfad* (v_0, …, $\mathbf{w_k}$) in G“) **do**

Weiterleiten (v_0, G)

else return

Weiterleiten (Node v, Planbibliothek G):

If ($X \neq$ *root*(G) mit $(X,v) \in E_z$) **do**

- Markiere X mit TimeStamps von v und w_k
- Lösche TimeStamps von v_0,…, w_k

If ($X \in$ *Plans*(G)) **do**

- Übertrage Markierung von X an Recognizer-Ausgabe
- Lösche Markierung von X

Abbildung 19: Pseudo-Code des Planerkennungsalgorithmus

6.6 Ausgabe der erkannten Pläne

Die Ausgabe erkannter Pläne aus dem Planerkennungsalgorithmus erfolgt mittels der Datei *Plan.html.*
Diese Datei beinhaltet neben der Auflistung aller erkannten Pläne eines Agententeams einen Link zur verwendeten Planbibliothek des Rescueanalysers. Damit soll dem Benutzer des Rescue-Analysewerkzeugs ein Überblick über die Zusammensetzung der erkannten Pläne gegeben werden.
Die Auflistung der erkannten Pläne erfolgt tabellarisch nach Agententypen. Zuerst werden alle Pläne der Ambulanzagenten dargestellt, gefolgt von erkannten Plänen der Feuerwehragenten, sowie Plänen der Polizeiagenten. Anschließend werden jene Pläne aufgeführt, die unabhängig vom Agententyp von jedem Agenten ausgeführt werden können. Abschließend erfolgt eine numerische Auswertung der ausgeführten Pläne, indem die Anzahl aller erkannten (Teil-)Pläne tabellarisch aufgelistet werden.
In Abbildung 20 ist der Aufbau der HTML-Datei skizziert.

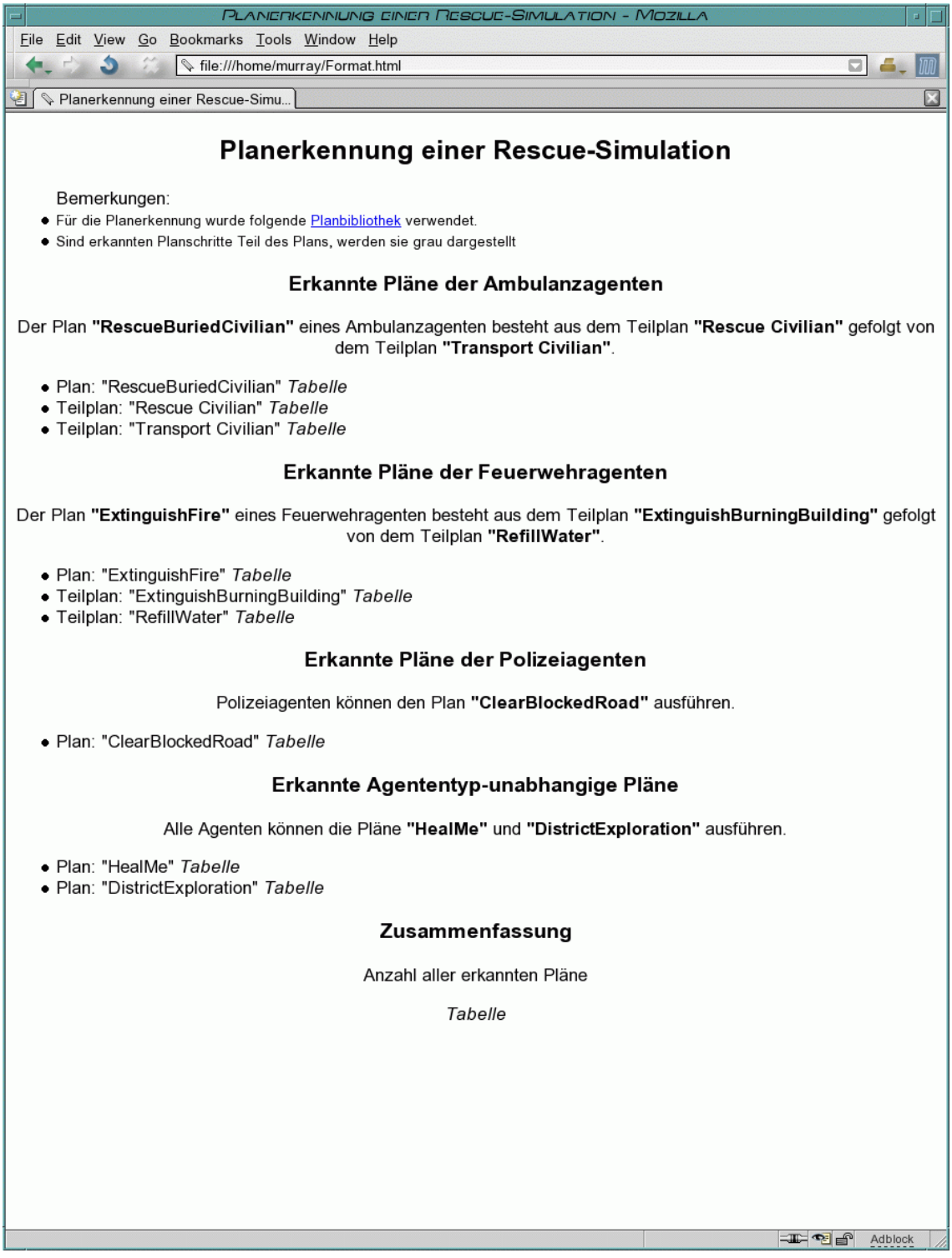


Abbildung 20: Aufbau der HTML-Datei *Plan.html*

Die Darstellung eines erkannten Plans erfolgt in Form einer Tabelle, in der für jeden Agenten die Start- und Endzeitpunkte (Time) der Planausführung angegeben werden. Falls während der Ausführung eines Plans ein weiteres Rescue-Objekt betroffen ist, wird auch dieses innerhalb der Ausgabe beschrieben. Beispielsweise wird durch die Ausführung des Plans *RescueBuriedCivilian* ein verschütteter Zivilist ausgegraben und zur Heilung in ein *Refuge* transportiert. Daher erfolgt bei erkannten Plänen vom Typ *RescueBuriedCivilian* die Darstellung des geretteten Zivilisten anhand seiner eindeutigen Id.
Tabelle 16 zeigt einen Ausschnitt der erkannten Pläne vom Typ *RescueBuriedCivilian*.

- Plan: *RescueBuriedCivilian*

Ambulance (Id:255824460)	Time:10-22 (Civ.Id:88592542)	Time:43-78 (Civ.Id:266178757)	
Ambulance (Id:268013544)	Time:43-68 (Civ.Id:65885550)		
Ambulance (Id:131972107)	Time:3-19 (Civ.Id:266712735)	Time:20-36 (Civ.Id:100651278)	Time:37-46 (Civ.Id:264157380)
Ambulance (Id:125029680)	Time:50-64 (Civ.Id:108901313)		
Ambulance (Id:93435534)	Time:6-22 (Civ.Id:188649328)	Time:23-54 (Civ.Id:133595403)	Time:55-60 (Civ.Id:267688040)
Ambulance (Id:227901454)	Time:16-33 (Civ.Id:133101500)	Time:43-68 (Civ.Id:251640377)	

Tabelle 16: Darstellung erkannter Pläne vom Typ *RescueBuriedCivilian*

Der Plan *RescueBuriedCivilian* kann nur von Ambulanzagenten ausgeführt werden, daher werden auch nur diese in der Tabelle aufgelistet. Weiter besteht der Plan *RescueBuriedCivilian* aus den Planschritten *RescueCivilian* gefolgt von *TransportCivilian* (vgl. Anhang B). Aus der Darstellung in Tabelle 16 ist allerdings nicht ersichtlich über welche Zeitspanne die Planschritte *RescueCivilian* und *TransportCivilian* ausgeführt wurden. Um auch diese als Teilpläne interpretierbaren Planschritte in die Ausgabe der Pläne mit aufzunehmen, werden diese Teilpläne zusätzlich in Form von Tabellen angegeben.

Die Tabellen 17 und 18 zeigen die Teilpläne *RescueCivilian* (Tabelle 17) und *TransportCivilian* (Tabelle 18), die während der Ausführung von *RescueBuriedCivilian* (Tabelle 16) erkannt wurden.

- Teilplan: *RescueCivilian*

Amb.(Id:255824460):	Time:10-18 (Civ.Id:88592542)	Time:23-33 (Civ.Id:100651278)	Time:34-42 (Civ.Id:264157380)	Time:43-75 (Civ.Id:266178757)
Amb.(Id:268013544):	Time:5-18 (Civ.Id:88592542)	Time:19-30 (Civ.Id:133101500)	Time:31-42 (Civ.Id:264157380)	Time:43-65 (Civ.Id:65885550)
Amb.(Id:131972107):	Time:3-15 (Civ.Id:266712735)	Time:20-33 (Civ.Id:100651278)	Time:37-42 (Civ.Id:264157380)	Time:47-60 (Civ.Id:108901313)
Amb.(Id:125029680):	Time:50-60 (Civ.Id:108901313)	Time:65-79 (Civ.Id:201193072)		
Amb.(Id:93435534):	Time:6-19 (Civ.Id:188649328)	Time:23-51 (Civ.Id:133595403)	Time:55-57 (Civ.Id:267688040)	
Amb.(Id:227901454):	Time:5-15 (Civ.Id:266712735)	Time:16-30 (Civ.Id:133101500)	Time:34-42 (Civ.Id:264157380)	Time:43-64 (Civ.Id:251640377)

Tabelle 17: Darstellung erkannter Teilpläne vom Typ *RescueCivilian*

- Teilplan: *TransportCivilian*

Ambulance (Id:255824460):	Time:19-22 (Civ.Id:88592542)	Time:76-78 (Civ.Id:266178757)	
Ambulance (Id:268013544):	Time:66-68 (Civ.Id:65885550)		
Ambulance (Id:131972107):	Time:16-19 (Civ.Id:266712735)	Time:34-36 (Civ.Id:100651278)	Time:43-46 (Civ.Id:264157380)
Ambulance (Id:125029680):	Time:61-64 (Civ.Id:108901313)		
Ambulance (Id:93435534):	Time:20-22 (Civ.Id:188649328)	Time:52-54 (Civ.Id:133595403)	Time:58-60 (Civ.Id:267688040)
Ambulance (Id:227901454):	Time:31-33 (Civ.Id:133101500)	Time:65-68 (Civ.Id:251640377)	

Tabelle 18: Darstellung erkannter Teilpläne vom Typ *TransportCivilian*

Die grau abgebildeten Einträge stellen jene Teilpläne *RescueCivilian* und *TransportCivilian* dar, welche innerhalb des Plans *RescueBuriedCivilian* ausgeführt wurden.
Die schwarzen Einträge beschreiben jene Planschritte, die nicht als Teilpläne des übergeordneten Plans ausgeführt wurden. Im Fall von Tabelle 17 lautet die Interpretation der schwarz dargestellten Einträge, dass ein Teilplan *RescueCivilian* ausgeführt wurde, ohne dass anschließend der Teilplan *TransportCivilian* erfolgte.
Tabelle 18 beinhaltet keine schwarzen Einträge da die Ausführung des Teilplans *TransportCivilian* immer nur im Anschluss an den Teilplan *RescueCivilian* erfolgt. Da-

her sind alle erkannten Teilpläne vom Typ *TransportCivilian* als Bestandteil des Plans *RescueBuriedCivilian* grau dargestellt.

Nach Angabe der Pläne und Teilpläne von Ambulanzagenten erfolgt die Auflistung der ausgeführten Pläne und Teilpläne von den Feuerwehragenten.
Ein Feuerwehragent kann den Plan *ExtinguishFire* ausführen, welcher aus den Teilplänen *ExtinguishBurningBuildung* gefolgt von *RefillWater* besteht (vgl. Anhang B). Die Aufbereitung der erkannten Pläne bzw. Teilpläne entspricht der Darstellung und Erläuterung zu den Tabellen 16-18.
Bei den betroffenen Rescue-Objekten von *ExtinguishFire* und *ExtinguishBurningBuilding* handelt es sich um brennende Gebäude. Im Falle des Teilplans *RefillWater* existiert kein weiteres Rescue-Objekt, welches von der Ausführung von *RefillWater* betroffen ist.
Während der Ausführung von *ExtinguishFire* bzw. *ExtinguishBurningBuildung* können mehrere Gebäude innerhalb eines Plans bzw. eines Teilplans betroffen sein. Um ein Gebäude zu löschen, darf ein Feuerwehragent einen bestimmten Abstand zu diesem Gebäude nicht überschreiten (vgl. Kapitel 3.3). Befinden sich mehrere brennende Gebäude innerhalb dieses Löschradius, so können auch alle diese Gebäude Zielobjekte des ausgeführten (Teil-)Plans sein. Die Angabe dieser Gebäude innerhalb der Tabellen erfolgt durch eine „Auswahlbox“, welche durch anklicken alle Ids der betroffenen Gebäude anzeigt.

Die Tabellen 19 bis 21 zeigen auszugsweise die Darstellung der erkannten Pläne und Teilpläne der Feuerwehragenten.

- Plan: *ExtinguishFire*

FireBrigade (Id:195602368):	Time:56-79 Bldg.Ids: 198691075	Time:85-103 Bldg.Ids: 249438620	
FireBrigade (Id:107865680):	Time:3-39 Bldg.Ids: 220438674	Time:67-79 Bldg.Ids: 268265829	Time:96-120 Bldg.Ids: 198691075
FireBrigade (Id:266282397):	Time:3-36 Bldg.Ids: 249438620	Time:58-78 Bldg.Ids: 259353412	Time:85-97 Bldg.Ids: 746582239

Tabelle 19: Darstellung erkannter Pläne vom Typ *ExtinguishFire*

- Teilplan: *ExtinguishBurningBuilding*

FireBr. (Id:1956 02368):	Time:31-48 Bldg.Ids: 150840025	Time:49-55 Bldg.Ids: 232299889	Time:56-63 Bldg.Ids: 259353412	Time:85-95 Bldg.Ids: 150840025
FireBr. (Id:1078 65680):	Time:3-24 Bldg.Ids: 220438674	Time:51-57 Bldg.Ids: 156771656	Time:67-69 Bldg.Ids: 268265829	Time:96-105 Bldg.Ids: 198691075
FireBr. (Id:26628 2397):	Time:3-20 Bldg.Ids: 249438620	Time:46-50 Bldg.Ids: 247227846	Time:58-62 Bldg.Ids: 259353412	Time:85-91 Bldg.Ids: 746582239

Tabelle 20: Darstellung erkannter Teilpläne vom Typ *ExtinguishBurningBuilding*

- Teilplan: *RefillWater*

FireBrigade (Id:195602368):	Time:64-79	Time:96-103	
FireBrigade (Id:107865680):	Time:25-39	Time:70-79	Time:106-120
FireBrigade (Id:266282397):	Time:3-20	Time:46-50	Time:92-97

Tabelle 21: Darstellung erkannter Teilpläne vom Typ *RefillWater*

Die schwarz dargestellten Einträge des Teilplans *ExtinguishBurningBuilding*, die nicht Bestandteil des Plans *ExtinguishFire* sind, lassen sich wie folgt interpretieren: Ein Feuerwehragent löscht eines oder mehrere brennende Gebäude von seiner Position. Sind die Brände gelöscht und es befindet sich noch Wasser im Tank, wird ein neues Ziel gewählt. Handelt es sich bei diesem wiederum um ein brennendes Gebäude, wird dieses und gegebenenfalls weitere im Löschradius befindlichen Gebäude, gelöscht. Erst wenn während einer Löschaktion eines Feuerwehragenten der Wasservorrat verbraucht ist, oder ein definierter Minimalwert unterschritten wurde, folgt die Ausführung des Teilplans *RefillWater*. Daher erscheinen in Tabelle 21 nur graue Einträge als erkannte Teilpläne von *RefillWater*. Dieser Teilplan wird nur im Anschluss an *ExtinguishBurningBuilding* ausgeführt, und ist somit immer Bestandteil des Plans *ExtinguishFire*.

Nachfolgend werden in der Datei *Plan.html* die Pläne der Polizeiagenten aufgelistet.
Ein Polizeiagent kann den Plan *ClearBlockedRoad* ausführen. Der Plan *ClearBlockedRoad* enthält keine als Teilplan interpretierbaren Planschritte (vgl. Anhang B). Da-

her erfolgt in der Ausgabe nur die Angabe des Plans *ClearBlockedRoad* ohne eine Darstellung weiterer Teilpläne.
Das vom Plan *ClearBlockedRoad* betroffene Rescue-Objekt entspricht einer blockierten Straße, die durch die Ausführung des Plans geräumt wird. Daher erfolgt neben der Beschreibung der Zeitintervalle die Angabe des Objekts *Road* mittels der eindeutigen Id.
In Tabelle 22 ist ein Ausschnitt erkannter Pläne der Polizeiagenten dargestellt.

- Plan: *ClearBlockedRoad*

Police (Id:158846995):	Time:45-47 (Road.Id:259425188)	Time:48-49 (Road.Id:163571012)	Time:50-51 (Road.Id:196081161)
Police (Id:155143925):	Time:51-53 (Road.Id:260029951)	Time:54-56 (Road.Id:234577850)	Time:57-58 (Road.Id:263148956)
Police (Id:228454049):	Time:47-49 (Road.Id:268184005)	Time:61-63 (Road.Id:217701386)	Time:66-68 (Road.Id:123659825)

Tabelle 22: Darstellung erkannter Pläne vom Typ *ClearBlockedRoad*

Anschließend werden innerhalb der Ausgabe jene Pläne aufgelistet, welche von allen Agententypen ausgeführt werden können. Diese bestehen aus Plänen vom Typ *HealMe* sowie Plänen vom Typ *DistrictExploration*. Während der Ausführung dieser Pläne ist kein weiteres Rescue-Objekt durch die Planausführung betroffen. Damit werden in den entsprechenden Tabellen nur die zugehörigen Zeitintervalle dargestellt, zu denen ein Plan erkannt wurde.
Tabellen 23 und 24 zeigen exemplarisch die erkannten Pläne eines Rescue-Agententeams.

- Plan: *HealMe*

Ambulance (Id:255824460):	Time:120-130	Time:289-300	
Ambulance (Id:268013544):	Time:178-189		
Ambulance (Id:131972107):	Time:165-185	Time:213-256	
FireBrigade (Id:195602368):	Time:161-164		
FireBrigade (Id:107865680):	Time:120-122	Time:152-154	Time:258-260
FireBrigade (Id:266282397):	Time:131-133	Time:265-268	Time:299-300
Police (Id:158846995):	Time:199-245		
Police (Id:155143925):	Time:98-101	Time:299-300	
Police (Id:228454049):	Time:266-269		

Tabelle 23: Darstellung erkannter Pläne vom Typ *HealMe*

- Plan: *DistrictExploration*

Ambulance (Id:255824460):	Time:3-9	Time:90-101	Time:129-131
Ambulance (Id:268013544):	Time:98-109	Time:116-118	
Ambulance (Id:131972107):	Time:98-101	Time:206-213	
FireBrigade (Id:195602368):	Time:3-16	Time:19-29	Time:156-159
FireBrigade (Id:107865680):	Time:110-116		
FireBrigade (Id:266282397):	Time: 3-14	Time:18	
Police (Id:158846995):	Time:3-5	Time:8-12	Time:29-36
Police (Id:155143925):	Time:3-19	Time:28-31	Time:38-47
Police (Id:228454049):	Time:3-7	Time:11	Time:3-15

Tabelle 24: Darstellung erkannter Pläne vom Typ *DistrictExploration*

Abschließend beinhaltet die Datei *Plan.html* einen Überblick über die Anzahl aller erkannten (Teil-)Pläne des betrachteten Rescue-Simulationsdurchlaufs. Tabelle 25 zeigt exemplarisch eine numerische Auflistung der erkannten (Teil-)Pläne eines Rescue-Simulationsdurchlaufs.

RescueBuriedCivilian	42
RescueCivilian	75
TransportCivilian	42
ExtinguishFire	34
ExtinguishBurningBuilding	250
RefillWater	34
ClearBlockedRoad	105
HealMe	0
DistrictExploration	265

Tabelle 25: numerische Auflistung aller erkannten (Teil-)Pläne eines Rescue-Simulationsdurchlaufs

7 Experimente und Ergebnisse

Dieses Kapitel beschäftigt sich mit der Validation des Rescue-Analysewerkzeugs und den dabei erzielten Ergebnissen.
Das Rescue-Analysewerkzeug arbeitet unabhängig vom agierenden Agententeam auf Basis der Dateien *action.log* und *rescue.log*, welche vom Rescue-Simulator für jedes agierende Agententeam erzeugt werden. Die schrittweise Analyse kann sowohl visuell auf Basis des Rescueanalyser-Monitors, sowie textbasiert auf Basis der generierten Dateien *Ausgabe.html* und *Plan.html* erfolgen. Die nachfolgend beschriebenen Experimente belegen, dass alle möglichen Pläne, die in der Planbibliothek spezifiziert wurden auch erkannt werden. Die Erkennung von Agentenplänen stellt einen Mehrwert für die Analyse der Agententeams dar.

7.1 Testumgebung

Das Rescue-Analysewerkzeug wird mit den Logdateien der RoboCupRescue-Simulationsliga von 2005 getestet. Die Dateien beinhalten die Informationen aller angetretenen Teams und der durchgeführten Runden innerhalb des Rescue-Simulationswettkampfes. Die Logdateien sind als freier Download verfügbar [30].
Aus diesen Daten können unterschiedliche Testfälle gewonnen werden, die sich aus den verschiedenen Teams und dem unterschiedlichen Schwierigkeitsgrad der Simulationsumgebung ergeben. Die Simulationsumgebungen unterschieden sich in der Größe der zu erforschenden Stadtgebiete, sowie dem Ausmaß der Katastrophensituation. Diese wird durch die Verbreitungsgeschwindigkeit der Brände bestimmt, der Menge und Platzierungen von Blockaden, sowie durch die Anzahl und das Ausmaß der verschütteten Zivilisten und deren Verletzungsgrad.
Zur Validation wurden vier Teams und vier verschiedene Simulationsumgebungen ausgewählt. Die Auswahl der Teams orientierte sich an den erreichten Platzierungen in der RoboCupRescue-Simulationsliga von 2005. Es wurden zwei Teams aus den vorderen Rängen und zwei aus den hinteren Platzierungen ausgewählt.
Folgende Teams wurden für die Tests verwendet [30]:

- *Impossibles* [19]
- *Caspian* [25]
- *Persia*
- *ResQ*

Die Simulationsumgebungen wurden nach der Größe der Stadt, sowie dem Schwierigkeitsgrad der Katastrophensituation ausgewählt. Aus den existierenden Stadtgebieten wurde ein großes (*Kobe4*), ein mittleres (*Voligno*) und ein kleines (*Kobe*) Stadtgebiet erwählt. Der Schwierigkeitsgrad der Katastrophensituationen nimmt von Tag 1 (*Day1*) bis zu den Finaldurchläufen (*Finals*) zu.
Folgende Szenarien wurden zur Validation herangezogen [30]:

- *Day1/Kobe*
- *Day2/Foligno*
- *Finals/Kobe*
- *Finals/Kobe4*

7.2 Ergebnisse

Nachfolgende Ergebnisse konnten aus den Tests der verschiedenen Logdateien gewonnen werden.

7.2.1 Überblick

Tabelle 26 zeigt eine Zusammenfassung über die Anzahl der erkannten Pläne der exemplarisch herangezogenen Teams in verschiedenen Rescue-Simulationsumgebungen.

Zunächst ist erkennbar, dass alle in der Planbibliothek spezifizierten Pläne erkannt werden. Anhand der Anzahl der erkannten Pläne bzw. Teilpläne lassen sich außerdem analytische Schlüsse über die Rescue-Einsätze der unterschiedlichen Teams ableiten.
Beispielsweise ist zu erkennen, dass Pläne vom Typ *HealMe* von einigen der betrachteten Teams nicht oder nur in geringem Umfang ausgeführt werden (vgl. Tabelle 26).
Werden keine Pläne eines bestimmten Typs ausgeführt, so wird als Anzahl der Wert „0“ ausgegeben.
Aus der Anzahl der ausgeführten Teilpläne und deren übergeordneten Pläne können direkt Strategien der einzelnen Teams abgeleitet werden. Zum Beispiel lassen sich aus den vereinzelt auftretenden Werten „0“ innerhalb des ausgeführten Teilplans *TransportCivilian* sowie folglich auch der Anzahl „0“ an Ausführungen des übergeordneten Plans *RescueBuriedCivilian* zwei verschiedene Strategien zur Rettung verschütteter Zivilisten ableiten: Während einige Teams verschüttete Zivilisten in zerstörten Ge-

bäuden ausgraben und anschließend in ein *Refuge* transportieren, werden sie von anderen Teams lediglich ausgegraben und danach sich selbst überlassen.
Eine weitere Schlussfolgerung ergibt sich aus der Anzahl des erkannten Plans *District-Exploration*, welcher in den verschiedenen Rescue-Simulationsumgebungen am meisten variiert. Ist die Anzahl des erkannten Plans *DistrictExploration* sehr hoch, so handelt es sich um den Einsatz in einem größeren Stadtgebiet, in dem die Gegend erst ausreichend erforscht werden muss, bevor die Ausführung der (Teil-)Pläne mit einem genau bestimmten Zielobjekt durchgeführt werden kann.

	Day1/ Kobe: Imp.	*Day 1/ Kobe: ResQ*	*Day1/ Kobe4: Persia*	*Day2/ Foligno: Casp.*	*Day2/ Foligno: Persia*	*Finals/ Kobe: Casp.*	*Finals/ Kobe: Imp.*	*Finals/ Kobe4: Imp.*
Rescue Buried Civilian	42	0	13	20	0	26	40	5
Rescue Civilian	75	183	58	159	80	219	69	22
Transport Civilian	42	0	13	25	0	29	40	6
Extinguish Fire	34	28	28	51	54	123	90	7
Extinguish Burning Building	250	176	148	119	175	364	267	48
Refill Water	34	28	33	53	54	125	92	18
Clear Blocked Road	105	29	160	0	122	0	127	272
HealMe	0	0	7	0	3	38	0	0
District Exploration	265	443	820	737	512	381	273	585

Tabelle 26: numerische Auswertung erkannter (Teil-)Pläne

7.2.2 Detaillierter Vergleich zweier Teams

Bei dem Vergleich zweier verschiedener Teams in identischen Rescue-Simulationsumgebungen können ganz unterschiedliche Strategien in den Rescue-Einsätzen beobachtet werden. Abbildung 21 und 22 zeigen die Ergebnisse der Teams *Impossibles* und *ResQ*, die ein einfaches Katastrophenszenario (*Day1*) in einer kleinen Stadt (*Kobe*) durchlaufen haben.

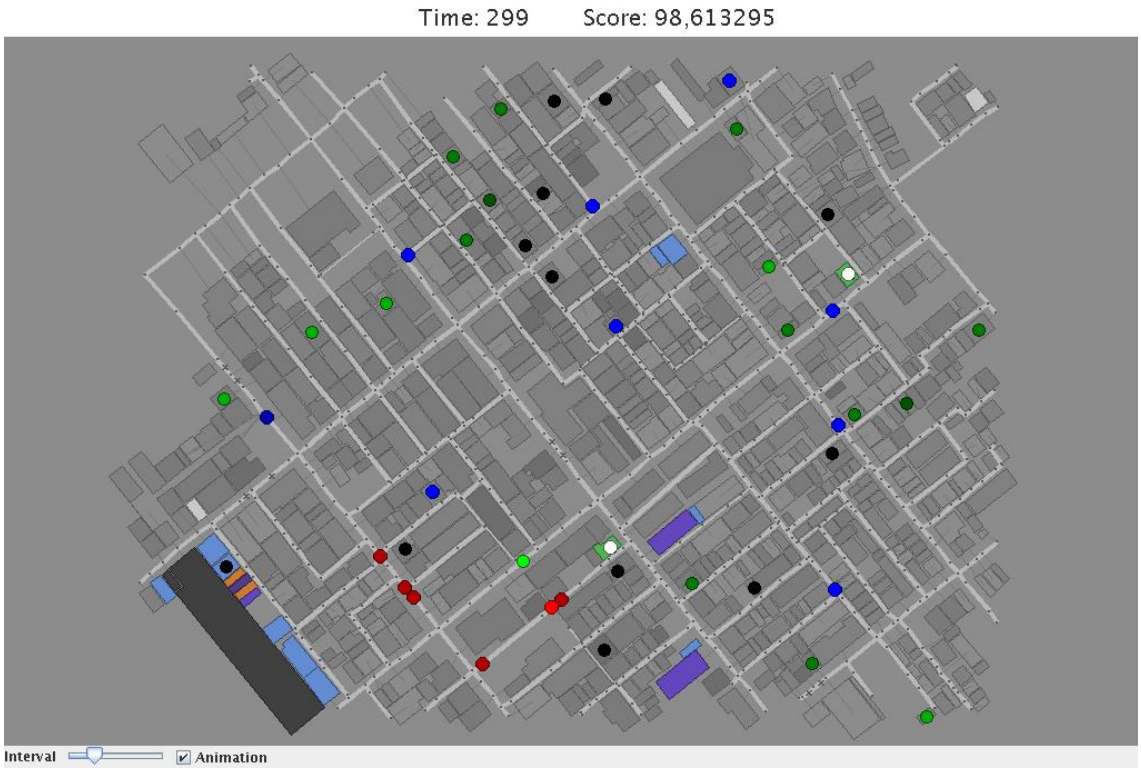


Abbildung 21: Team *Impossibles* zum Zeitpunkt 299 in der Rescue-Simulationsumgebung *Day1/Kobe*

Abbildung 22: Team *ResQ* zum Zeitpunkt 299 in der Rescue-Simulationsumgebung *Day1/Kobe*

Die Abbildungen 21 und 22 zeigen das Ende des Rescue-Simulationsdurchlaufs in der Darstellung des Rescueanalyser-Monitors zum Zeitpunkt 299.
Anhand der Abbildungen ist erkennbar, dass die beiden Teams sehr unterschiedliche Ergebnisse in diesem Rescue-Simulationsdurchläufen erzielt haben. Während die Simulationsumgebung des Teams *Impossibles* zum Zeitpunkt 299 der Rescue-Simulation (Abbildung 21) kaum verbrannte Häuser und nur wenig verstorbene Zivilisten aufweist, sind innerhalb der Simulationsumgebung des Teams ResQ zum selben Zeitpunkt (Abbildung 22) mehr als die Hälfte aller Häuser verbrannt und zahlreiche Zivilisten verstorben.

Durch die Beobachtung der kompletten Rescue-Simulation im Rescueanalyser-Monitor kann ein Überblick über die Strategien der einzelnen Teams gewonnen werden. Besonders gut kann die Aufteilung der einzelnen Agentengruppen beobachtet werden: Die Aufteilung von Feuerwehragenten auf mehrere Brandherde (vgl. Abbildung 23) oder die Verteilung der Polizeiagenten zu den Blockaden rund um die verschiedenen Brandherde können entscheidend zum Erfolg eines Teams beitragen.

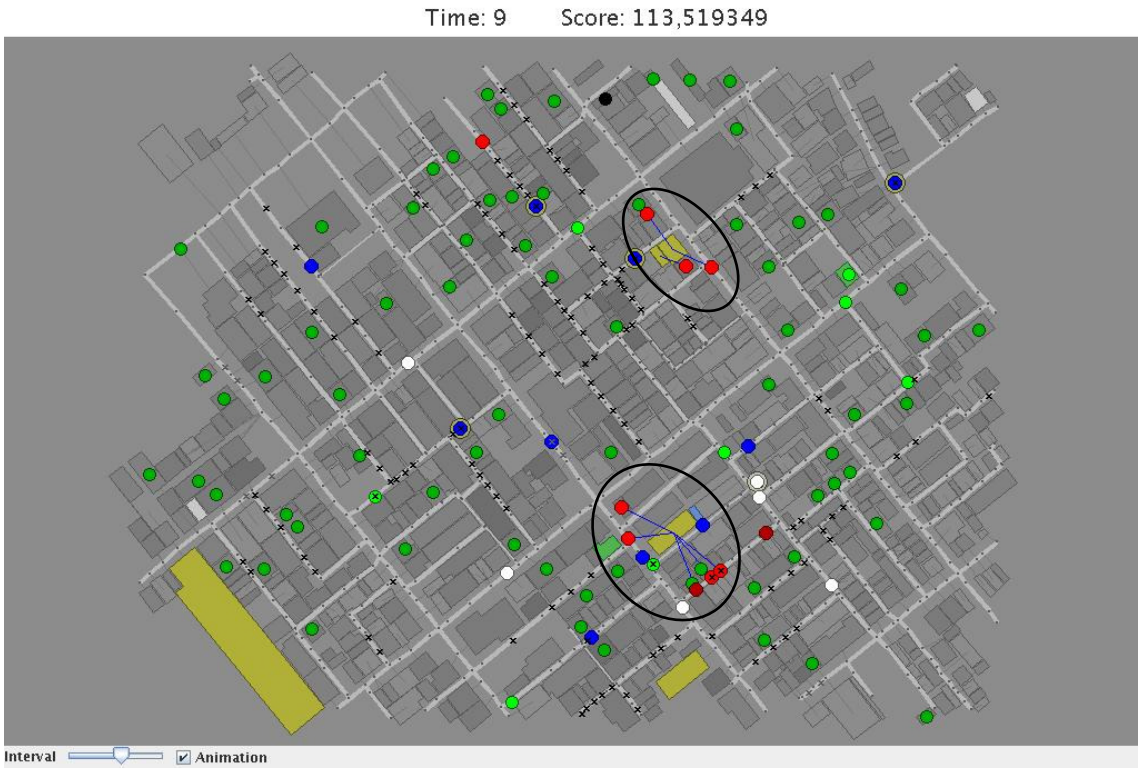


Abbildung 23: Aufteilung der Feuerwehragenten auf mehrere Brandherde

Eine detaillierte Analyse der erkannten (Teil-)Pläne kann anhand der generierten Ausgabedatei *Plan.html* erfolgen, in der die erkannten (Teil-)Pläne mit genauen Zeitangaben und Zugehörigkeit zu den einzelnen Agenten angegeben sind. Mittels dieser Angaben kann genau nachvollzogen werden, in welchen Zeitspannen ein (Teil-)Plan vermehrt ausgeführt wurde. Beispielsweise zeigt der Vergleich der erkannten Pläne vom Typ *DistrictExploration* der Teams *ResQ* und *Impossibles* die unterschiedliche Ausführung des Plans (vgl. Abbildung 24).

Vergleich der Ausführung von *DistrictExploration*:

Während Agenten des Teams *ResQ* den Plan *DistrictExploration* über längere Perioden besonders zu Beginn der Rescue-Simulation ausführen, wird derselbe Plan von den Agenten des Teams *Impossibles* in kürzeren Abständen ausgeübt.

Dementsprechend werden weniger (Teil-)Pläne anderen Typs von den Agenten des Teams *ResQ* ausgeführt als von den Agenten des Teams *Impossibles*.

AT (Id:182428972):	Time: 3-9	Time: 12-14
AT (Id:256102096):	Time: 4-12	Time: 25-29
FB (Id:251621028):	Time: 3-17	Time: 20-29
FB (Id:62113820):	Time: 3-14	Time: 16-19
FB (Id:268097163):	Time: 3-10	Time: 13-17
FB (Id:46071506):	Time: 3-15	Time: 17-23
PF (Id:257368549):	Time: 3-6	Time: 8-15
PF (Id:267342325):	Time: 3-8	Time: 10-18

Plan *DE* des Teams *ResQ*

AT (Id:225825296):	Time: 3-4	Time: 24-24
AT (Id:132113564):	Time: 3-4	Time: 11-12
FB (Id:54256862):	Time: 3-6	Time: 45-45
FB (Id:126820996):	Time: 3-5	Time: 60-61
FB (Id:268403660):	Time: 3-4	Time: 28-29
FB (Id:133035573):	Time: 3-4	Time: 17-17
PF (Id:267656983):	Time: 3-5	Time: 11-11
PF (Id:159369467):	Time: 3-7	Time: 12-12

Plan *DE* des Teams *Impossibles*

Abbildung 24: Vergleich der Ausführung des Plans *DistrictExploration* (DE)

Vergleich der Ausführung von *ExtinguishBurningBuilding*:

Abbildung 25 zeigt Ausschnitte des erkannten Teilplans *ExtinguishBurningBuilding* (EBB) beider Teams.

Es ist deutlich zu erkennen, dass die Ausführung des Teilplans EBB von Feuerwehragenten des Teams *Impossibles* zu einem früheren Zeitpunkt in der Rescue-Simulation begonnen wird als die Ausführung von EBB der Feuerwehragenten des Teams *ResQ*. Außerdem ist erkennbar, dass der Teilplan EBB von Agenten des Teams *Impossibles* über längere Zeiträume ausgeführt wird.

Dies hat zur Folge, dass sich das Feuer in den Zeitspannen, in denen Feuerwehragenten des Teams *ResQ* noch *DistrictExploration* ausführen, weiter verbreitet, was die völlige Eindämmung des Feuers im Laufe der restlichen Simulation erschwert. Somit kann es zu einem Übergreifen des Feuers kommen, wie auch Abbildung 22 belegt.

FB (Id:251621028):	Time:18-31 Bldg.Ids 199660348	Time: 32-33 Bldg.Ids 199660348	Time:91-97 Bldg.Ids 197616282
FB (Id:62113820):	Time:20-26 Bldg.Ids 264228399	Time:31-33 Bldg.Ids 124751394	Time:53-65 Bldg.Ids 124751394
FB (Id:268097163):	Time:20-25 Bldg.Ids 264228399	Time:27-28 Bldg.Ids 162474680	Time:54-63 Bldg.Ids 162474680
FB (Id:46071506):	Time:24-27 Bldg.Ids 199660348	Time:59-66 Bldg.Ids 199660348	Time:73-85 Bldg.Ids 199660348

Plan EBB des Teams *ResQ*

FB (Id:54256862):	Time:7-15 Bldg.Ids 133579687	Time:16-24 Bldg.Ids 133579687	Time:40-47 Bldg.Ids 133579687
FB (Id:126820996):	Time:6-11 Bldg.Ids 199042184	Time:12-15 Bldg.Ids 199042184	Time:16-23 Bldg.Ids 192929545
FB (Id:268403660):	Time:5-11 Bldg.Ids 199042184	Time:12-15 Bldg.Ids 133579687	Time:16-26 Bldg.Ids 133579687
FB (Id:133035573):	Time:5-15 Bldg.Ids 133579687	Time:16-24 Bldg.Ids 132878055	Time:41-52 Bldg.Ids 132878055

Plan EBB des Teams *Impossibles*

Abbildung 25: Vergleich der Ausführung des Teilplans *ExtinguishBurningBuilding* (EBB)

Vergleich der Ausführung von *RescueBuriedCivilian*:

Weiterhin wird nun vergleichend die Ausführung des Plans *RescueBuriedCivilian* beider Ambulanzagententeams ermittelt.

In Abbildung 26 sind Ausschnitte des erkannten Plans *RescueBuriedCivilian* (RBC) und deren Teilpläne *RescueCivilian* (RC) und *TransportCivilian* (TC) dargestellt.

AT (Id:182428972):
AT (Id:256102096):
AT (Id:266835063):

Plan RBC des Teams *ResQ*

AT (Id:225825296):	Time:5-47 (Civ.Id: 268434378)	Time:48-56 (Civ.Id: 249515164)
AT (Id: 132113564):	Time:5-18 (Civ.Id: 268113066)	Time:19-52 (Civ.Id: 241143878)
AT (Id: 268432282):	Time:15-39 (Civ.Id: 128815400)	Time:53-87 (Civ.Id: 192581873)

Plan RBC des Teams *Impossibles*

AT (Id:182428972):	Time:10-11 (Civ.Id: 208538600)	Time:15-18 (Civ.Id: 161190149)
AT (Id:256102096):	Time:13-24 (Civ.Id: 208538600)	Time:30-33 (Civ.Id: 256863136)
AT (Id:266835063):	Time:7-14 (Civ.Id: 256863136)	Time:26-32 (Civ.Id: 97236234)

Teilplan RC des Teams *ResQ*

AT (Id:225825296):	Time:5-43 (Civ.Id: 268434378)	Time:48-52 (Civ.Id: 249515164)
AT (Id: 132113564):	Time:5-14 (Civ.Id: 268113066)	Time:19-49 (Civ.Id: 241143878)
AT (Id: 268432282):	Time:5-14 (Civ.Id: 268113066)	Time:15-35 (Civ.Id: 128815400)

Teilplan RC des Teams *Impossibles*

AT (Id:182428972):
AT (Id:256102096):
AT(Id:266835063):

Teilplan TC des Teams *ResQ*

AT (Id:225825296):	Time:44-47 (Civ.Id: 268434378)	Time:53-56 (Civ.Id: 249515164)
AT (Id: 132113564):	Time:15-18 (Civ.Id: 268113066)	Time:50-52 (Civ.Id: 241143878)
AT (Id: 268432282):	Time:36-39 (Civ.Id: 128815400)	Time:85-87 (Civ.Id: 192581873)

Teilplan TC des Teams *Impossibles*

Abbildung 26: Vergleich der Ausführung der (Teil-)Pläne RBC, RC und TC

In Abbildung 26 wird deutlich, dass die Rettung von Zivilisten durch Ambulanzagenten des Teams *ResQ* nur durch Ausgraben aus verschütteten Gebäuden besteht. Geborgene Zivilisten werden niemals in ein *Refuge* transportiert. Sind die Zivilisten so schwer verletzt, dass sie nicht von selbst die *Refuges* aufsuchen können, so sterben sie an Ort und Stelle.

Im Unterschied dazu graben die Ambulanzagenten des Teams *Impossibles* die verschütteter Zivilisten aus und transportieren anschließend einen Großteils der ausgegrabenen Zivilisten in *Refuges*. Da dies nicht für alle Zivilisten gilt (vgl. Abbildung 26, Tabelle: Teilplan RC des Teams *Impossibles,* schwarzer Tabelleneintrag) kann davon ausgegangen werden, dass die Strategie des Teams definierte Bedingungen vorsieht, nach denen ein ausgegrabener Zivilist entweder in ein *Refuge* transportiert wird oder sich selbst überlassen wird. Ein guter Indikator für diese Entscheidung ist der Gesundheitswert des ausgegrabenen Zivilisten, welcher in der Ausgabedatei *Ausgabe.html* der Analyser-Komponente genau eingesehen werden kann.
Tabelle 27 zeigt exemplarisch die entsprechenden Einträge in der Datei *Ausgabe.html*, aus denen ersichtlich wird, dass der ausgegrabene Zivilist anschließend nicht in ein *Refuge* transportiert wird. Stattdessen ermöglicht der hohe Gesundheitswertes des Zivilisten das eigenständige Aufsuchen eines *Refuges*.

14	AT (Id:199197927)	Building (Id:191015316)	10000	RES-CUE	Civilian (Id:268113066)	Civ.Buriedness: 1, Civ.HP: 9075
…	…	…	…	…	…	…
15	Civilian (Id:268113066)	Building (Id:191015316)	9075	MOVE	Refuge (Id:268170126)	null

Tabelle 27: Ausgraben eines Zivilisten ohne Transport

Vergleich der Ausführung von *ClearBlockedRoad*:

Es folgt die Betrachtung der Planausführung *ClearBlockedRoad* der Polizeiagenten beider Teams. Abbildung 27 zeigt die Ausschnitte des erkannten Plans *ClearBlockedRoad* (CBR) der Polizeiagenten der Teams *ResQ* und *Impossibles*.
Dabei ist deutlich zu erkennen, dass die Polizeiagenten des Teams *Impossibles* weitaus öfter gezielt Straßenblockaden entfernen, als die Polizeiagenten des Teams *ResQ*. Gezielt bedeutet hier, dass der Plan *ClearBlockedRoad* mit dem Zielobjekt einer blockierten Strasse ausgeführt wird. Hingegen können blockierte Strassen auch zufällig beispielsweise bei der Ausführung des Plans *DistrictExploration* vorgefunden und geräumt werden. Dabei wird kein Plan vom Typ *ClearBlockedRoad* ausgeführt. Die Zufälligkeit der Ausführung erkennt man in der Datei *Ausgabe.html* zu jenen Zeitpunkten, in denen die Aktion *Clear* ausgeführt wird, während in der Datei *Plan.html* der Plan *ClearBlockedRoad* zu diesen Zeitpunkten nicht nachgewiesen werden kann (vgl. Abbildung 28).

PF (Id:134185529):	Time:9-10 (Road.Id: 150968293)	Time:144-147 (Road.Id: 251077116)	
PF (Id:257368549):			
PF (Id:134029423):	Time:8-9 (Road.Id: 180346823)	Time:10-11 (Road.Id: 83606579)	Time:51-52 (Road.Id: 201049815)
PF (Id:264099417):	Time:9-12 (Road.Id: 251618543)		

Plan CBR des Teams *ResQ*

PF (Id:267656983):	Time:7-8 (Road.Id: 96454015)	Time:9-10 (Road.Id: 132441284)	Time:14-15 (Road.Id: 267453032)
PF (Id:159369467):	Time:8-9 (Road.Id: 130020272)	Time:12-13 (Road.Id: 213880540)	Time:14-16 (Road.Id: 267875323)
PF (Id:267475462):	Time:5-6 (Road.Id: 130625149)	Time:7-8 (Road.Id: 132886814)	Time:9-11 (Road.Id: 268298767)
PF (Id:167735884):	Time:18-20 (Road.Id: 197640528)	Time:24-25 (Road.Id: 162945856)	Time:26-27 (Road.Id: 255250217)

Plan CBR des Teams *Impossibles*

Abbildung 27: Vergleich der Ausführung des Plans *ClearBlockedRoad* (CBR)

PF (Id:134185529):	Time:9-10 (Road.Id:150968293)	Time:144-147 (Road.Id:251077116)

Ausschnitt der Datei *Plan.html* des Polizeiagenten (Id: 134185529)

38	PoliceForce (Id:134185529)	Road (Id: 259245716)	10000	CLEAR	Road (Id:259245716)	Rd.Block: 2175
…	…	…	…	…	…	…
41	PoliceForce (Id:134185529)	Road (Id: 266334111)	10000	CLEAR	Road (Id:266334111)	Rd.Block: 2175
…	…	…	…	…	…	…
65	PoliceForce (Id:134185529)	Road (Id:200728102)	10000	CLEAR	Road (Id:200728102)	Rd.Block: 2175
…	…	…	…	…	…	…

Ausschnitt der Datei *Ausgabe.html* des Polizeiagenten (Id: 134185529)

Abbildung 28: Vergleich des Plans *ClearBlockedRoad* zur zufälligen Bereinigung einer Straße

7.2.3 Zusammenfassung der Ergebnisse

Anhand der durchgeführten Tests und der erzielten Ergebnisse können folgende Resultate abgeleitet werden:

- Das Rescue-Analysewerkzeug arbeitet unabhängig vom agierenden Agententeam.
- Die möglichen durchführbaren, sinnvollen Aktionsfolgen der Rescue-Agenten wurden in der Planbibliothek als Pläne spezifiziert und werden im Laufe des Planerkennungsprozesses korrekt klassifiziert.
- Es kann zwischen zufällig ausgeführten Aktionen und gezielten - als Plan definierten - Aktionsfolgen unterschieden werden.
- Neben der Erkennung und Darstellung der von Agenten ausgeführten (Teil-)Pläne mittels der Recognizer-Ausgabe *Plan.html* können mit Hilfe der Datei *Ausgabe.html* zusätzliche Details wie beispielsweise Ausführungsbedingungen erkannter (Teil-)Pläne abgeleitet werden.

Allgemein konnte gezeigt werden, dass mittels der verschiedenen Ausgaben des Rescue-Analysewerkzeugs, bestehend aus der grafischen Ausgabe durch den Monitor sowie den textbasierten Ausgaben der Recognizer- und Generatorkomponente, die implementierten Strategien eines Rescue-Agententeams sehr detailliert erkannt und nachvollzogen werden können.

Teil II: Zusammenfassung

Im zweiten Teil dieser Arbeit wurde die Entwicklung und Realisierung des Analysewerkzeugs für die RoboCupRescue-Simulation beschrieben. Zunächst wurden in Kapitel 4 die Anforderungen an das Analysewerkzeug - den Rescueanalyser - definiert und die Ausgangsbasis für die Entwicklung erörtert. Weiterhin wurde eine Abgrenzung zu bisherigen Entwicklungen im Bereich der Rescue-Simulation geschaffen und damit die Stärken und Vorteile des Analysewerkzeugs dargelegt. Anschließend wurden die Architektur und die Arbeitsweise des Rescueanalysers erläutert.
Im fünften Kapitel erfolgte die detaillierte Darstellung der Konzeption zur Entwicklung des Analysewerkzeugs. Durch die Beschreibung der einzelnen Komponenten des Rescueanalysers, die Darstellung deren Arbeitsweise sowie der entsprechenden Ein- und Ausgabeschnittstellen, wurde ein Einblick in die Feinkonzeption vermittelt.
Das nachfolgende sechste Kapitel behandelte die detaillierte Beschreibung des Planerkennungsprozesses. Neben der Eingabe für den Planerkennungsprozess wurden die weiteren erforderlichen Repräsentationen - Planbibliothek, Feature Decision Tree und Zeitstempel - erörtert. Des Weiteren wurde die Arbeitsweise des Planerkennungsprozesses unter Verwendung der eingeführten Repräsentationen aus algorithmischer Sicht behandelt. Die Ausgabe des Planerkennungsprozesses wurde anhand von Beispielen erkannter Pläne der Rescue-Agenten dargestellt.
Abschließend erfolgte in Kapitel 7 die Validation des entwickelten Analysewerkzeugs anhand von Testdurchläufen verschiedener Rescue-Teams und die exemplarische Auswertung der erzielten Ergebnisse.

8 Rück- und Ausblick

Dieses Kapitel stellt abschließend eine Zusammenfassung des Inhalts dieser Arbeit dar und bietet ferner Anregungen zu möglichen Erweiterungsentwicklungen des realisierten Analysewerkzeugs.

8.1 Resümee

Innerhalb dieser Arbeit wurde ein Analysewerkzeug, der Rescueanalyser, für die RoboCupRescue-Simulation entwickelt. Im Vordergrund dabei stand die Entwicklung eines Planerkennungsalgorithmus, welcher die beobachtbaren Handlungen der Agenten mit Plänen einer spezifizierten Planbibliothek abgleicht und erkennt.
Der Planerkennungsprozess wurde auf Basis der im Artikel *Fast and Complete Symbolic Plan Recognition* von Dorith Avrahami-Zilberbrand und Gal A. Kaminka vorgestellten Methoden zur Planerkennung bei modernen Roboteranwendungen umgesetzt.
Während der Implementation des Rescueanalysers galt es, eine Vielzahl von untergeordneten Problemstellungen zu lösen. Zum einen mussten die Eingabedaten des Rescueanalysers in ein für die Analyse und Planerkennung verwendbares Format gebracht werden. Weiterhin galt es die Entwicklung der benötigten Repräsentationen an die vorhandenen Daten anzupassen, und schließlich mussten geeignete Ausgabestrukturen für die Präsentation der generierten Ergebnisse erarbeitet werden.
Die Funktionalität und Funktionsfähigkeit des entwickelten Analysewerkzeugs wurde schließlich anhand von unterschiedlichen Testumgebungen und Beispielen dargelegt.

8.2 Zukünftige Arbeiten

Zum Abschluss dieser Arbeit werden noch einige Ideen zu möglichen Erweiterungen des Rescueanalysers präsentiert.
Das entwickelte Rescue-Analysewerkzeug für die RoboCupRescue-Simulation besteht aus mehreren Komponenten, die verschiedene Erweiterungsmöglichkeiten bieten.

Die Planbibliothek ermöglicht das Hinzufügen weiterer Pläne, sowie die Detaillierung vorhandener Pläne. Eine mögliche Erweiterung der Planbibliothek stellt die Aufspaltung des Plans *ExtinguishFire* in die Teilpläne *ExtinguishNearestFire*, *ExtinguishLargestFire* und *ExtinguishRandomFire* dar. Zur Erkennung dieser Teilpläne müssen die

verschiedenen Brände der Simulation in bestimmte Gruppen (*Cluster*) unterteilt werden. Diese ermöglichen die Klassifizierung der einzelnen Brände in die unterschiedlichen Kategorien: *NearestFire*, *LargestFire* und *RandomFire*.
Eine Klasse zur Zusammenfassung mehrerer brennender Gebäude zu einem Cluster ist in Ansätzen bereits innerhalb der erarbeiteten Version des Rescueanalysers unter der Bezeichnung *Cluster* implementiert. Bei künftiger Nutzung differenzierter Brandkategorien kann die Auswertung dieser Kriterien zusätzlich implementiert werden.

Eine Erweiterungsmöglichkeit an anderer Stelle besteht in der Entwicklung einer Schnittstelle zu dem neuesten Datenformat der Logdateien *rescue.log* und *action.log*. Aufgrund der ständigen Weiterentwicklung des Rescue-Simulators ändert sich gelegentlich auch das Datenformat der vom Simulator generierten Logdateien. Zu Beginn der Implementation des Rescue-Analysewerkzeugs stand eine endgültige Version noch nicht fest. Der innerhalb dieser Arbeit entwickelte Rescueanalyser ist in der Lage die Logdateien der Simulatorversionen 0.44 bis 0.48 zu verarbeiten.
Neben der Möglichkeit der Entwicklung einer Schnittstelle zur Adaption des neuen Logdatei Formats, besteht aufgrund der modularen Struktur des Rescueanalyser ebenfalls die Möglichkeit die entsprechende Komponente - die Klasse *Logparser* - an das neue Datenformat anzupassen.

Das entwickelte Rescue-Analysewerkzeug ist in der Lage, die implementierten Strategien von Rescue-Agententeams detailliert zu erkennen und einer vergleichenden Analyse zu unterziehen. Damit ist die Basis gelegt für die künftige Optimierung der Rescue-Strategien und die effektive Einsatz-Steuerung von Agenten-Teams.

Anhang A: Erweiterungen des Kernel

In diesem Anhang werden die Änderungen im Quellcode der Kernel-Komponente des Rescue-Simulators beschrieben, die in Kapitel 5.5 erwähnt wurden.
Die Modifikation des Kernels bewirkt die Darstellung der Kommunikationsbefehle in der Ausgabedatei *Ausgabe.html* des Rescueanalysers.

Die Methode *sendToSimulator()* der Datei *System.cxx* muss folgendermaßen abgeändert werden:

Original:

```
void System::sendToSimulators() {
[…]
  for(; it2 != it->second.end(); it2++) {
        Agent* agent = *it2;
        RescueObject* object = agent->asObject();
        if(agent->getCommandType() == it->first) {
              m_outputBuffer.writeInt32(object->id());
              agent->outputCommand(m_outputBuffer);
              agent->resetCommand();
        }
  }
[…]
}
```

Neu: Löschen oder Auskommentieren der grau markierten Zeilen

```
void System::sendToSimulators() {
[…]
  for(; it2 != it->second.end(); it2++) {
        Agent* agent = *it2;
        RescueObject* object = agent->asObject();
        //if(agent->getCommandType() == it->first) {
              m_outputBuffer.writeInt32(object->id());
              agent->outputCommand(m_outputBuffer);
              //agent->resetCommand();
        //}
  }
[…]
}
```

Die Methode *processCommand(…)* der Datei *RescueSystem.cxx* muss wie folgt verändert werden:

Original:

```
void RescueSystem::processCommand(Agent* agent, Header header, INT_32 size, Input&
input/*, const LongUDPConnection& from*/)  {
[…]
  if(header == AK_SAY) {
        say(sender, buffer, size);
  } else {
        tell(sender, buffer, size);
  }
}
[…]
}
```

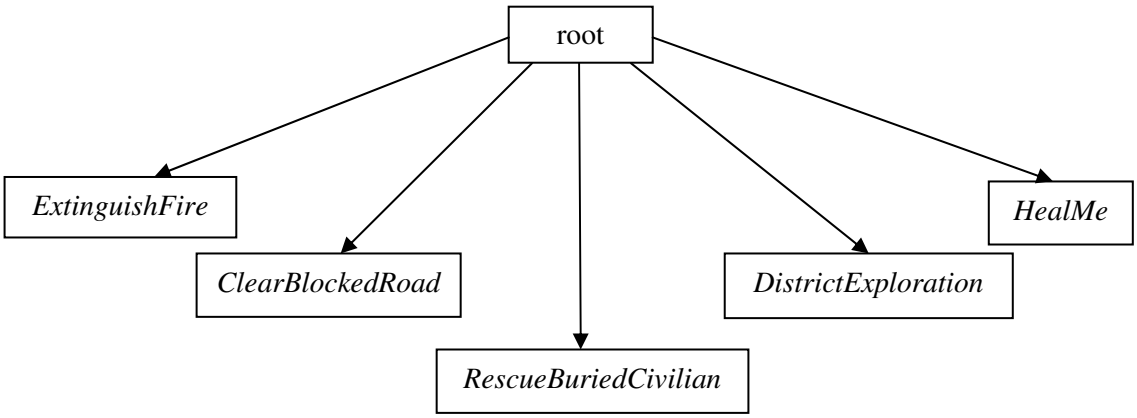
Neu: Hinzufügen der grau markierten Zeile

```
void RescueSystem::processCommand(Agent* agent, Header header, INT_32 size, Input&
input/*, const LongUDPConnection& from*/)  {
[…]
  if(header == AK_SAY) {
        say(sender, buffer, size);
  } else {
        tell(sender, buffer, size);
  }
  System::processCommand(agent, header, size, input/*, from*/);
}
[…]
}
```

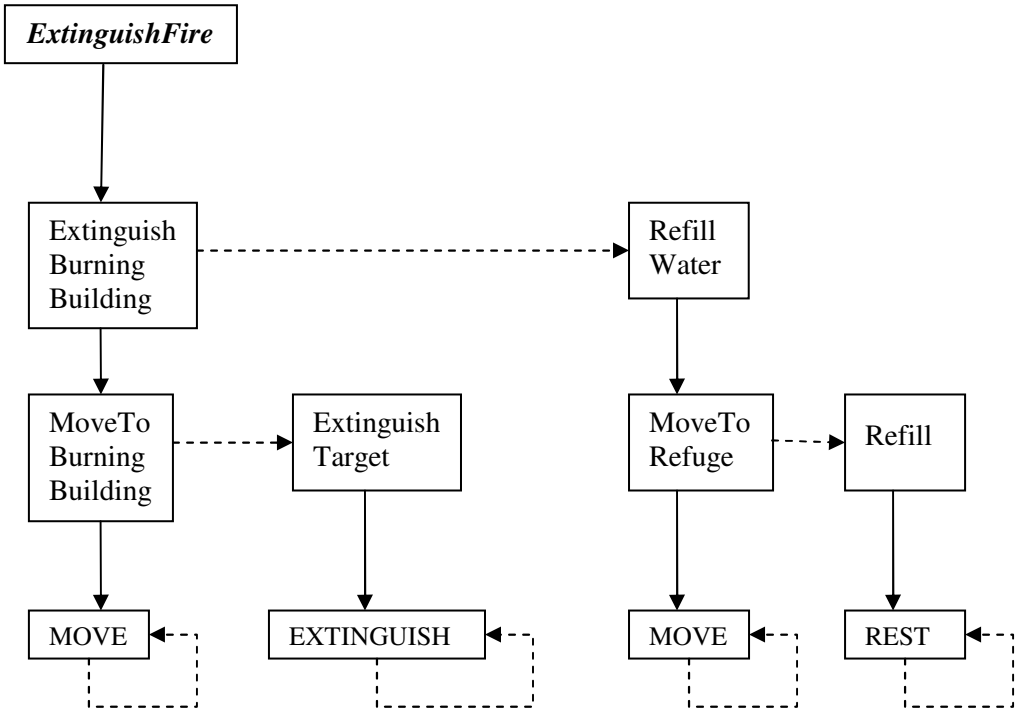
Anhang B: Planbibliothek

Der Aufbau der Planbibliothek wird zunächst anhand der Pläne dargestellt ohne die einzelnen untergeordneten Planschritte aufzuführen. Anschließend erfolgt die Veranschaulichung der einzelnen Pläne samt zugehörigen Planschritten.

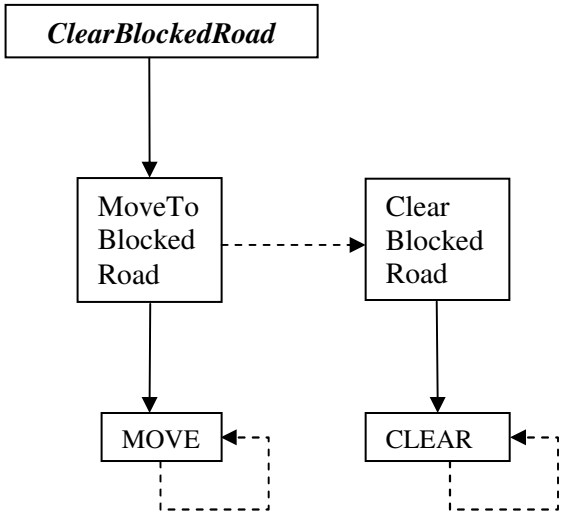
Aufbau der Planbibliothek:



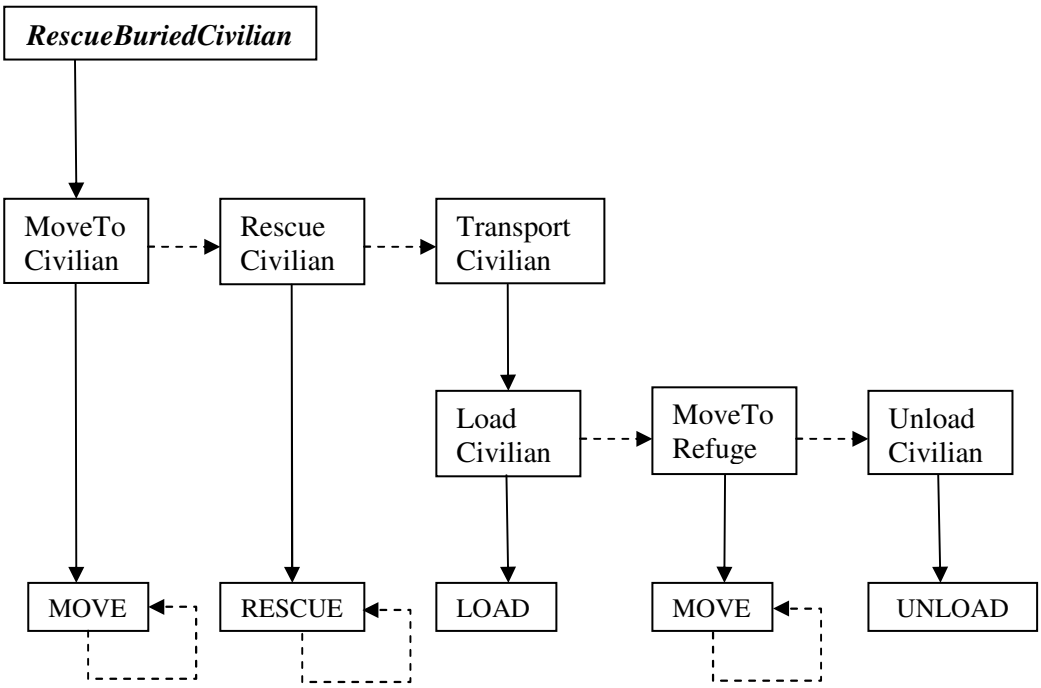
Plan *ExtinguishFire:*



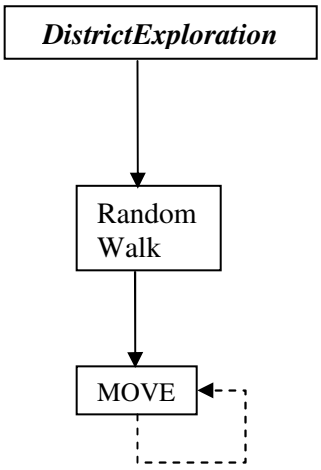
Plan *ClearBlockedRoad*:



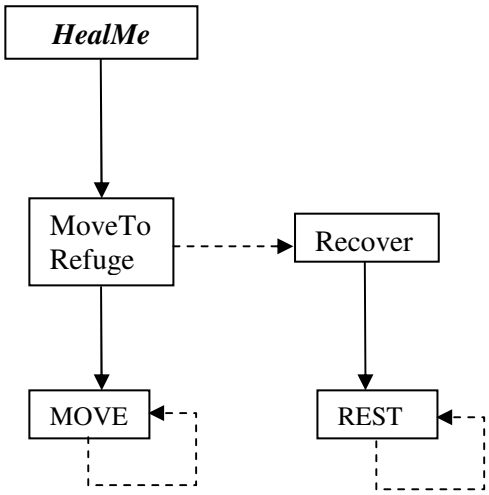
Plan *RescueBuriedCivilian*:



Plan *DistrictExploration*:



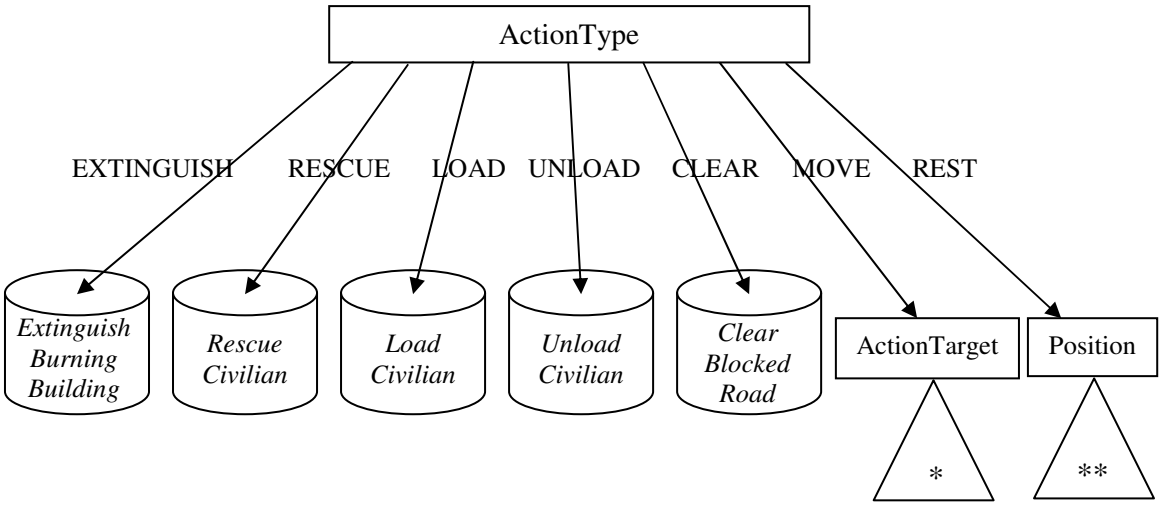
Plan *HealMe*:



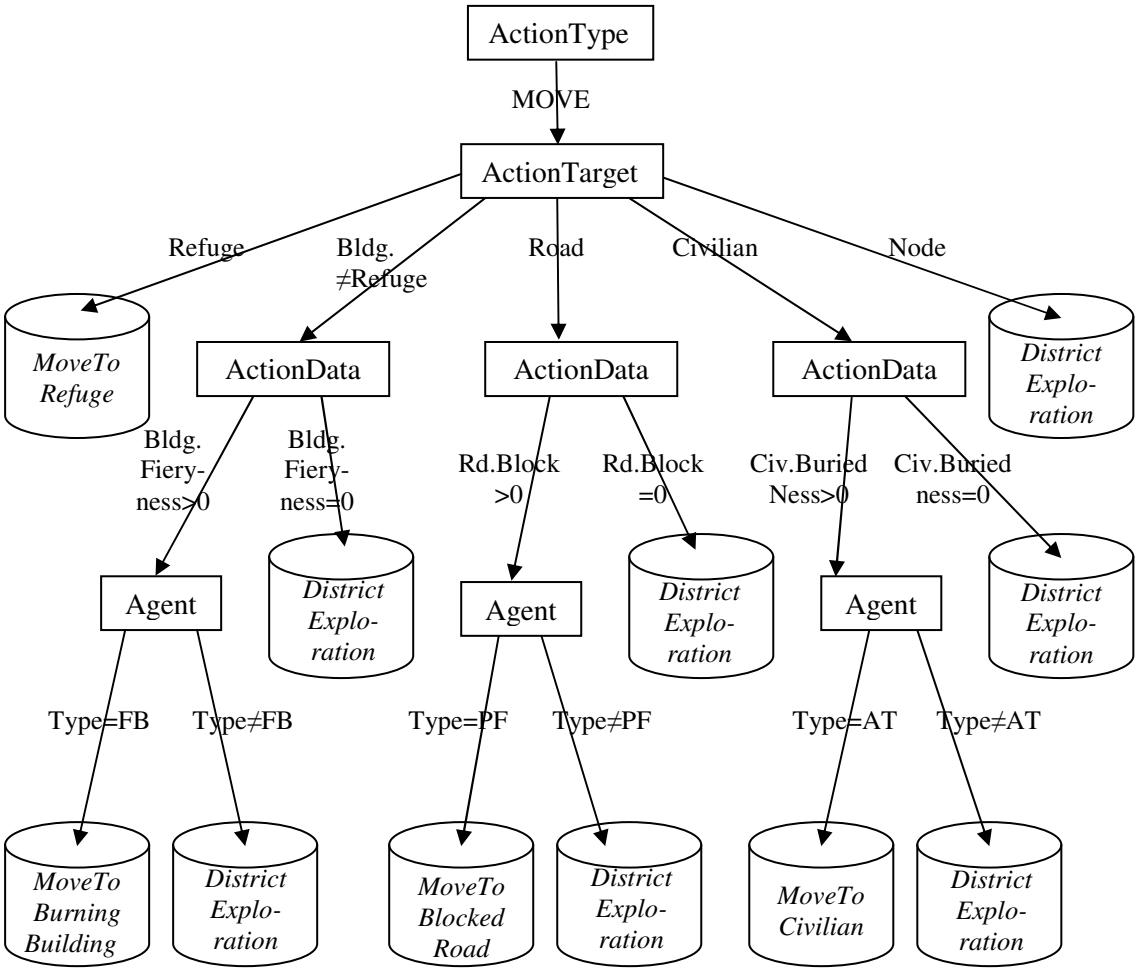
Anhang C: Feature Decision Tree

Die Darstellung des Feature Decision Tree erfolgt in drei Teilen. Zunächst wird der FDT nur bis zur Tiefe 1 abgebildet. Anschließend werden die beiden Fälle skizziert, in denen der FDT eine Baumtiefe größer 1 erreicht.

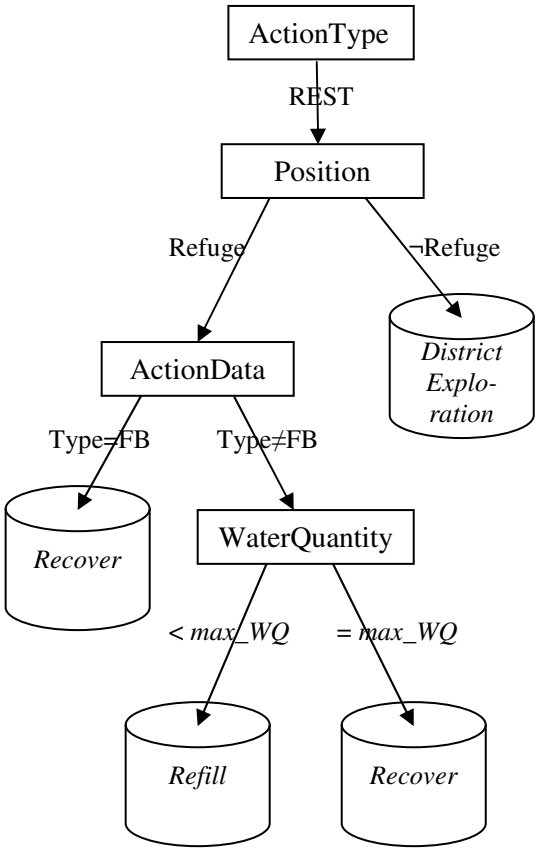
FDT bis zur Tiefe 1:



Ausschnitt des FDT für *ActionType = MOVE* (*):



Ausschnitt des FDT für *ActionType* = *REST* (**):



Anhang D: Verwendung des Rescueanalysers

In diesem Anhang wird die Verwendung des Rescue-Analysewerkzeugs skizziert. Die Beschreibung erfolgt mittels der README-Datei, die innerhalb des Rescueanalyser Softwarepakets in deutscher und englischer Sprache enthalten ist.

Auszug der README-Datei des Rescueanalysers:

Installationsanleitung
****

Systemvorraussetzungen:
==========

Unix oder Gnu/Linux System, Java Version 1.5 oder höher.

Installation:
=========

Um den Quellcode zu kompilieren führe die Makedatei aus:

./make

Um den Rescueanalyser zu starten kopiere die Logdateien rescue.log und action.log
ins Hauptverzeichnis des Rescueanalysers (dort, wo sich die Shellskripte befinden)
oder erstelle einen Link zu den Logdateien. Dann führe das Startskript aus:

./start.sh

Bemerkung:
==========

Nach jedem Durchlauf generiert der Rescueanalyser die Dateien Ausgabe.html und Plan.html
in dem Unterverzeichnis Ausgabe. Neben den Aktionen der Agenten und den erkannten Plänen
kann die Planbibliothek eingesehen werden, die zur Planerkennung genutzt wurde.

Die Datei config.txt speichert Informationen über den Zustand des Monitors.
Standardmäßig ist er ausgeschaltet. Durch ersetzen von

viewer=0 mit viewer=1

kann der Monitor des Rescueanalysers zugeschaltet werden.

Viel Spaß!!!

Literaturverzeichnis

[1] Robolog. Webpage(s). www.uni-koblenz.de/FB4/Institutes/IFI/AGKI/Research/Current/Robolog. Abruf: 27.03.2007

[2] The RoboCup Federation. RoboCup Official Site. Webpage(s). http://www.robocup.org. Abruf: 27.03.2007

[3] Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Fast and Complete Symbolic Plan Recognition. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05). 2005.

[4] Stuart Russell and Peter Norvig. Artificial Intelligence. A Modern Approach. Prentice Hall, Englewood Cliffs, NJ, 1995.

[5] Frieder Stolzenburg. Agenten-Systeme. Skriptum zur Vorlesung, Hochschule Harz, Wintersemester 2005/2006.

[6] Jürgen Dix. Künstliche Intelligenz. Skriptum zur Vorlesung, Universität Koblenz-Landau, 1997.

[7] Henry A. Kautz. A Formal Theory of Plan Recognition. Department of Computer Science. University of Rochester. Rochester. May 1987.

[8] Henry A. Kautz and James F. Allen. Generalized plan recognition. In American Association for Artificial Intelligence (AAAI press), pages 32–37.1986.

[9] M. Tambe and P.S. Rosenbloom. RESC. An approach to agent tracking in a real-time, dynamic environment. In IJCAI-95. August 1995.

[10] Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. Journal of Artificial Intelligence Research (JAIR), Ausgabe 12, Seiten 105-147. 2000.

[11] RoboCup 2005 Osaka Committee. RoboCup 2005 Osaka. What is RoboCup. Webpage(s). http://www.robocup2005.org/about/what.aspx. Abruf: 27.03.2007

[12] Official Page of RoboCup 2006. Webpage(s). http://www.robocup2006.org. Abruf: 27.03.2007

[13] Anja Tempelhoff. Lehrer-Online. RoboCup Junior 2006. Webpage(s). http://www.lehrer-online.de/url/robocup-junior. Abruf: 27.03.2007

[14] RoboCupRescue Technical Committee and Organising Committee. RoboCup Rescue. Webpage(s). http://www.robocuprescue.org. Abruf: 27.03.2007

[15] Takeshi Morimoto, edited by RoboCupRescue Technical Committee. How to Develop a RoboCupRescue Agent. November 10, 2002.

[16] RoboCup-Rescue Official Web Page. Webpage(s). http://www.rescuesystem.org/robocuprescue. Abruf: 27.03.2007

[17] Alexander Kleiner et al. ResQ Freiburg: Team Description and Evaluation. Universität Freiburg. 2004.

[18] Ansgar Bredenfeld, Hans-Dieter Burkhard, Martin Riedmiller, Raúl Rojas. c't 13/2006, S. 102: RoboCup. KI auf dem Fußballfeld. 2006.

[19] Jafar Habibi, Alireza Fathi, Saeed Hassanpour, Mohammadreza Ghodsi, Behzad Sadjadi, Hamidreza Vaezi, Majid Valipour. Impossibles Team Description. Sharif University of Technology, Computer Engineering, Iran, Tehran. Februar 22, 2005.

[20] The RoboCup Rescue Technical Committee. RoboCup-Rescue Simulator Manual. Version 0 revision 4. Juli 1, 2000.

[21] Michael Brenner, Nanda Wijermans, Timo Nüssle, and Bart de Boer. Simulating and Controlling Civilian Crowds in Robocup Rescue.

[22] Stefan Fricke. Grundlagen der Künstlichen Intelligenz. Technische Universität Berlin. 20.10.2005.

[23] BBS Winsen/Luhe. GoBlack. Mikrocontroller. Entstehung eines Maschinenprogramms. 10.03.2000. Webpage(s). http://www.goblack.de/desy/mc8051chip/theorie/pgmentwt.html. Abruf: 27.03.2007

[24] Takeshi Morimoto. RoboCup Rescue Viewer. 2002. Webpage(s). http://ne.cs.uec.ac.jp/~morimoto/rescue/viewer/index.html. Abruf: 27.03.2007

[25] Seyed Hamid Hamraz, Seyed Shams Feyzabadi, Amir Khayati Motlagh. Caspian RoboCup Rescue Simulation Agent Competition Team Description. Computer Engineering Department, Iran University of Science and Technology, Tehran, Iran.

[26] Fraunhofer Institut für Autonome Intelligente Systeme. RoboCup German Open 2005. Webpage(s). http://www.ais.fraunhofer.de. Abruf: 27.03.2007

[27] Alexander Kleiner. Rescue3D. 2004. Webpage(s). http://kaspar.informatik.uni-freiburg.de/~rescue3D/index.html. Abruf: 27.03.2007

[28] Kaminka, G. A. and Avrahami, D. Symbolic Behavior Recognition. In Proceedings of the AAMAS 2004 Workshop on Modeling Other agents from Observations (MOO 2004). 2004.

[29] Tom M. Mitchell. Machine Learning. McGraw-Hill, 1997.

[30] Alexander Kleiner. RoboCupRescue Simulation league 2005. 14. December 2004. Webpage(s). http://kaspar.informatik.uni-freiburg.de/~rcr2005. Abruf: 27.03.2007