Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniela Niemeyer

Goal-Oriented Action Planning für die Simulationsplattform MARS

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniela Niemeyer

Goal-Oriented Action Planning für die Simulationsplattform MARS

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas Thiel-Clemen
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 19.12.2014

Daniela Niemeyer

Thema der Bachelorarbeit
Goal-Oriented Action Planning für die Simulationsplattform MARS

Stichworte
Dynamisches Planen; GOAP; Multiagentensimulation; SimPan;

Kurzzusammenfassung
Die Arbeit beschreibt die Implementation von Goal-Oriented Action Planning für die Multiagentensimulationsplattform MARS. Ein besonderer Fokus dabei ist, dass sich die Komponenten sauber in die bestehende Architektur einfügen. Für nachfolgende Tests wird ein Szenario im Sinne des Referenzmodells SimPan erstellt. Anschließend wird untersucht wie sich dieses Modell sinnvoll umstrukturieren lässt um es mit GOAP nutzen zu können.

Daniela Niemeyer

Title of the paper
Goal-Oriented Action Planning for the simulation platform MARS

Keywords
Dynamic planning; GOAP; Multi-Agent Simulation; SimPan;

Abstract
This work is about the implementation of Goal-Oriented Action Planning for the multi-agent simulation platform MARS. A particular focus is on integrating the components neatly into the existing architecture. For successive tests a scenario in terms of the reference model Simpan is created. Finally it is examined how this model can be restructured to make sense in use with GOAP.

Inhaltsverzeichnis

1 Einleitung 9

1.1 Motivation 9

1.2 Zielsetzung 10

1.3 Abgrenzung 11

1.4 Gliederung der Arbeit 11

2 Grundlagen 12

2.1 Multiagentensimulation 12

2.2 MARS 12

2.3 Goal-Oriented Action Planning (GOAP) 13

2.3.1 Goals 13

2.3.2 Worldstate 14

2.3.3 Actions 14

2.4 Planen und GOAP 15

2.5 SimPan 17

3 Analyse 18

3.1 Die Architektur von MARS 18

3.2 Die Architektur von GOAP 19

3.2.1 „Perception and Memory“ 20

3.2.2 „Action Selection“ 21

3.2.3 „Motor and Navigation“ 21

3.3 Graphen und A-Stern 21

3.4 Modell und Szenario-Entwicklung 22
3.4.1 Die Umwelt in SimPan 22
3.4.2 Menschliche Agenten 23
3.5 Anforderungen 25
3.5.1 Integration der GOAP-Komponente in das MARS-System 25
3.5.2 Nutzbarkeit für Modellentwickler 26
3.5.3 GOAP Performance 26
3.5.4 Umsetzung eines Szenario nach SimPan 27
4 Design 28
4.1 GOAP 28
4.1.1 Architektur 28
4.1.2 Komponente Goap-Action-System 29
4.1.3 Schnittstellenpaket Goap-Common 30
4.1.4 Komponente Goap-Graph-Service 30
4.1.5 Effect-Action Map 30
4.1.6 Heuristik 31
4.2 Details zur Inferenzbildung 31
4.2.1 Fehlerquellen in der Inferenzbildung 33
4.2.2 Resultierende Regeln und Abläufe für die Knotenerstellung 38
4.2.3 Regeln um Inkonsistenzen beim Erstellen von GOAP-Nodes zu vermeiden . 38
4.3 Erste Schritte für Modellentwickler 39
4.4 Entwicklung des GOAP Testszenarios 40
4.4.1 Die Umwelt 40
4.4.2 Die Menschen 41
4.4.3 Resultierende Layer in MARS 44
4.4.4 Szenario Daten 45
4.4.5 Visualisierung 45
5 Realisierung 47
5.1 GOAP 47
5.1.1 Schnittstellen in GOAP-Common 47
5.1.2 GOAP-Action-System 49
5.1.3 Anfrage der nächsten Action 51

5.2 Blackboard 52
5.3 Goap-Graph-Service 52
5.3.1 Libraries für Graphen und zugehörige Algorithmen 52
5.3.2 Goap-Simple-Graph-Service 53
5.4 GOAP Elemente der Simulation 53
5.4.1 Goals 53
5.4.2 Actions reaktiver Agent 54
5.4.3 Actions deliberativer Agent 54
5.4.4 Actions reflektiver Agent 55
5.4.5 Gemeinsame Actions 55
5.4.6 Symbole 55
5.5 Implementierung der Simulation mit GOAP 56
5.5.1 Konfiguration von GOAP 58
5.5.2 Design der Actions 59
5.5.3 Visualisierung der Simulation 59
6 Tests und Ergebnisse 61
6.1 GOAP 61
6.1.1 Performance Tests 61
6.2 GOAP-Komponente im MARS Kontext 64
6.3 Die Simulation anhand von SimPan 64
7 Diskussion und Ausblick 67
7.1 Rückblick 67
7.2 Zusammenfassung 67
7.3 Ausblick 68
8 Literaturverzeichnis 69

Abbildungsverzeichnis

Abbildung 1 "The plan formulation process" (Orkin, 2003) 15
Abbildung 2 "The planner's regressive search" (Orkin, 2003) 16
Abbildung 3 „General system overview“ (Hüning, Wilmans, Feyerabend, & Thiel-Clemen, 2014) 18
Abbildung 4 „Runtime Deployment view of LIFE“ (Hüning et al., 2014) 19
Abbildung 5 „The C4 brain architekture“ (Burke et al., 2001) 20
Abbildung 6 „Zusammenfassung der unterschiedlichen Zelltypen“ (Schneider, 2011) 22
Abbildung 7 „Allgemeine Eigenschaften von Agenten“ (Schneider, 2011) 24
Abbildung 8 „Die Stufen der modellierten menschlichen Verhaltenskontrolle“ (Schneider, 2011) 24
Abbildung 9 „Elemente des mentalen Weltbilds eines SimPan-Agenten“ (Schneider, 2011)25
Abbildung 10 Blackbox GOAP Komponenten 28
Abbildung 11 Agentendefinition und GOAP Komponenten 29
Abbildung 12 „Zustandsvariable und Motiv“ (Schneider, 2011) 41
Abbildung 13 „Berücksichtigte Verhaltensweisen im Überblick“ (Schneider, 2011) 42
Abbildung 14 Darstellung der Abhängigkeiten unter den Layern 45
Abbildung 15 „Visualisierung des initialen Zustands des Prüfszenars“ (Schneider, 2011) 46
Abbildung 16 Initialisierung der GOAP-Komponente 49
Abbildung 17 Klassen im Goap-Action-System 50
Abbildung 18 Ablaufdiagramm: Anfrage der nächsten Action. 51
Abbildung 19 Projektmappenstruktur der Simulation 56
Abbildung 20 Abhängigkeiten der Projekte der Simulation 57
Abbildung 21 Die Konfigurationsklasse des reaktiven Agenten 58
Abbildung 22 Action für die schrittweise Annäherung 59
Abbildung 23 Darstellung der Simulation zu Beginn 60
Abbildung 24 Die möglichen Verzweigungen. 63
Abbildung 25 Der durch GOAP optimierte Graph. 63
Abbildung 26 Das kritische Ereignis hat einen Agenten getötet 65
Abbildung 27 Reflektiver Agent mit Beruhigungs- und Massenfluchtsphäre 66

Abbildung 28 Massenfluchtsphäre bei diagonaler Bewegung..... 66
Abbildung 29 Massenfluchtsphäre bei vertikaler Bewegung 66
Abbildung 30 Reaktiver Agent nimmt an der Massenflucht teil..... 66
Abbildung 31 Ausgang und Ausgangsbereich 66
Abbildung 32 Technische Beruhigungsquelle 66

1 Einleitung

In Multiagentensimulationen werden Agenten in der Regel durch endliche Automaten, Entscheidungsbäume oder Verhaltensbäume (Colledanchise & Ögren, 2014) in ihrem Verhalten/ ihren Reaktionen gesteuert. Dies macht die Modellierung nicht unbedingt übersichtlich, führt teilweise zu starrem Verhalten und bietet Möglichkeiten für Fehler bei der Gestaltung der Zustandsübergänge, wie z.B. nicht erreichbaren Zuständen (Orkin, 2003).

Menschen können flexibel planen und diese Pläne, im Rahmen ihrer unterschiedlichen Fähigkeiten, an nicht vorhersehbare Ereignisse anpassen. Für die Simulation von Menschen scheint der Ansatz des dynamischen Planens näher an der Realität. Dazu gehört auch, dass Agenten mehr als ein Ziel verfolgen können. Der Agent kann zwischen verschiedenen Zielen wechseln. Dies kann als Reaktion auf seine Umwelt passieren oder auch durch interne Ereignisse ausgelöst werden. Wie in der „Maslowschen Bedürfnishierarchie“[1] beschrieben kann z.B. in GOAP das Bedürfnis der Sicherheit als hoch priorisiertes Ziel modelliert werden.

Ein Ziel muss insbesondere nicht immer auf dem gleichen Weg erreicht werden. Der Plan und die Erfüllung des Ziels wird dynamisch erzeugt und kann somit angepasst an die aktuellen Gegebenheiten erfolgen.

Viele Modelle beschreiben Agenten und die damit verbundenen Verhaltensweisen im Sinne der oben aufgezählten Mechanismen. Inwieweit lassen sich diese Beschreibungen sinnvoll in GOAP überführen?

1.1 Motivation

Agentensimulationen werden immer wichtiger um Situationen in der Natur, ökologischen Systemen oder auch dem Alltag von Menschen bewerten zu können und

[1] http://de.wikipedia.org/wiki/Maslowsche_Bed%C3%BCrfnishierarchie

Entscheidungshilfen zu bieten. Viele Systeme werden heute detaillierter verstanden, wodurch die Komplexität ansteigt.

Die Besonderheit an GOAP ist die Verfolgung von Zielen und das dynamische Planungsverhalten. GOAP Systeme bieten eine leichte Erweiterbarkeit von Agenten im Simulationskontext. Aus einem bereits erstellten Aktionsraum lassen sich schnell neue Agenten mit Teilmengen zusammenstellen.

Es entsteht der Eindruck, dass die Modelle zuvor aus den Ansprüchen von GOAP heraus gebildet wurden. Was ist aber mit bereits bestehenden Modellen? Können diese portiert werden? Müssen sie dazu stark verändert werden, oder lässt sich dies mit kleinen Veränderungen und Erweiterungen bewerkstelligen?

1.2 Zielsetzung

Für die Multiagentensimulationsplattform MARS, die im Rahmen eines Projektes an der Hochschule für angewandte Wissenschaften Hamburg entwickelt wurde, soll eine GOAP-Komponente bereitgestellt werden. Die Nutzer sollen dann entsprechend dem Modell Ziele, Aktionen und Zustände definieren und in den Agenten einbinden können.

Im besonderen Fokus ist die Stabilität und Wartbarkeit des Gesamtsystems MARS weiterhin auf hohem Niveau zu halten. Die bisherige Architektur ist orientiert an Quasar (Siedersleben, 2004).

Wichtig ist ebenfalls die Performance eines Planungssystems. Simulationen sollen teils Situationen anhand der aktuellen Lage hervorsagen können und müssen somit ausreichend schneller Ergebnisse liefern als das in der Realität zu beobachtende Phänomen. Zudem sind häufig auch mehrere Läufe sinnvoll um Extremwerte zu erkennen und zu entfernen. GOAP wird bisher in der Computerspieleindustrie genutzt und kann dort effizient Echtzeitanforderungen der Spiele befriedigen (Orkin, 2005).

Weiterhin ist ein Modell und daraus folgend ein konkretes Szenario mit zugehöriger Programmierung zur Überprüfung der entwickelten Komponente erforderlich. Als Grundkonstrukt dient das Referenzmodells SimPan (Schneider, 2011), welches sich mit dem menschlichen Verhalten vor allem in Ausnahmesituationen beschäftigt.

Da das Referenzmodell SimPan nicht auf die Strukturen von GOAP ausgelegt ist, werden auch die Überführung und daraus entstehende Konsequenzen betrachtet. Dabei ist interessant, worin die Schwierigkeiten liegen, wenn ein Modell auf ein GOAP System mindestens teilweise übertragen wird.

1.3 Abgrenzung

Es ist nicht Teil dieser Arbeit das gesamte Referenzmodell SimPan in ein GOAP System zu überführen. Dieses Vorhaben würde absolut den Rahmen einer Bachelorarbeit sprengen, weil das Referenzmodell SimPan sehr umfangreich und hochgradig detailliert ist. Die Implementierung eines Modells auf Basis von SimPan dient Tests des entwickelten GOAP Systems.

1.4 Gliederung der Arbeit

Anfangs erfolgt eine Einführung in die wichtigsten Begriffe um eine klare Basis für den Rest der Arbeit zu legen. Nachfolgend werden die Teile eines GOAP Systems detailliert abgehandelt, um daraus klare Anforderungen und Teilaufgaben zu identifizieren.

Im Kapitel Design werden Konzepte zu den entstandenen Aufgabenbereichen entwickelt und validiert. Im nächsten Abschnitt folgt die Beschreibung des Entwicklungsverlaufs inklusive Dokumentation von Design- und Implementierungsentscheidungen. Anschließend wird die Entwicklung des Szenarios anhand von SimPan erläutert, welche als Test für die GOAP-Komponente fungieren soll.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe und Konzepte für diese Bachelorarbeit vorgestellt. Es soll als Basis für die nachfolgenden Kapitel und als Einstieg in den Themenbereich dienen.

2.1 Multiagentensimulation

> *„Multiagentensimulation beruht auf der Formulierung des Modells als Multiagentensystem, d.h. als System, das aus interagierenden, autonomen Agenten besteht, in einer Umwelt, die ebenfalls Bestandteil des Modells ist. Das Spektrum agentenbasierter Modelle reicht dabei von Modellen, die nur auf einem abstrakten Konzept eines „Agenten beruhen, bis zu Modellen, die menschliche Akteure als intelligente Agenten nachbilden.“* (Klügl, 2006)

2.2 MARS

Das MARS System[2] ist eine Multiagentensimulationsplattform, die an der Hochschule für Angewandte Wissenschaften entwickelt wird. Ein großes Team aus Master-, Bachelorstudenten und Doktoranden erarbeitet kontinuierlich neue Konzepte und Komponenten.

Besonders an MARS ist der verteilte Simulationsansatz. Dadurch können ganz neue Dimensionen der Simulation erreicht werden um auch komplexe Szenarien mit hunderttausenden Agenten durchzuführen. Auch sehr große Szenarien, wie sie bei der Loveparade 2010 in Duisburg[3] nötig gewesen wären, sollen mit MARS durchgeführt werden können.

Weiteres Merkmal von MARS ist die Aufteilung der Aspekte einer Simulation in verschiedene Layer. Ein Beispiel eines Aspekts ist die statische Raumstruktur eines Gebäudes. Ein anderer Aspekt könnte die Menge von reaktiven Objekten sein, die durch Agenten genutzt werden können.

---

[2] http://mars-group.org/

[3] http://de.wikipedia.org/wiki/Ungl%C3%BCck_bei_der_Loveparade_2010

In MARS sind Agenten immer zu einem Layer zugehörig. Verschiedene Typen von Agenten lassen sich durch verschiedene Layer gruppieren. Der Modellentwickler besitzt einen hohen Freiheitsgrad (Hüning, 2013) bei der Implementierung eines Modells.

Services in MARS sind z.B. der Import und die Verschneidung von Geodaten (MARS GROUND), Sammlung und Kombination von Simulationsereignissen (MARS SURVEYOR). MARS bietet mehr und mehr Services, die Modellentwickler und Modellprogrammierer bei der Umsetzung unterstützen. Es werden kontinuierlich neue Ideen umgesetzt und bereitgestellt.

2.3 Goal-Oriented Action Planning (GOAP)

GOAP dient der dynamischen Planung von Aktionen. Es basiert auf einer vereinfachten STRIPS (**St**anford **R**esearch **I**nstitute **P**roblem **S**olver) (Fikes & Nilsson, 1971) ähnlichen Struktur.

Der Ursprung liegt in der Computerspieleindustrie. GOAP wurde für ein glaubwürdigeres Verhalten der Non-Player-Character (NPC), Wiederverwendbarkeit der Fähigkeiten von NPCs und einen erleichterten Arbeitsprozess zwischen Gamedesignern und Programmierern (“The 2004 AIISC Report - Working Group on Goal-oriented Action Planning,” 2004) entwickelt. Ein hierzu nennenswerter Vertreter aus der Spieleindustrie ist z.B. F.E.A.R.™ [4] von MONOLITH™ [5] aus dem Jahr 2005.

GOAP wurde bereits prototypisch im Kontext von Multiagentensimulationen (Klingenberg, 2010) untersucht. Auch im Bereich der Robotik wurde durch Felix Kolbe schon eine zielorientierte Aufgabenplanung (Kolbe, 2013) für einen Service-Roboter umgesetzt.

GOAP beinhaltet drei Hauptelemente, die hier detaillierter in ihrer Funktion und im Zusammenspiel erläutert werden.

2.3.1 Goals

Goals befähigen einen Agenten zum zielorientierten Verhalten. Sie sind der Ausgangspunkt für das Handeln eines Agenten.

Goals beinhalten eine Beschreibung der Welt in Form einer Teilmenge der verfügbaren Einzelzustände. Ob ein Goal als erfüllt angesehen werden kann, wird anhand eines Vergleiches des Ist-Zustandes mit dem Soll-Zustandes überprüft.

[4] http://www.lith.com/Games/F-E-A-R-
[5] http://www.lith.com/Games/

Wie und nach welchen Kriterien ein Agent seine aktiven Goals wählt, ist bei GOAP größtenteils durch den Modellierer entscheidbar. Es müssen lediglich Prüfungen auf Erreichbarkeit des Ziels durchgeführt werden.

Goals besitzen eine Priorisierung. Dies macht z.B. bei Szenarien, die eine Gefährdung des Agenten beinhalten, Sinn. Konkreter kann modelliert werden, dass ein Ziel der Genussbefriedigung eine geringere Priorität hat, als das Ziel der Selbsterhaltung. Diese Priorität kann auch abhängig von dem aktuellen Weltzustand sein. Ein Agent der sich gerade in keiner Bedrohungssituation befindet, entscheidet sich hier logischerweise für das Goal der Genussbefriedigung.

2.3.2 Worldstate

> *„We represent the state of the world with a data structure that consists of a fixed-size array of symbols, implemented as key-value pairs. Keys are represented by enumerated world properties. Values are a union of possible data types. Each NPC maintains its own symbolic view of the world through a world state member variable. The world state includes symbols for properties such as the NPC's position, weapon, amount of ammo, target object, and health of target object."* (Orkin, 2004)

Die Kombination dieser Symbole ergibt für jeden Agenten seinen Worldstate in Form eines individuellen Weltbildes. Dabei können diese nicht nur die Umgebung, also äußere Eigenschaften darstellen, sondern auch agenteneigene interne Merkmale veranschaulichen.

Da GOAP nun aber im Kontext der Multiagentensimulation angewendet wird, repräsentieren Symbole seltener, wie oben genannt, den Waffenstatus oder die Menge der Munition. Zum Beispiel könnte modelliert werden, ob dem Agenten ein Fluchtweg bekannt ist oder, ob dieser geöffnet ist. Eine Abbildung agenteninterner Merkmale könnte sich auf den Hunger des Agenten beziehen.

2.3.3 Actions

Ähnlich wie in STRIPS (Fikes & Nilsson, 1971) ist eine Action definiert über ihre einzigartige Bezeichnung, die Vorbedingungen unter denen sie ausgeführt werden kann und die Änderungen am Weltzustand (in GOAP als Effekte bezeichnet) zu dem sie führt.

Vorbedingungen und Effekte bestehen aus einer Teilmenge der Symbole des Modells. Da einige Teile eines Worldstate aber nicht optimal in Werte abgebildet werden können, gibt es zusätzliche Eigenschaften einer Action. Orkin bezeichnet diese als „Kontext Vorbedingungen" und „Kontext Effekte". Ein Beispiel wäre, ob ein Objekt auf dem eine

Interaktion ausgeführt werden soll, auch erreichbar ist (Orkin, 2003). Hier kann beliebiger Code untergebracht werden.

2.4 Planen und GOAP

Planen ist das Erstellen einer Folge von Aktionen, die von einem aktuellen Zustand in einen Zielzustand überführen (Beierle & Kern-Isberner, 2008).

Um dieses noch weiter zu differenzieren, nutzen Alexandra Kirsch und Josef Schneeberger eine Einteilung in „Handlungsplanung“ (auch als klassische Planung bezeichnet) und „Situierte Aktivität“. Sie nennen zwei Beispiele um die Unterschiede zu verdeutlichen:

Beispiel 1:
Wenn das Ziel die Teilnahme an einem Kongress ist, ergeben sich daraus viele Teilziele. Erwerb einer Bahnkarte, Hotelreservierung, Registrierung bei der Konferenz und viele mehr. Es handelt sich um eine komplexe Folge von Aktionen, bei der auch die Reihenfolge wichtig ist. Insgesamt wird dies unter „Handlungsplanung“ verstanden.

Beispiel 2:
Wenn es das Ziel ist, auf einem überfüllten Bahnsteig den Zug zu betreten, gilt es vorrangig Anderen auf dem Bahnsteig auszuweichen und Platz zu machen, damit der Zug sich leeren kann. Zuletzt natürlich ist auch von Relevanz den Zug vor dessen Abfahrt zu erreichen und zu betreten. Dies wird als „Situierte Aktivität“ eingestuft. (Görz, Schneeberger, & Schmid, 2014).

GOAP soll in diesem Sinne der Handlungsplanung dienen, da ansonsten die Kosten der Planung nicht mehr zweckdienlich sind. GOAP zeichnet sich durch flexible Planung aus. Dem Agenten wird eine Menge von Zielen und Aktionsmöglichkeiten zur Verfügung gestellt wobei diese Elemente keine Abhängigkeiten voneinander besitzen. Der Agent entscheidet sich für eines der verfügbaren Ziele und versucht einen Weg vom aktuellen Zustand in den Zielzustand zu finden.

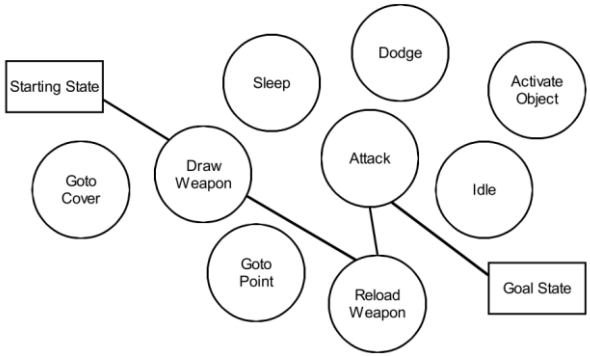


Abbildung 1 "The plan formulation process" (Orkin, 2003)

Es können sich natürlich auch mehrere Wege innerhalb eines Aktionsraumes zwischen Start und Ziel befinden, wobei es aber ausreichend ist, wenn einer gefunden wird, der zum gewünschten Ziel führt.

Es ergeben sich verschiedene Vorteile aufgrund dieser dynamischen Suche und unabhängiger Aktionen in GOAP:

- Da Pläne nicht statisch formuliert und vorgegeben sind, kann der Agent zur Laufzeit einen, an die aktuelle Situation angepassten, Plan entwickeln. Dadurch ergibt sich bereits eine hohe Flexibilität.
- Mögliche Aktionen eines Agenten werden immer nur in kleinen unabhängigen Einheiten formuliert. Es muss kein großer und schwer änderbarer endlicher Automat formuliert werden.
- Unterschiedliche Agenten lassen sich anhand verschiedener Zusammenstellungen von Aktionen und Zielen für ein Szenario erstellen.

Aktionen in GOAP besitzen Vorbedingungen und Effekte. Diese werden anhand von Wertbelegungen einzelner Weltzustände in der Aktion festgelegt. Diese Vorbedingungen und Effekte sind die Grundlage für das Auffinden von passenden Aktionsketten.

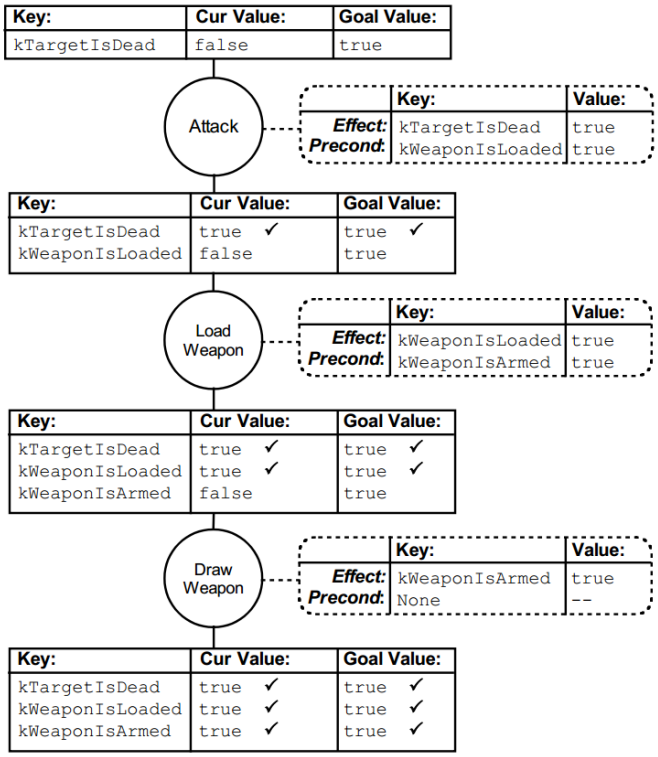


Abbildung 2 "The planner's regressive search" (Orkin, 2003)

In GOAP wird ein Plan, wie im obigen Titel schon erwähnt, vom Ziel aus rückwärts erstellt. Das bedeutet, dass nur der Ausschnitt der relevanten Symbole berücksichtigt werden muss, also dass weniger Daten verarbeitet werden müssen um einen Plan zu finden.

2.5 SimPan

Bernhard Schneider entwickelte in seinem Buch „Die Simulation des menschlichen Panikverhaltens“ (Schneider, 2011) ein Referenzmodell zur agentenbasierten Modellierung menschlichen Verhaltens in Paniksituationen. SimPan bietet eine fundierte Grundlage um Szenarien mitsamt der bestehenden Anhängigkeiten und Auswirkungen auf das Verhalten eines Menschen in Paniksituationen zu simulieren. Es beinhaltet physische, emotionale, kognitive und soziale Aspekte eines Agenten und stellt somit eine komplexe und auch fundierte Basis für die Entwicklung von Szenarien zur Verfügung.

Für die Entwicklung des Referenzmodells wurden Großschadensereignisse analysiert, die von verschiedenen Behörden dokumentiert und archiviert wurden. Darunter ist ein Ereignis in Brüssel am 29. Mai 1985 als bei der Final-Ausspielung des Europapokals durch gegenseitige Provokationen und das Erstürmen von Blöcken im Stadion, Panik ausbrach. Dies forderte 39 Tode durch Erdrücken, zu Tode trampeln und wenige sogar durch Wurfgeschosse. Zusätzlich wurden 454 Menschen verletzt.

Als ein weiteres, sich wiederholendes Großschadensereignis, wird die islamische Pilgerfahrt nach Mekka genannt. Durch die Ballung der Menschen und entstandener Panik resultierten Katastrophen mit variierenden Opferzahlen zwischen zwei im Jahre 1990 und 1425 im Jahre 2000.

SimPan bietet eine ganzheitlichere Sicht auf menschliche Verhalten inklusive der Abbildung von Unzulänglichkeiten im Wahrnehmen, Denken und Handeln eines Menschen. Somit werden komplexere Agenten-basierte Modellierungen möglich, die das Individuum deutlicher berücksichtigen.

3 Analyse

Nachfolgend wird betrachtet, wie in einem so allgemeinen Kontext wie der Multiagentensimulation, die GOAP Funktionalität für Modellprogrammierer möglichst optimal angeboten werden kann.

Anhand der bestehenden MARS Architektur und den bisherigen Funktionen des MARS-Systems wird in diesem Kapitel untersucht, wie sich die GOAP-Komponente in MARS integrieren lassen kann.

Um das Modell SimPan als Testsimulation in GOAP umzusetzen, werden die Eigenschaften der Agenten und der zugehörigen Umwelt untersucht.

3.1 Die Architektur von MARS

Um die neue GOAP-Komponente einpassen zu können und das Design des Gesamtsystems nicht zu beeinträchtigen, muss die Grundstruktur und Aufteilung des Systems betrachtet werden.

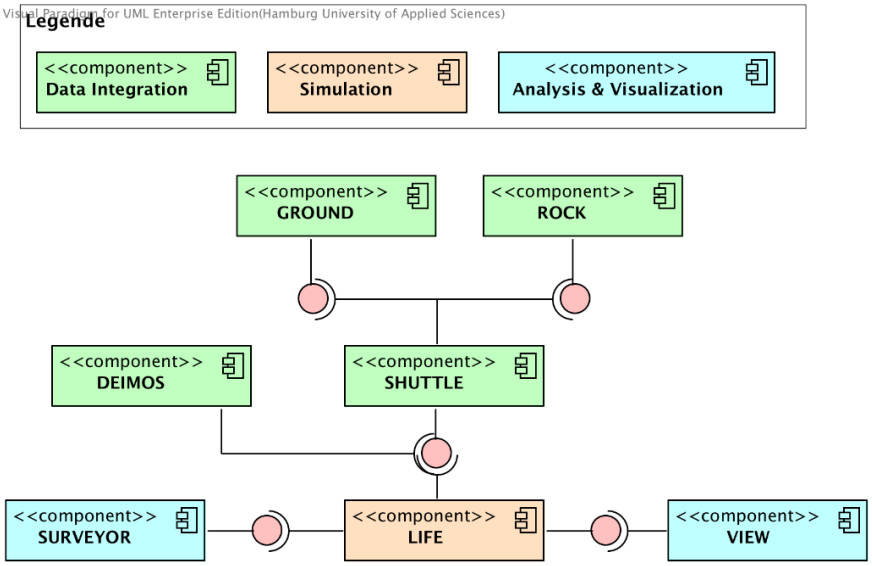


Abbildung 3 „General system overview“ (Hüning, Wilmans, Feyerabend, & Thiel-Clemen, 2014)

Die grünen Komponenten für die Datenintegration beinhalten z.B. GIS Formate[6], Geodaten[7] oder stellen Daten aufbereitet zur Verfügung. Die blauen Komponenten enthalten beispielsweise die Visualisierungskomponenten, die benötigt werden, um eine Simulation unter anderem optisch zu verfolgen und auswerten zu können. Der Kern der Simulationsberechnung liegt in der LIFE Komponente. Die LIFE-Architektur ist in drei Hauptkomponenten organisiert und beinhaltet den „Simulation Controller“ für Zugriff und Kontrolle des Systems, den „Simulation Manager“, der die Anweisungen des Controllers ausführt und die „Layer Container“ welche viele Layer und Agenten beinhalten können. (Hüning et al., 2014)

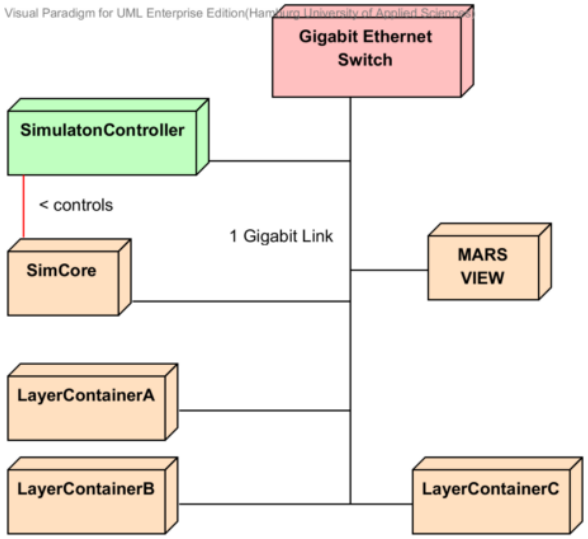


Abbildung 4 „Runtime Deployment view of LIFE“ (Hüning et al., 2014)

Hier werden auch z.B. verschiedene Layer-Typen für den Modellentwickler im Bereich LIFE-Services angeboten. Die Agenten der Simulation werden immer zu einem Layer zugehörig implementiert und gestartet. Die GOAP-Komponente wird im LIFE-Services Bereich angeboten.

3.2 Die Architektur von GOAP

Die C4-Architektur (Burke, Isla, Downie, Ivanov, & Blumberg, 2001) des MIT ist bereits erfolgreich von Jeff Orkin für seine GOAP Implementation in F.E.A.R.™ genutzt worden und bietet eine lose Kopplung der kommunizierenden Komponenten mittels eines Blackboards.

[6] http://de.wikipedia.org/wiki/GIS-Datenformat
[7] http://de.wikipedia.org/wiki/Geodaten

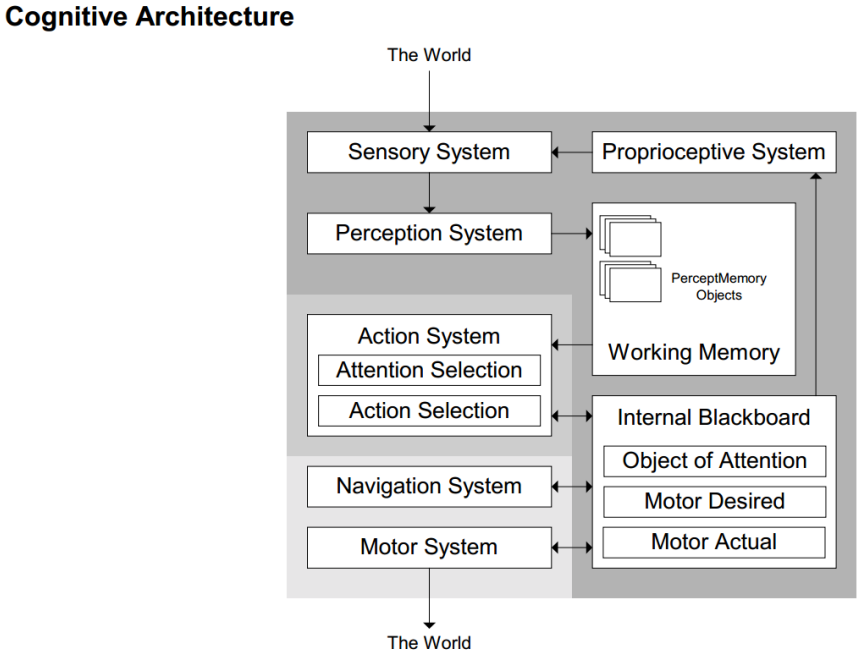


Abbildung 5 „The C4 brain architekture“ (Burke et al., 2001)

In der C4 Architektur werden die drei Hauptbestandteile „Perception and Memory“, „Action Selection“ und „Motor and Navigation“ genannt. Die Zusammenarbeit dieser Bestandteile erfolgt seriell. Zuerst ist „Perception and Memory“ aktiv, danach folgt „Action Selection“ und zuletzt immer „Motor and Navigation“. Diese Struktur ist grundlegend dem Sense-Reason-Act-Zyklus (Görz et al., 2014) eines Agenten sehr ähnlich und sollte sich somit auch gut für diese eignen.

GOAP bietet allerdings keine vollständige Definition eines Agenten. Es kann daher als Teil eines Agenten gesehen werden, der Entscheidungen bezüglich der nächsten Handlungen trifft. Die GOAP-Komponente ist somit der Kategorie „action selection“ oder dem Reason eines Agenten zuzuordnen.

3.2.1 „Perception and Memory“

Kernaufgabe ist die Aufnahme und Bereitstellung von Sensorinformationen. Hier befindet sich die Schnittstelle um Informationen aus der Umwelt abzurufen. In der C4 Architektur ist einen weitere Komponente vorgeschaltet, welche die Aufgabe hat, aus den eintreffenden Daten Erkenntnisse zu gewinnen und diese zu speichern.

Komponente Working Memory
Sammelt die von der Umwelt abgefragten Informationen und daraus aufbereiteten Information. Dies bedeutet noch nicht, dass auch alle Informationen in die Entscheidungsfindung eingehen.

Komponente Internal Blackboard
Dient der Kommunikation der internen Komponenten und verringert somit Abhängigkeiten zwischen diesen. Es können vorher festgelegte Informationstypen hinterlegt, wieder abgerufen und nutzende Komponenten nach dem „Publish Subscribe Pattern“ bei Änderungen informiert werden.

3.2.2 „Action Selection“
Hier werden die Erkenntnisse aus den Sensordaten verarbeitet und anhand deren Entscheidungen über die nächsten Actions des Agenten getroffen und im „Internal Blackboard“ gespeichert. In diesem Bereich ist die GOAP-Komponente anzusiedeln.

3.2.3 „Motor and Navigation“
An dieser Stelle befindet sich die ausführende Komponente. Die ausgewählten Aktionen, die im „Internal Blackboard“ hinterlegt wurden, können jetzt aktiviert werden. „Motor and Navigation“ stellt die zweite Schnittstelle zur Umwelt dar.

3.3 Graphen und A-Stern
Die GOAP-Komponente benötigt Zugang zu einer Library bzw. einer Komponente, die Graphen erstellen und Algorithmen zur Suche bereitstellen kann. Eine Anforderung aus GOAP ist die dynamische Erweiterung des Graphen im Wechsel mit dem nächsten Explorationsschritt des A*. Im Graphen beinhalten die Kanten die zugehörige GOAP-Action. Die Knoten halten die Zielsymbole und die aktuellen Symbole.

Der Graph kann nicht vollständig im Voraus konstruiert und mit den Werten der Heuristik versehen werden, weil sich eventuell ein unendlich großer Zustandsraum anhand der Actions aus GOAP ergibt. Aufgrund des theoretisch unendlich großen Graphen ist auch eine Beschränkung der Suchtiefe nötig.

Für GOAP werden gerichtete Graphen benötigt. Diese werden mit einem Quadrupel $G = (V, R, a, w)$ beschrieben. V bezeichnet die nicht leere Menge der Ecken/Knoten. R die Menge der Pfeile/Kanten. $a(r)$ bezeichnet die Anfangsecke und $w(r)$ die Endecke des Pfeils r. Bei GOAP können zusätzlich parallele und antiparallele Pfeile vorkommen. Zwei Pfeile $r, r' \in R, r \neq r'$ erfüllen für Parallelität $a(r) = a(r') \wedge w(r) = w(r')$. Antiparallele Pfeile müssen $a(r) = w(r') \wedge a(r') = w(r)$ (Krumke & Noltemeier, 2012) erfüllen.

Die Library bzw. Komponente soll austauschbar sein und somit über eine Stützschnittstelle angesprochen werden können. Damit die GOAP-Komponente korrekt arbeiten kann und die Komponente im MARS-System genutzt werden darf, müssen folgende Eigenschaften erfüllt sein.

1. Eine Action kann von mehreren Kanten referenziert werden.
2. Kanten können trotz gleicher referenzierter Action unterschiedliche Kosten haben.
3. Der A-Sternalgorithmus kann auf einem dynamisch erzeugten Graphen schrittweise angewandt werden.
4. Multigraphen mit Mehrfachkanten müssen möglich sein.
5. GOAP benötigt einen Digraph (vollständig gerichteter Graph).
6. Unter einer kostenlosen Lizenz verfügbar

3.4 Modell und Szenario-Entwicklung

Als Orientierung für das Szenario dient das Referenzmodell SimPan (Schneider, 2011). Menschliches Verhalten ist der Kernbestandteil von SimPan. Es bietet umfangreiche und detaillierte Beschreibungen der Elemente der Simulation. Es wird eine hohe Wahrscheinlichkeit angenommen, dass GOAP mit seinem dynamischen und zielverfolgenden Verhalten gute Ergebnisse liefern kann. Aufgrund der Komplexität von SimPan ist nur eine stark vereinfachte Variante im Rahmen einer Bachelorarbeit umsetzbar.

3.4.1 Die Umwelt in SimPan

Das Referenzmodell veranschaulicht die Umwelt mittels diskreter Zellen, die eine zweidimensionale Fläche aufspannen. Diese Zellen können sich in verschiedenen Zuständen (sogenannten Feldtypen) befinden.

Feldtyp	Beschreibung
Neutrale_Zelle (NZ)	Atomare Elemente eines begehbaren Areals, das von Agenten frei betreten, besetzt und verlassen werden kann.
Hinderniszelle (HZ)	Hindernisse halten einem gewissen Druck stand. Ist der, auf die Zelle wirkende Druck kleiner, ist die Zelle nicht durch Agenten betretbar.
Panikereignis_Zelle (PZ)	Ursprungszelle eines kritischen Ereignisses. Agenten, die sich bei Erscheinen einer Zelle dieses Typs auf diesem befinden, sterben.
Chaoszelle (CZ)	Chaoszellen modellieren den verwüsteten (Trümmer-) Bereich der durch ein kritisches Ereignis erzeugt werden kann. Ein Chaosfeld ist permanent nicht betretbar. Agenten, die sich bei Entstehung eines Chaosfeldes auf diesem befinden, sterben.
Verlustzelle (VZ)	Verlustzellen repräsentieren Areale, auf denen sich bewegungsunfähige Menschen befinden. Die Überquerung einer derartigen Zelle durch einen Agenten erfolgt mit reduzierter Geschwindigkeit und setzt einen bestimmten Druck auf den Agenten voraus.

Abbildung 6 „Zusammenfassung der unterschiedlichen Zelltypen“ (Schneider, 2011)

Es sind klar definierte Zustandsübergänge zwischen den Zelltypen vorgegeben. Neutrale Zellen können in alle anderen Zelltypen umgewandelt werden. Hinderniszellen können zu neutralen Zellen, Panikereignis Zellen oder Chaoszellen wechseln.

Diese Zellen dienen als einziges Informationsmedium für Agenten. Agenten können die Werte einer Zelle beim Betreten auslesen. Zellen stellen auch begehbare und nicht-begehbare Bereiche dar. Einfluss auf Agenten ist in Form des sofortigen Todes möglich.

Zudem existiert eine Wechselbeziehung zwischen Zellen und Agenten. Die Agenten können auf Hinderniszellen Druck ausüben, der ab einem Schwellenwert diese in Neutrale Zellen transformiert. Entgegengesetzt können Flüchtende durch Zellen geschädigt werden oder gar sterben.

Neben dem Feldtyp kann ein Zelle noch weitere Informationen halten. Dazu gehört z.B. ob die betreffende Zelle ein Ausgang, eine technische Informationsquelle (Lautsprecher oder Wegweiser) oder eine Beruhigungsquelle ist.

Bestimmte Informationen besitzen eine Ausdehnung. Das bedeutet konkret, dass Zellen Informationen von Nachbarzellen anbieten. Ein Fluchtwegschild oder eine beruhigende Sphäre eines Agenten ist z.B. in einem Radius von n Zellen für Agenten wahrnehmbar. Ein Ausgang ist, in einem bestimmten Umkreis, ebenfalls sichtbar.

3.4.2 Menschliche Agenten

SimPan berücksichtigt umfangreiche Details für die Darstellung des Menschen. Individuelle Ziele, interne Zustände, physiologische Zustände, Persönlichkeitseigenschaften usw. werden berücksichtigt. Bernhard Schneider fasst die elementaren Eigenschaften nach Maes (Maes, 1995) in der nachfolgenden Abbildung zusammen.

Eigenschaft		Beschreibung
/EA1/	Autonomie	Agenten sind adaptive Systeme mit Eigendynamik und individuellem Verhaltenspotenzial, die sie dazu befähigen, Ziele auch eigeninitiativ zu bestimmen und selbstständig zu verfolgen.
/EA2/	Individuelle Weltsicht	Agenten verfügen über ein internes Weltbild, das ihre Umwelt so repräsentiert, wie sie der Agent subjektiv wahrnimmt. Dies schließt Wissenslücken oder fehlerhafte Wahrnehmungen ein.
/EA3/	Intelligentes und rationales Verhalten	Agenten sind lernfähige Nutzenmaximierer, die versuchen, aus einer Menge möglicher Handlungsalternativen diejenige auszuwählen und umzusetzen, die den höchsten Nutzen im Hinblick auf das aktuelle Ziel verspricht.
/EA4/	Soziale Kompetenzen	Agenten können Gemeinziele berücksichtigen und sind Teil einer sozialen Struktur, die ihnen Kommunikations- und Kooperationsfähigkeiten abverlangen.
/EA5/	Mobilität	Agenten können sich in ihrer Umwelt bewegen und so die Reichweite ihrer Effektoren und Sensoren erweitern.

Abbildung 7 „Allgemeine Eigenschaften von Agenten“ (Schneider, 2011)

Verhaltensweisen der Agenten in SimPan

Die Agenten verhalten sich, je nach Belastung durch die Situation und eigener Konstitution, unterschiedlich. Kernpunkt ist die Flucht und wie sich Agenten dabei aufgrund von äußeren Einflüssen und eigenen Merkmalen verhalten. Das Motiv Angst ist einer der Hauptregler bei der Auswahl der Verhaltensstränge.

Die vier möglichen Zustände in denen sich ein Agent befinden kann, bestimmen den Verhaltensspielraum. Der aktuell verfügbare Bereich wird durch die Stärke der Angst und die persönlichen Schwellenwerte festgelegt.

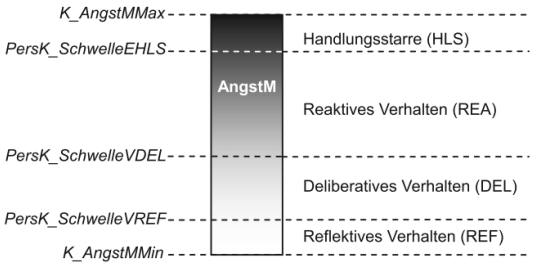


Abbildung 8 „Die Stufen der modellierten menschlichen Verhaltenskontrolle“ (Schneider, 2011)

Die Handlungsstarre:
Der Wert der Angst ist so hoch, dass der Agent sich nicht mehr bewegen kann. Er steht nur noch auf der Stelle. Intern schwächt sich seine Angst in kleinen Schritten ab.

Das reaktive Verhaltensspektrum:
Der Agent kann sich schrittweise an einen Ausgang annähern, wenn er in direkter Nähe ist. Weiterhin kann er anderen Agenten, die eine Massenflucht anführen, folgen. Wenn beide Möglichkeiten nicht gegeben sind, läuft er ziellos umher.

Das deliberative Verhaltensspektrum:
Hier kann der Agent auf die Position des Eingangs, durch den er den Raum betreten hat, zugreifen und diesen als Ausgang nutzen. Weitere Zielinformationen kann er von technischen Informationsquellen erhalten. Wenn der Agent seiner geplanten Route zum Ausgang folgt, kann er als Anführer einer Massenflucht fungieren.

Das reflektive Verhaltensspektrum:
Dieses Spektrum beinhaltet alle Fähigkeiten aus dem deliberativen Bereich und zusätzlich noch die Fähigkeit zu beruhigen. Durch eine Sphäre, die diesen Effekt auf umliegende Zellen verbreitet, können beeinflusste Agenten schneller ihre Angst abbauen.

Informationen über die Umwelt
Agenten besitzen ihr eigenes Bild der Welt. Die Informationen erhalten sie, durch das Betreten der Zellen. Ob sie diese Informationen nutzen können, hängt aber auch von ihrem internen Zustand ab.

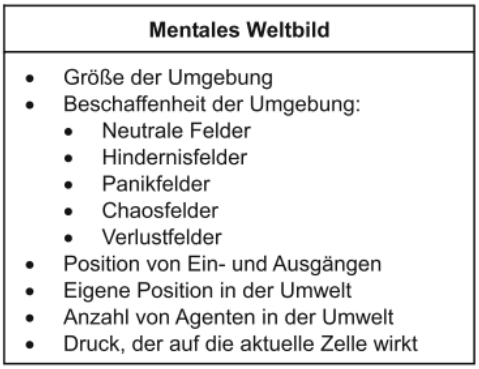


Abbildung 9 „Elemente des mentalen Weltbilds eines SimPan-Agenten“ (Schneider, 2011)

3.5 Anforderungen

3.5.1 Integration der GOAP-Komponente in das MARS-System

Die MARS Architektur orientiert sich anhand der sprachübergreifenden Quasar Standardarchitektur (Siedersleben, 2003). Das MARS System bietet für die Programmierung von Agenten große Freiheiten bezüglich der Umsetzung. Um eine Simulation zu starten wird mindestens ein Layer benötigt auf dem die Agenten „leben“ können. Um die Agenten in der Simulationsumgebung einmal pro Tick zu aktivieren, müssen diese das Interface

IAgent implementieren. Der Modellierer hat volle Freiheit was der Agent im einzelnen Simulationsschritt tut.

Da der Funktionsumfang von GOAP für viele zukünftige Modelle im MARS System einfach nutzbar gemacht werden soll, muss die Komponente eine hohe Wiederverwendbarkeit aufweisen. Modellentwickler müssen die GOAP-Elemente nach ihren Bedürfnissen modifizieren können ohne die Funktionalität zu beeinträchtigen. Zudem besteht das MARS Projekt aus immer wieder neuen Teammitgliedern, die ständig neue Anforderungen an das System mitbringen.

Die Verhaltenssteuerung über GOAP nicht die einzige für die Simulationsplattform MARS und es ist zwingend erforderlich diese austauschen bzw. abkoppeln zu können, da für viele Modelle auch schon endliche Automaten oder Ähnliches als Steuerung der Agenten vollständig befriedigend sind.

3.5.2 Nutzbarkeit für Modellentwickler

Zurzeit sind stets Programmierarbeiten nötig. Daher ergeben sich nicht-funktionale Anforderungen in Form von einfacher Integration eines Modells in die GOAP Strukturen.

Der Modellentwickler soll möglichst frei in der Umsetzung seines Modells sein. Er soll Goals, Actions und Symbole definieren, sich aber nicht um die Funktionalität der GOAP-Komponente kümmern müssen. Es ist eine einfache und übersichtliche Einweisung bzw. Anleitung für Modellentwickler nötig, da Fehlerquellen auch in der Formulierung von Actions und Goals liegen können.

Der Modellentwickler sollen die GOAP-Komponente einfach in Bezug auf das entwickelte Szenario konfigurieren können. Dazu gehören bei GOAP die, für den konkreten Agenten verfügbaren, Actions, Goals, der Worldstate mit dem er startet, und die maximale Suchtiefe im Graphen.

3.5.3 GOAP Performance

Aufgrund der wechselnden Aktivitäten in Abhängigkeit davon, ob bereits ein Plan vorliegt oder dieser gesucht werden muss, sind die benötigten Rechenzeiten immer wieder von Spitzen geprägt. Durch die Möglichkeit in Actions Kontext Vorbedingungen und Kontext Effekte mit beliebigem Programmcode zu definieren, kann die Performance von GOAP deutlich gemindert werden. Es gibt kein absolutes Zeitlimit innerhalb dessen ein Plan erstellt werden darf, es muss aber auf sparsame Operationen geachtet werden.

3.5.4 Umsetzung eines Szenario nach SimPan

Ein Aspekt ist der umfassende Test der Komponente in Zusammenarbeit mit dem MARS-System. Zu untersuchen ist, inwiefern eine Umwandlung eines Modells beziehungsweise die Überführung des Agentenverhaltens in lose Aktionen die grundlegende Struktur verändert. Fraglich ist, ob diese Änderungen noch im Sinne des Modells und zuträglich sind. Als Vorbild für das Szenario der Simulation soll SimPan von Bernhard Schneider dienen.

Für die Entwicklung und einen abschließenden Gesamteindruck der laufenden Simulation ist eine visuelle Präsentation sinnvoll. Fehler in Laufbewegungen der Agenten, kritischen Ereignissen und Statusänderungen von Agenten können somit übersichtlich und schnell erfasst werden.

4 Design

In diesem Kapitel werden die konkreten Entscheidungen bezüglicher der Entwicklung der GOAP-Komponente, der Tests und der Umsetzung der Simulation getroffen. Die Basis dafür ist das vorangegangene Analysekapitel.

Um den Prozess der Modellentwicklung und die daraus entstehenden Hürden, Besonderheiten und auch Fehlerquellen in Bezug auf die GOAP Konzepte feststellen zu können, wird anhand des Referenzmodells SimPan (Schneider, 2011) das Testszenario für die GOAP-Komponente entwickelt.

4.1 GOAP

4.1.1 Architektur

Die GOAP Elemente werden in eine Action-System-Komponente und eine Graph-Service-Komponente zerteilt. Um zyklische Abhängigkeiten und eine Vermischung von Funktionalität und Typbeschreibungen zu vermeiden, werden alle abstrakten und Interface Elemente in einem separaten Paket formuliert. Dieses wird dann durch die Action-System-Komponente und die Graph-Service-Komponente referenziert und dient als Basis der Kommunikation.

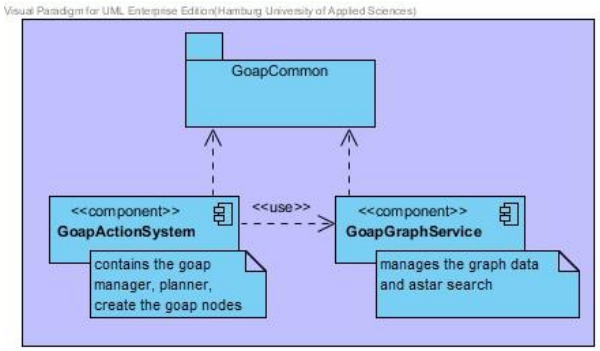


Abbildung 10 Blackbox GOAP Komponenten

Modellentwickler können mit den abstrakten Klassen aus Goap-Common eigene Actions und Goals formulieren. Die Konfiguration eines Agententyps wird bei Initialisierung der Goap-Komponente übergeben. Eine Konfiguration beinhaltet dabei die konkreten Objekte

der Actions, der Goals und eine Menge von Symbolen, die den Startzustand des Agenten festlegen.

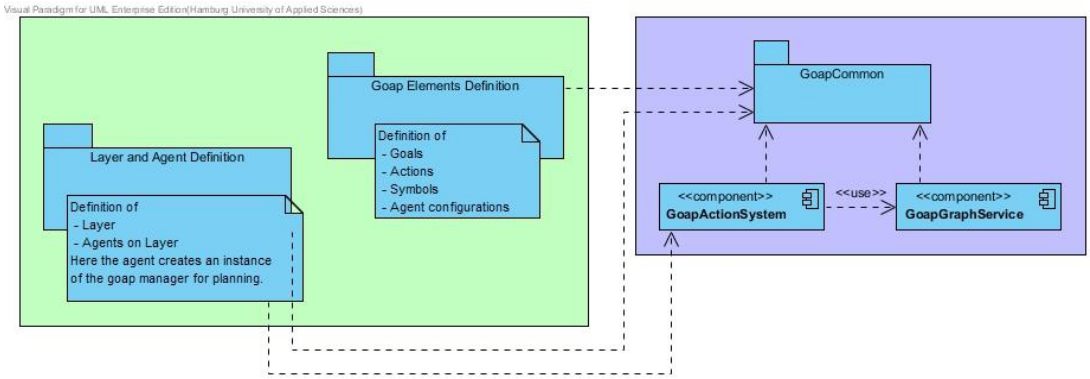


Abbildung 11 Agentendefinition und GOAP Komponenten

Die Technik befindet sich bereits im MARS System. Die zu simulierenden Layer, auf denen die Agenten „leben“, werden als Erweiterung der Layer Interfaces formuliert und über Annotationen als Extension Point mittels Mono.Addins[8] markiert. Die DLLs, die über Mono.Addins zugefügt werden, müssen im Add-ins Ordner auffindbar sein.

4.1.2 Komponente Goap-Action-System

Hier ist die Logik von GOAP untergebracht. Sie nutzt Daten aus dem Blackboard und den gegebenen Symbolen um Entscheidungen in Bezug auf Goals, Actions und die Manipulation von Symbolen durchzuführen.

Klasse GoapComponent

Hier wird die Konfigurationsklasse entgegengenommen, die übergebenen Daten validiert und damit einen Goap Manager erstellt.

Klasse GoapManager

Ist die koordinierende Instanz und hält die Informationen über verfügbare Actions, Worldstates und Goals. Hier wird das aktuell zu verfolgende Goal anhand von Prioritäten und Erreichbarkeit gewählt. Der Manager erhält Referenzen direkt oder indirekt von allen beteiligten Klassen. Er liefert auf Anfrage die nächste auszuführende Action.

Klasse GoapPlanner

Der GOAP-Planner ist zuständig für die Planentwicklung des Agenten anhand eines GOAP-Goals und der aktuellen Symbolmenge. Er nutzt dafür die zur Verfügung stehende Graphen-Komponente, prüft die in Frage kommenden Actions um eine Vorauswahl zu treffen und identifiziert zielführende und ausführbare Actions. Dabei sind die Kanten des Graphen

[8] http://monoaddins.codeplex.com/

Wrapper der Actions. Die Knoten repräsentieren einen Teilweltzustand aus Symbolen. Um optimale Ergebnisse zu liefern wird mittels A* wird die Suche im Graphen durchgeführt.

4.1.3 Schnittstellenpaket Goap-Common

Beinhaltet Interfaces und abstrakte Klassen, die durch den Modellentwickler genutzt werden können. Diese Komponente ist die Grundlage für die Verständigung der beteiligten Komponenten und verhindert zyklische Abhängigkeiten.

4.1.4 Komponente Goap-Graph-Service

Diese bietet für das Goap-Action-System den Aufbau des Graphen und die Suche darauf. Die von GOAP benötigten Operationen sind über eine Schnittstelle aus GOAP-Common bekannt. Theoretisch können hier auch komplexe Libraries angeschlossen werden.

4.1.5 Effect-Action Map

Zu den Elementen von Goap gehört an vielen Stellen eine kostengünstige Suche aufgrund der Echtzeit-Anforderungen von vielen Computerspielen. Die Effect-Action Map verkürzt die Suche nach passenden Actions (Orkin, 2005).

Bei jeder Exploration eines Knoten muss nach Actions gesucht werden, die noch nicht erfüllte Symbole befriedigen. Alle Actions müssen auf Effekte geprüft werden, die zu gesuchten Symbolen führen. Um dies effizienter zu gestalten, wird bei der Initialisierung des Goap-Managers eine Abbildung von Symbolen der Effekt Listen der Actions hin zu den Actions selbst gespeichert. Wenn eine Action mehrere Effekte aufweist, ist sie in mehreren Listen zu finden. Dadurch ist ein indizierter Zugriff mit den nicht erfüllten Symbolen des Goals möglich.

Beispiel der Inhalte der Effect-Action Map

ActionName	PreCond		Effect	
BuyToy	HasMoney	true	HasToy	true

ActionName	PreCond		Effect	
Play	HasToy	true	IsHappy	true

ActionName	PreCond		Effect	
GoJogging	-	-	IsHappy	true

WorldstateSymbol		Actions
IsHappy	true	Play; GoJogging;
HasToy	true	BuyToy;

4.1.6 Heuristik

Die Knoten im Graphen sind abstrakter als Wegpunkte. Wenn im Gelände eine Route geplant wird, kann die Heuristik sich anhand der Luftlinienentfernung zwischen aktuellem Punkt und Zielpunkt orientieren.

Jeff Orkin beschreibt für GOAP eine einfache Heuristik, die einen Wert für die Knoten anhand der Anzahl von nicht erfüllten Symbole errechnet (Orkin, 2003). Wenn also noch zwei Symbole nicht erfüllt sind beträgt der heuristische Wert 2.

Mathematisch betrachtet sollte eine Heuristik für den A* monoton sein. Monotonie geht aus zwei Kriterien heraus. Die Heuristik darf niemals überschätzen. Zweitens muss für jeden Nachfolgeknoten j von k gelten:

$$Kantenkosten\ zwischen\ v_i\ und\ v_j = \ I_{i,j}$$
$$Heuristischer\ Wert\ eines\ Knoten = \ h(k)$$

Der heuristische Wert des Vorgängers darf nicht größer sein als die Kantenkosten + der heuristische Wert des Nachfolgers.

$$h(k) \leq I_{k,j} + h(j)$$

Die Heuristik die Jeff Orkin vorschlägt, kann durch bestimmte Designs von Actions beeinträchtigt werden. In einer Beispielsituation sind zwei Symbole noch nicht erfüllt. Die Heuristik ist somit $h(k) = 2$. Wenn nun eine Action zwei Effekte besitzt, die zielführend sind und im nachfolgenden Knoten alle Symbole als erfüllt gelten, ist die Heuristik des Nachfolgeknoten: $h(j) = 0$. Wenn nun die Action und somit auch die Kante Kosten von $I_{k,j} = 1$ besitzt, ist die Bedingung nicht mehr erfüllt.

$$2 \leq 1 + 0 \quad \text{ist falsch!}$$

Die Heuristik hat überschätzt und der A* kann nicht mehr optimal arbeiten. Der eigentliche Fehler entstand hier, weil die Heuristik nicht zur Action passte. Damit die Heuristik von Jeff Orkin funktioniert, hätte die Action nur ein Symbol bei der Knotenbildung zufriedenstellen dürfen oder die Kosten der Action mit $I_{k,j} = 2$ festgelegen müssen.

4.2 Details zur Inferenzbildung

Die Bildung von Knoten für den Graphen ist Bestandteil des Planungsprozesses, und die Verwendung der Vorbedingungen und Effekte in den Knoten ist maßgeblich für nachfolgende Entscheidungen des GOAP-Planner. Bei den Beschreibungen der Bildung der Knoten sind leider einige Details nicht explizit geregelt. Obwohl Edmund Long in seiner

Masterarbeit (Long, 2007) schon detailliertere Angaben gemacht hat, bleiben noch Fragen offen.

Auffällig ist, dass scheinbar keine Action Vorbedingungen und Effekte mit gegensätzlichen Werten enthält. Diese Art der Modellierung von Übergängen in Form des „Verbrauchs“ von Ressourcen erscheint aber intuitiv. Die daraus entstehenden Problematiken werden im Kapitel 4.2.1 „Konsequenzen und Einschränkungen aus der Inferenzbildung" ausführlicher beschrieben.

Der GOAP-Planner arbeitet mit Knoten (GOAP-Nodes), die den Zielzustand und den aktuellen Zustand beschreiben. Die Spalte GoalValues startet mit der Beschreibung der Symbole des gewählten Goals unter denen es erfüllt ist und den CurrentValues aus dem Startzustand. Sobald alle CurrentValues den gleichen Wert enthalten wie die zugehörigen GoalValues ist ein möglicher Plan gefunden.

StartState („IsHappy“ => false; „HasMoney“ => true; „HasToy“ => false)
Goal („IsHappy“=> true)

Goap-Node #0

Key	CurrentValues	GoalValues
IsHappy	false (gegeben in StartState)	true

Die Nachbarn eines zu expandieren GOAP-Node werden anhand der Effekt-Action Map gesucht. Diese Vorauswahl von Actions wird zusätzlich noch auf Bestehen der Prüfung der Kontext Vorbedingungen gefiltert.

Jeder Kindknoten enthält die unerfüllten GoalValues des Elternknotens und evtl. noch weitere, falls die aktuelle Action Preconditions beinhaltet. Die Belegungen in der Spalte CurrentValues kommen aus dem Elternknoten und den Effekten der aktuellen Aktion. Die CurrentValues speichern keine Werte aus dem Startzustand, damit keine Verwechslung zwischen den Quellen stattfindet. Der Startzustand wird aber genutzt, um zu prüfen, ob die fehlenden GoalValues durch ihn erfüllt werden können.

(Die grünen Zeilen sind nicht mehr manipulierbar ist, ihr Zustand wurde bereits erfüllt.)

ActionName	PreCond		Effect	
Play	HasToy	true	IsHappy	true

Goap-Node #1 (Kind von #0 durch „Play“)

Key	CurrentValues	GoalValues
IsHappy	true	true
HasToy	false (gegeben in StartState)	true

ActionName	PreCond		Effect	
BuyToy	HasMoney	true	HasToy	true

Goap-Node #2 (Kind von #1 durch „BuyToy“)

Key	CurrentValues	GoalValues
IsHappy	true	true
HasToy	true	true
HasMoney	true (gegeben in StartState)	true

4.2.1 Fehlerquellen in der Inferenzbildung

Im Verlauf der Detailüberlegungen und Tests zum Aufbau der GOAP-Nodes konnten Situationen aufgedeckt werden, die zu Inkonsistenzen in der Zustandsrepräsentation der Knoten im Graphen führen können. Diese Fehlersituationen sind die Grundlage für die Regeln und Abläufe bei der Knotenerstellung.

Problem #1: Actions die den Wert eines Symbols negieren bzw. wechseln

StartState („IsHappy“ => false; „HasMoney“ => true; „HasToy“ => false)
Goal („HasToy“=> true)

Goap-Node #0

Key	CurrentValues	GoalValues
HasToy	false (gegeben in StartState)	true

Ein intuitives Design einer Action

ActionName	PreCond		Effect	
BuyToy	HasMoney	true	HasMoney	false
			HasToy	true

Der Ausgang dieser Konstellation ist abhängig von der oben beschriebenen Reihenfolge der Einfüge-Operationen. Das Ergebnis wird davon beeinflusst, wann und ob die Effekte eingefügt werden, und ob bereits erfüllte Zustände unveränderlich markiert werden.

Goap-Node #1 (im Fehlerfall)

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	Konflikt durch Effekt	true

Es muss eine klare Priorität bzgl. der Quellen der CurrentValues geben. Die Effekte einer Action ermöglichen erst die Befriedigung von GoalValues und sollten somit Vorrang haben. Hier wurden alle Effekte der Action in die CurrentValues eingefügt. Dies kann passieren,

wenn die Vorbedingungen der Action zuerst einsetzt werden. Bedeuten würde es, dass bei der Prüfung der Vorbedingungen die Effekte der Action genutzt werden.

Diese Probleme der Planung können durch progressive Planung umgangen werden. Dies ist z.B. bei der Variante des JMonkeyEngine[9] Programmierers Brent Owens, aus seinem Goal-Oriented Action Planning Tutorial[10] der Fall. Die regressive Suche mit GOAP wurde aber von Jeff Orkin unter anderem aus Performanceanforderungen gewählt.

Damit ein Agent seinen momentanen Zustand als Einstiegspunkt nutzen kann, muss der Startzustand Relevanz besitzen. Effekte werden nur eingefügt, wenn sie ein GoalValue erfüllen. Anschließend werden alle erfüllten Zeilen als nicht mehr manipulierbar markiert. Hier können auch Datenduplizierungen vermieden werden, indem bereits erfüllte GoalValues nicht mehr an Kindknoten weitergegeben werden. Nun kommen die Vorbedingungen der Action dazu.

Goap-Node #1 (mit obigen Regeln)

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true (gegeben in StartState)	true

Problem #2: Wie mit Actions umgehen, die Effekte auf bereits erfüllte Bedingungen ausüben?

GoapNode #n

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true	true
ToyShopInRange	false (gegeben in StartState)	true

Um die Konsistenz des bisherigen Weges im Graphen zu wahren und nicht unvorhersehbare Fehler zu provozieren, werden erfüllte Bedingungen eines Knoten nicht mehr manipuliert.

ActionName	PreCond		Effect	
UseSubway			HasMoney	false
			ToyShopInRange	true

[9] http://www.jmonkeyengine.com/

[10] http://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793

Diese Action würde mit ihrem Effekt in den CurrentValues einen erfüllten Zustand wieder zerstören und kann somit nicht angewendet werden.

GoapNode #n+1 (möglicher Fehlerfall, wenn erfüllte Values manipuliert werden)

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true/false Konflikt	true
ToyShopInRange	true	true

Effekte dürfen nur eingetragen werden, wenn sie Vorbedingungen befriedigen.

GoapNode #n+1 (möglicher Fehlerfall, Effekt hat Auswirkungen auf die Vergangenheit)

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true	true
ToyShopInRange	true	true
HasMoney	False	

Da Effekte nur angewendet werden, wenn ein unbefriedigtes GoalValue zur Verfügung steht, kommt es auch zu keinem Inkonsistenten Zustand.

GoapNode #n+1

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true	true
ToyShopInRange	true	true

Problem #3: Wie soll mit Schlüsseln umgegangen werden, die doppelt im GOAP-Node eingetragen werden müssten?

Die vorige Situation bietet noch eine weitere Problemsituation und nötige Entscheidung. Ein GoalValue wird wiederholt durch eine Action im nächsten GOAP-Node verarbeitet.

StartState („IsHappy“ => false; „HasMoney“ => true; „HasToy“ => false)
Goal („HasToy“=> true)

GoapNode #0

Key	CurrentValues	GoalValues
HasToy	false (gegeben in StartState)	true

ActionName	PreCond		Effect	
BuyToy	HasMoney	true	HasMoney	false
	HasToy	false	HasToy	true

Diese Aktion kann auch in der Planung verwendet werden, obwohl sie schon sehr restriktiv gestaltet ist. Der Modellierer könnte diese anhand eines Automaten formuliert haben. Weil wieder die Effekte zuerst genutzt werden, lässt sie sich aber einsetzen. Dabei entstehen doppelte Schlüssel Einträge.

GoapNode #0

Key	CurrentValues	GoalValues
HasToy	true	true
HasMoney	true (gegeben in StartState)	true
HasToy	false (gegeben in StartState)	false

Problem #4: Doppelte unerfüllte GoalValues durch erneutes Hinzufügen anhand der Vorbedingungen?

GoapNode #0

Key	CurrentValues	GoalValues
HasCar	false (gegeben in StartState)	true
HasToy	false (gegeben in StartState)	true

ActionName	PreCond		Effect	
BuyToy	HasMoney	true	HasMoney	false
			HasToy	true

GoapNode #1

Key	CurrentValues	GoalValues
HasToy	true	true
HasCar	false (gegeben in StartState)	true
HasMoney	false (gegeben in StartState)	true

Actions werden anhand von Effekten ausgewählt, die einen unbefriedigten GoalValue erfüllen. In dieser Situation ist schnell zu erkennen, dass beide Actions zu der Erfüllung von GoalValues beitragen, aber beide auch einen Wert „verbrauchen“ und somit nicht direkt nacheinander angewendet werden dürfen.

ActionName	PreCond		Effect	
BuyCar	HasMoney	true	HasMoney	false
			HasCar	true

GoapNode #2 (Problem durch Effekt der gegensätzlich zu GoalValue wirkt)

Key	CurrentValues	GoalValues
HasToy	true	true
HasCar	true	true
HasMoney	Konflikt	true

Wie soll nun verhindert werden, dass beide Actions nacheinander ausgeführt werden? Die Nutzung einer Action, die einen GoalValue als definitiv unerfüllt markiert, könnte untersagt werden. Eine andere Variante wäre, weiterhin nur Effekte einzutragen, die GoalValues erfüllen, im Gegenzug allerdings unerfüllte GoalValues in doppelter Angabe zu erlauben und somit abzubilden, dass zwei Actions zwei Mal einen bestimmten Wert benötigen.

GoapNode #2 (keine angemessene Alternative)

Key	CurrentValues	GoalValues
HasToy	true	true
HasCar	true	true
HasMoney	false(gegeben in StartState)	true
HasMoney	false(gegeben in StartState)	true

Gegen die doppelten Einträge spricht, dass der Worldstate eines Agenten nicht zählen kann. Dieser oben gebildete Knoten entspricht keiner möglichen Belegung im Worldstate und wird daher untersagt. Insgesamt ist hier aber der negative Effekt von „BuyCar“ das erste Problem.

Problem #5: Mögliches Problem der wiederehrenden GoalValues (ähnlich einer Schleife)?

Für eine Erkennung von Schleifendurchgängen bzw. immer wiederkehrenden Folgen, ist es von Vorteil, wenn ein Goap-Node sich nicht in seinen GoalValues aufbläht, weil er die erfüllten Zustände des Elternknotens ebenfalls beinhaltet und seine Daten nicht, mit bereits bestehenden GOAP-Nodes, gleich sind. Die Heuristik wird anhand der unerfüllten GoalValues errechnet und ist dadurch ebenfalls identisch.

Durch den A* werden bisherige Wegkosten berücksichtigt und bereits geprüfte Knoten nicht erneut bearbeitet. Zudem gibt es die Suchtiefenbegrenzung, die einen eventuell unendlich großen Graphen verkürzt.

4.2.2 Resultierende Regeln und Abläufe für die Knotenerstellung

1. Der Kindknoten erhält nur die im Elternknoten nicht erfüllten GoalValues. Dies dient der Wiedererkennung von bereits betretenen Knoten und verhindert Umwege.
2. Effekte der Action in CurrentValues werden nur eintragen, wenn diese GoalValue befriedigen. Die übrigen Effekte sind nicht relevant, weil sie nur Auswirkungen auf nachfolgende Actions haben können. Da bei GOAP aber regressiv geplant wird, treten im nächsten Planungsschritt nur Vorgänger-Actions auf.
3. Prüfung, ob durch Effekte GoalValues erfüllt wurden. Da nur Actions genutzt werden die ein GoalValue befriedigen, sollte immer mindestens eine neue Zeile dazukommen, die nicht mehr manipuliert werden darf.
4. Die Vorbedingungen der Aktion als zusätzliche GoalValues einfügen. Diese Reihenfolge bewirkt, dass Actions die Werte in EINEM Symbol wechseln ausgeführt werden können.
5. Nun folgt die Prüfung auf noch unerfüllte GoalValues.
6. Wenn noch GoalValues unerfüllt sind, wird geprüft, ob ALLE mit dem Startzustand erfüllt werden können. Ist dies der Fall, bildet der Startzustand den Einstiegspunkt und es handelt sich um einen gefundenen Plan. Wenn dies nicht den Fall ist muss weitergesucht werden.

4.2.3 Regeln um Inkonsistenzen beim Erstellen von GOAP-Nodes zu vermeiden

Durch den bereits geregelten Ablauf der Nutzung der Elternknoten-Daten und der Action-Daten ist schon vielen Fehlerfällen vorgebeugt. Eine Trennung von Vorfilterung und letztlicher Datenverarbeitung ist anzustreben. Actions die nicht genutzt werden können, sollen nicht in den Erzeugungsprozess der Knoten gelangen.

Ausschlusskriterien für Actions anhand des aktuellen GOAP-Node:

- Wenn eine Action einen Effekt hat, der einen unbefriedigten GoalValue falsch belegt, darf sie nicht genutzt werden.
- Wenn eine Action als Vorbedingung ein Symbol besitzt, das sich mit einen aktuell unbefriedigten Symbol im GOAP-Node deckt. Diese Situation ist unabhängig von den Werten, die bei Elternknoten oder Action vorhanden sind.

4.3 Erste Schritte für Modellentwickler

Szenario umsetzen
Zurzeit sind Programmierarbeiten für die Umsetzung eines Szenarios nötig. Das MARS Projekt wurde hauptsächlich mit C# erstellt, wobei auch Layer und Agenten über C# Projekte implementiert werden müssen.

1. Für das gesamte Szenario einen zusammenfassenden Ordner anlegen.
2. In MARS benötigt jeder Layer ein separates Projekt. Dabei entscheidet der Modellentwickler, welche Aspekte die Simulation beinhaltet und wie diese in Layer aufgegliedert werden können. Ein Layer kann eine Fachlichkeit repräsentieren. Zu beachten ist, dass die Layer in Abhängigkeit zu einander stehen können.
3. Für die Definition der gesamten GOAP spezifischen Klassen empfiehlt sich ebenfalls ein separates Projekt. Hier kann in Actions, Goals, Symbole und Konfiguration gegliedert werden.
4. Zuletzt müssen Konfigurationsklassen für die Initialisierung von Agenten geschrieben werden. Diese Klassen legen fest, welche Actions und Goals den Agenten zur Verfügung stehen und in welchem Startzustand sie sich befinden.
5. Um diese zu laden, werden die DLLs in den Add-ins-Ordner erstellt.

Damit Actions das Verhalten des Agenten steuern können, ist eine Erweiterung des Konstruktor nötig um die Referenz des Agenten zu erhalten und diesen manipulieren zu können. Dazu übergibt der Agent sich selbst bei der Initialisierung der GOAP-Komponente.

Agent aus bestehenden Elementen zusammenstellen
Sobald die Programmierung der GOAP Elemente eines Szenarios vorhanden ist, können aus den Goals und Actions neue Agenten durch neue Kombination von Handlungsräumen und Zielen zusammengestellt werden. Diese Zusammenstellungen können in Form von Konfigurationsklassen umgesetzt werden.

Die GOAP-Komponente konfigurieren
Über die Konfigurationsklasse kann die Verbindung zwischen dem Agent und seinen Actions erstellt werden. Die Konfigurationsklasse kann mit einer Referenz auf das zu steuernde Agentenobjekt initialisiert werden und mit dieser Referenz auch die Action Objekte initialisieren.

In der Konfigurationsklasse können die Variablenwerte des Agenten auf Symbole der GOAP-Komponente abgebildet werden. Die Übergabe der Symbole kann in Form einer Startzustandsbelegung und auch in Form von regelmäßigen Updates erfolgen.

4.4 Entwicklung des GOAP Testszenarios

Es gilt verschiedenen Konzepte und Anforderungen aus der Simulationsumgebung MARS, der GOAP Technik und dem Szenario auf Basis des Referenzmodells SimPan zusammen zu bringen und eine Beschreibung der berücksichtigten Modellelemente des Szenarios zu finden.

Um einen Vergleich zwischen den Agenten von SimPan und den GOAP Agenten ziehen zu können, wird das Prüfszenario aus Kapitel sieben in „Die Simulation menschlichen Panikverhaltens“ (Schneider, 2011) als Vorlage für die testweise Umsetzung eines Szenario genutzt. Es besteht aus einer Umgebung von 20 x 20 Zellen die nur über einen Ausgang verlassen werden kann. Die 20 menschlichen Agenten werden in unterschiedlichen Verhaltensmodi auf der Fläche gestartet. Es sind Hindernisse und Informationsquellen vorhanden. Das kritische Ereignis tritt sofort ein.

4.4.1 Die Umwelt

Um keine zyklischen Abhängigkeiten bei der Referenzierung von Objekten oder deren Layer zu gelangen, wird die Umwelt in zwei Layer aufgespalten. Die Zellen werden nicht als Agenten, sondern als passive Datencontainer-Objekte modelliert.

Die Eigenschaften der Zell-Objekte:

- Eigene ID
- Eigene Position in x, y
- Typ der Zelle (Neutral, Hindernis, Panik, Chaos, Verlust)
- Die ID eines Menschen-Agenten, falls sich ein Mensch auf ihr befindet
- Ist die Zelle ein Ausgang
- Informationen über die Position einer Ausgangszelle aufgeschlüsselt nach Quelle visuell oder anderer Agent
- Druck auf die Zelle

SimPan schreibt nicht vor, wie Aktualisierungen der Belegungen der Zellwerte durchgeführt werden sollen. Für die Aktualisierung und die Ausbreitung von kritischen Ereignissen ist ein separater Agent zuständig, der Events und Einflussbereiche umsetzt.

Die Event-Agent Eigenschaften:

- Wissen über Tick-Nummer und Zell ID um Panik Ereignisse inklusive Radius des Chaosbereiches umsetzen zu können
- Wissen über Tick-Nummer und Zell ID inklusive Wirkungsradius um Informationszellen zu erstellen
- Kann einen Menschen-Agenten „töten“ bzw. „verletzen“

4.4.2 Die Menschen

SimPan steuert Agenten über Motive. Die Herausforderung ist diese Abläufe in sinnvolle und unabhängige Actions zu zerlegen und das Verhalten der Agenten nicht zu verfälschen.

Die Eigenschaften der menschlichen Agenten:

- Eindeutige Kennung
- Angst Level
- Energie Level
- Widerstandskraft gegen Druck
- Wissen über die Umwelt in Form von Informationen über die aktuelle Zelle
- Wissen über den Eingang und somit auch Ausgang
- Wissen über eigene Position
- Können sich „auf" der Zellenwelt bewegen {N,S,W,O,NW,NO,SW,SO}
- Können Informationen auf Zellen manipulieren

Goals entwickeln

Anhand der drei Handlungsstränge erscheint es eventuell intuitiv diese durch unterschiedliche Goals umzusetzen. Goals sollten aber nicht benutzt werden um Handlungsstränge zu separieren. Dies würde dem Grundgedanken des Modells und von GOAP nicht entsprechen und könnte auch durch einen Automaten abgedeckt werden. Das eigentliche Ziel für die Menschen im Modell ist sich in Sicherheit zu bringen, das ändert sich in den Handlungssträngen nicht.

Trotzdem ist es nötig, die verschiedenen Handlungsstränge wie in SimPan abzubilden. Der Aktionsraum muss, je nach Zustand des Menschen eingeschränkt oder erweitert werden. Dadurch sind sehr wahrscheinlich häufigere Neuplanungen nötig.

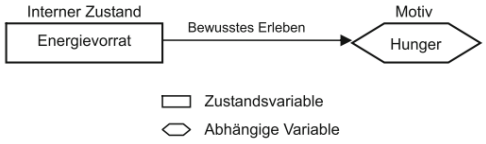


Abbildung 12 „Zustandsvariable und Motiv" (Schneider, 2011)

In SimPan werden drei Motive genannt: „Angst", „Angst Kontrolle" und „Neugier". Die Neugier spielt nur eine Rolle, wenn noch kein kritisches Ereignis eingetreten ist. Sie dient der Erforschung der Umgebung. Im, als Vorbild dienenden Prüfszenar, tritt das kritische Ereignis sofort ein und die Neugier ist nicht erforderlich.

Actions entwickeln

In SimPan sind Handlungen in Abhängigkeit vom Verhaltensmodus verfügbar. Nach den ersten Überlegungen kommt man schnell auf die Vorbedingungen der Actions. Der Sinn der Symbole ist aber nicht Actions auszuklammern, sondern Verbindungen zwischen Actions zu

schaffen. Wenn überhaupt Actions sich selbst beurteilen sollen, ob sie genutzt werden können, dann über die „Kontext Vorbedingungen“.

Weiterhin kommt die Frage auf, ob es sinnvoll ist, ungeplantes Verhalten oder sogar die Handlungsstarre mittels GOAP abzubilden. GOAP dient der Planung und nicht als Zustandsmaschine. Es kann Situationen geben, in denen es sinnvoller ist den Agenten nicht durch GOAP planen zu lassen, weil es keine Auswahl gibt.

Für die drei Verhaltensstränge von SimPan werden jeweils eigene Konfigurationen erstellt. Die Agenten können zur Laufzeit die Konfiguration wechseln, wenn der bestimmende Angst-Level sich ändert.

Abbildung 13 „Berücksichtigte Verhaltensweisen im Überblick“ (Schneider, 2011)

In der Abbildung 12 sind Gemeinsamkeiten bei den unterschiedlichen Handlungssträngen z.B. bei „Selbständige Zielauswahl“ oder „Druck/ Drängeln“ zu erkennen. Der Vorteil darin ist, dass Actions gebildet und in mehreren Verhaltensmodi verwendet werden können.
Menge der Actions:

- Zielloses Umherirren
- An Massenflucht teilnehmen
- Annäherung an den Ausgang
- Ein Ziel auswählen
- Wegplanung zu einem Ziel
- Route zu Ziel verfolgen

Zusätzlich:

- Die Simulation über den Ausgang verlassen

Symbole entwickeln
Hier ist zu beachten, dass die Symbole der Planung dienen und nicht die gesamte Variablenmenge eines Agenten darstellen sollen (Orkin, 2006). Schwierig ist leider auch die Modellierung von Werten, die sich an einen numerischen Zielwert schrittweise annähern, wenn die Anzahl der Schritte vorher nicht klar ist. Der GOAP-Planner kann nicht beurteilen, ob ein Symbol eine Annäherung zu einem Anderem darstellt. Er kann Symbole nur auf Gleichheit prüfen. Ansonsten könnte die Prüfung auf Annäherung an den Ausgang über Symbole geleistet werden.

Voraussichtliche Symbole:

- Boolean: Hat ein Ziel
- Boolean: Befindet sich im Ausgangsbereich
- Boolean: Hat einen Pfad zum Ausgang
- Boolean: Kennt einen Ausgang

Zusätzlich:

- Boolean: Befindet sich auf dem Ausgang
- Boolean: Befindet sich außerhalb der Simulation

Sensoren
Menschen-Agenten nehmen ihre Umwelt nur durch den Zugriff auf die Informationen einer Zelle wahr. Zusätzlich ändern die Agenten die Zellenwerte. Dabei ist es nötig den konkurrierenden Zugriff bei Schreiboperationen zu synchronisieren, damit keine Inkonsistenzen oder Fehler auftreten.

4.4.3 Resultierende Layer in MARS

Der Cell-Layer
Als Startinformationen sind die Größe einer einzelnen Zelle, die mögliche Typen der Zelle und die Länge und Breite des Areals nötig. Der Cell-Layer muss als erstes initialisiert werden, weil die menschlichen Agenten ihre Position speichern und die Zellen als belegt markiert werden müssen. Zellen werden als Draufsicht im Zweidimensionalen Raum visualisiert.

Die Merkmale des Cell-Layer:
- Referenziert keine anderen Layer
- Ist nötig für die Modellierung von Layerabhängigkeiten ohne Zyklen
- Hält die abrufbaren Zelleneigenschaften
- Die Werte der Zellen können manipuliert werden
- Besitzt eine Startbelegung jeder einzelnen Zelle des Areals

Der Event-Layer
Kritische Ereignisse werden auf dem Event-Layer umgesetzt. Die Ereignisse können anhand von Zell-Koordinaten und Eintrittszeitpunkt in Form von Ticks ab Start der Simulation definiert werden. Ein kritisches Ereignis besitzt auch immer einen Kenntnisbereich. Dieser drückt aus, auf welchen Nachbarzellen die Information über das kritische Ereignis für menschliche Agenten abrufbar wird.

Die Merkmale des Event-Layer:
- Referenziert den Cell-Layer um Zellen zu manipulieren um z.B. den Status zu ändern
- Referenziert den Human-Layer um Menschen zu töten, wenn sie Opfer eines kritischen Ereignisses werden

Der Human-Layer
Die Agenten stellen Flüchtende dar, können als Typgleich betrachtet und somit zu einer Gruppe im Human-Layer zusammengefasst werden. Der Human-Layer hat Lese- und Schreibzugriff auf dem Cell-Layer.

- Layer hat Zugriff auf Cell-Layer
- Layer wird manipuliert durch Event-Layer

In der nachfolgenden Abbildung 14 ist die Referenzierung der Layer untereinander nochmals übersichtlich dargestellt. Die Aufspaltung der Umwelt in die passiven Zellen und den Event-Agenten verhindert zyklische Abhängigkeiten.

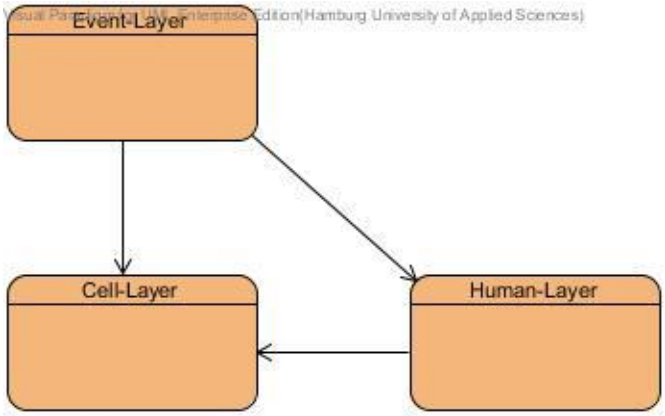


Abbildung 14 Darstellung der Abhängigkeiten unter den Layern

4.4.4 Szenario Daten

Es gilt Merkmale zu finden, die zeigen können, ob, die in GOAP implementierten Agenten, dem Referenzmodell grundlegend entsprechen. Dabei können z.B. insbesondere Daten genutzt werden, die eine bestimmte Dynamik im Verlauf der Simulation repräsentieren.

Ereignisse in der Simulation

1. Das Auflösen der Handlungsstarre
2. Todesfälle durch zu hohen Druck
3. Todesfälle durch ein kritisches Ereignis
4. Wechsel zwischen den Verhaltensweisen (reaktives, deliberatives und reflektives Verhalten)
5. Das kritisches Ereignis und dessen Ausdehnung
6. Die Zerstörung von Hindernissen durch Druck

4.4.5 Visualisierung

Die Visualisierung der Simulation soll an die Darstellung aus SimPan angelehnt sein. Agenten werden als einfache gefüllte Kreise auf einem Zellraster gezeichnet. Die Zustände der Agenten und Zellen werden durch unterschiedliche farbliche Hervorhebungen ausgedrückt.

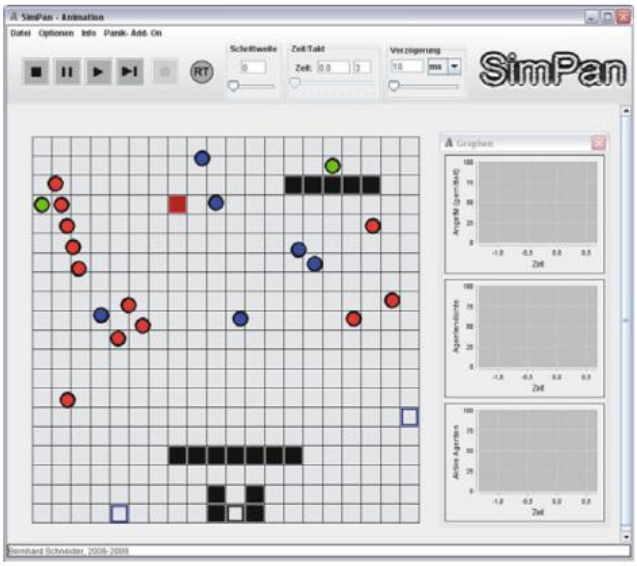


Abbildung 15 „Visualisierung des initialen Zustands des Prüfszenars“ (Schneider, 2011)

Visualisierungsdaten

Für die grundlegende Darstellung von Agenten werden die Koordinaten und die Füllfarbe benötigt. Die Füllfarben sind abhängig vom Verhaltenszustand des Agenten.

- Reaktive Agenten und Agenten in Handlungsstarre sind rot
- Deliberative Agenten sind blau
- Reflektive Agenten sind grün
- Agenten, die durch ein kritisches Ereignis gestorben sind, werden hellblau

Die Zellen der Simulation bilden den Bewegungsraum für die Agenten und werden zustandsabhängig farblich gekennzeichnet.

- Neutrale Zellen werden hell gefüllt
- Hinderniszellen werden dunkelgrau gefüllt
- Panikzellen werden dunkelrot gefüllt
- Chaoszellen, die einen Trümmerbereich darstellen, werden hellrot gefüllt
- Verlustzellen, auf denen Agenten gestorben sind, werden lila gefüllt
- Zellen die im Umkreis eines deliberativen Agenten eine Beruhigungssphäre haben werden hellgrün gefüllt
- Die Zellen um einen Agenten, der eine Massenflucht anführen kann, stellen die Massenflucht-Koordinaten dar
- Zellen um einen Ausgang stellen die Koordinaten des Ausgangs dar

5 Realisierung

Dieses Kapitel gibt die Hauptpunkte der Umsetzung der GOAP-Komponente und des abgewandelten SimPan-Szenarios wieder. Zuerst werden Implementierungsdetails und Abläufe der GOAP-Komponente dargestellt. Anschließend werden die Goals, Actions und Symbole der Simulation konkretisiert. Zum Schluss werden Details zur Entwicklung der Simulation präsentiert.

5.1 GOAP

Die GOAP-Komponente konnte unabhängig von anderen MARS Komponenten umgesetzt werden und ist im Bereich LIFE Services angeboten. Sie besitzt lediglich Abhängigkeiten zum Blackboard, dem Schnittstellenpaket GOAP-Common und der Graph-Komponente.

5.1.1 Schnittstellen in GOAP-Common

Hier sind alle öffentlich benötigten Schnittstellen, abstrakte Klassen und konkreten Klassen untergebracht, die für die Benutzung oder Zusammenarbeit mit der GOAP-Komponente notwendig sind. Um die Grundfunktionalität für GOAP sicherzustellen, sind die Mindestinformationen für Goals und Actions in Form von abstrakten Klassen inklusive der Methoden, die für die Funktionalität von GOAP erwartet werden, implementiert.

Details: AbstractGoapAction

Eine Action besteht aus einer nicht leeren Effekt-Liste und einer, bei Bedarf leeren, Liste von Vorbedingungen.

Durch virtuelle Methoden ist ein Standardverhalten bereits implementiert:

- Die Ausführungskosten betragen eins.
- Die Kontext-Vorbedingungen und -Effekte sind immer erfüllt.
- Die Action wird immer als beendet betrachtet.
- Die Anfrage der zurückgesetzten Kopie der Action wird mit „this“ beantwortet, weil im Standard keine Werte an Actions geändert werden.
- Gleichheit der Actions ist anhand des Typs des Objektes und der Gleichheit der enthaltenen Elemente der Vorbedingungen und Effekte gegeben.

Bei der minimalen Ableitung der Klasse wird der Konstruktor und die Execute Methode gefordert. Diese minimale Definition wurde für die Implementierung der Tests genutzt und

bietet für Modellentwickler eine einfache Möglichkeit, erste Probeläufe mit der GOAP-Komponente durchzuführen.

Details: AbstractGoapGoal
Ein Goal definiert mindestens einen Zielzustand anhand einer nicht leeren Liste von Symbolen. Diese Liste und ein Startwert für die Priorität müssen mindestens gegeben werden. Im Standard wird die Priorität nicht verändert. Die minimale Ableitung der Klasse bietet ebenfalls die Möglichkeit, schnelle Tests zu erstellen.

Hier kann ebenfalls der Konstruktor erweitert, der zugehörige Agent referenziert und somit die Aktualisierung der Priorität durch Werte des Agenten gesteuert werden.

Details: WorldstateSymbol
Symbole kommen bei der Planung immer wieder doppelt vor, müssen verglichen und kopiert werden. Um dem Risiko der versehentlichen Manipulation vorzubeugen, wurden Symbole nicht als Klasse, sondern als C# Wert-Typ „struct“ implementiert.

Felder sind das Schlüssel-Enum (Bezeichnung des Symbols), ein konkreter Wert des C# „Value-Type“ (z.B. true oder 10) und die Typbezeichnung (z.B. Boolean oder Integer).

Details: IGoapAgentConfig
Anhand dieses Interface können diverse Konfigurationsklassen und somit verschiedene Agententypen aus dem Pool der Goals und Actions eines Modells erstellt werden.

Über das Interface werden folgende Informationen für GOAP bereitgestellt:

- Der Startzustand
- Alle verfügbaren Actions und Goals
- Die maximale Suchtiefe des Graphen
- Ob berücksichtigt werden soll, dass Actions beendet sind
- Ob und wann die Belegungen des Worldstate aktualisiert werden sollen
- Ob die Goals ihre Priorität bei Neuplanung erneuern sollen
- Die Aktualisierungsfunktion des Worldstate

Details: GoapNode
Die GoapNodes repräsentieren Teilzustände bei der Suche nach einem Plan. Bei Erzeugung werden die unbefriedigten GoalValues identifiziert und der heuristische Wert durch den GOAP-Planner berechnet. Die Knoten werden durch den GOAP-Planner erzeugt und an die Komponente Goap-Graph-Service weitergeleitet.

Details: GoapEdge
Dient als Verbindung zwischen GoapNodes und repräsentiert die Action, mit welcher die Verbindung geschaffen werden konnte. Die Kosten entsprechen genau den

Ausführungskosten der zugehörigen Action. Die Kanten des Graphen werden durch den GOAP-Planner erstellt und an die Komponente Goap-Graph-Service weitergeleitet.

5.1.2 GOAP-Action-System

Details: GoapComponent (statisch)
Nimmt die Konfigurationsklasse via Reflektion entgegen, validiert die Startparameter, erzeugt den GoapManager und gibt ihn an den Aufrufer zurück.

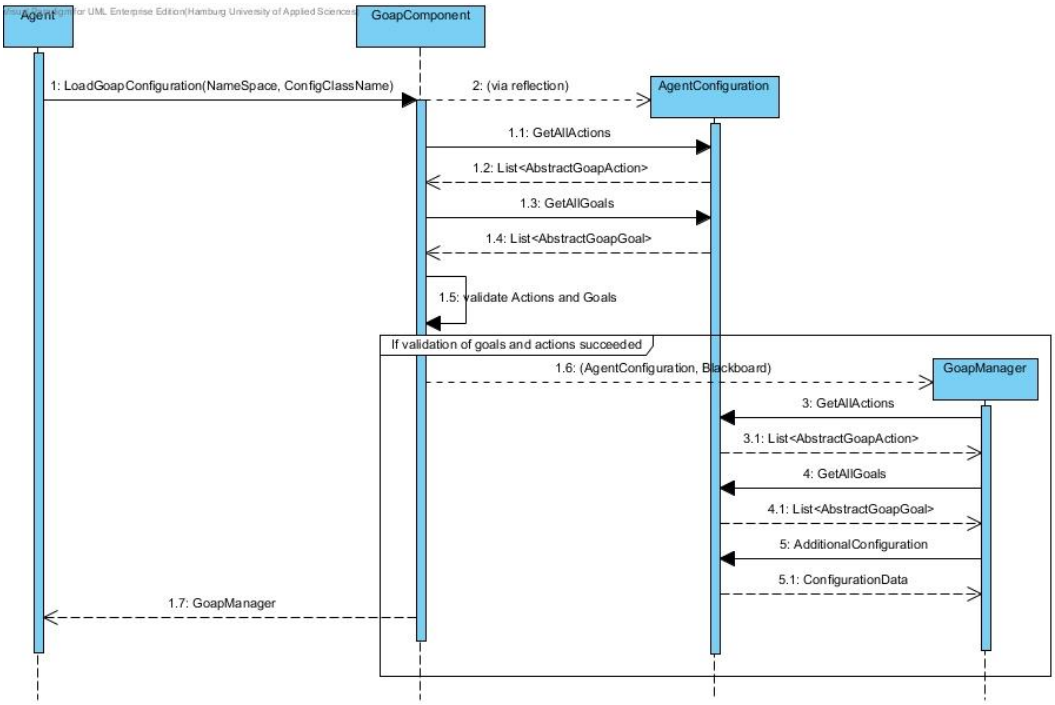


Abbildung 16 Initialisierung der GOAP-Komponente

Hier sind zwei Methoden zur Erstellung des GoapManager verfügbar. Beide fordern den Namen der Konfigurationsklasse, ihren Namensraum und eine Instanz des Blackboards. Bei der zweiten Variante können beliebige Parameter als Array für den Konstruktor der Konfigurationsklasse übergeben werden.

Bevor der GoapManager erstellt wird, werden die Actions und Goals noch geprüft:

- Die Liste der Actions und Goals darf nicht leer sein, sonst wird eine „ArgumentException“ geworfen.
- In Goals darf innerhalb der Symbolliste kein Schlüssel doppelt vorkommen, sonst wird eine „GoalDesignException“ geworfen.

- In Actions darf innerhalb der Listen der Preconditions oder Effects kein Symbolschlüssel doppelt vorkommen, sonst wird eine „ActionDesignException“ geworfen.

Details: GoapManager
Er kann nur über die Klasse GoapComponent erzeugt werden, um Prüfungen auf den Parametern durchführen zu können. Der GoapManager ist von der abstrakten Klasse AbstactGoapSystem abgeleitet. Diese definiert die Blackboard-Einträge ActionForExecution und Worldstate. Innerhalb dieser kann der GoapManager die aktuelle Action und den aktuellen Worldstate speichern. Zusätzlich ist die Implementierung der Methoden GetNextAction und ForceReplanning vorgegeben. Das Erzwingen der Neuplanung ist eine zusätzliche Service Methode, die vom Modellprogrammierer genutzt werden kann.

Details: GoapPlanner
Der GOAP-Planner ist kein permanentes Objekt und wird nur bei Erstellung eines neuen Plans benötigt. Er wird mit der Map von Symbolen zu Actions, der Liste von verfügbaren Actions und dem Start-Worldstate durch den GoapManager initialisiert. Erst beim Suchauftrag nach einem Plan erhält er das entsprechende Goal, weil es wahrscheinlich ist, dass Goals auch nicht erreichbar sind und mehrere getestet werden müssen. Der GOAP-Planner bedient sich der Graph-Komponente um den Suchvorgang mit A* durchzuführen.

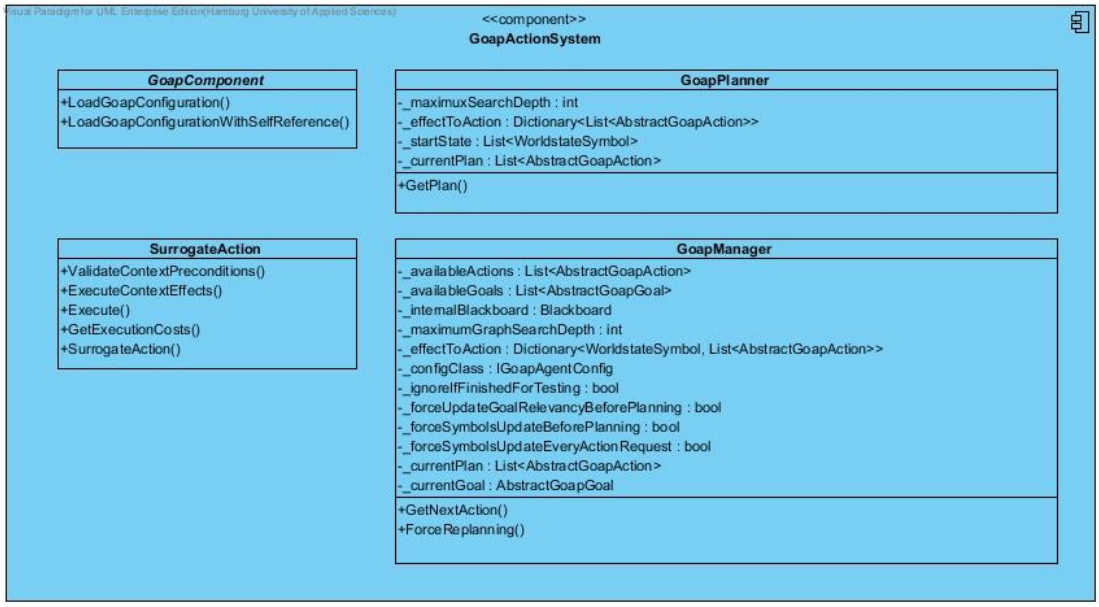


Abbildung 17 Klassen im Goap-Action-System

5.1.3 Anfrage der nächsten Action

Die GOAP Komponente besitzt grundlegend „nur“ die Funktion, die nächste gültige Action zurückzuliefern. Je nach Zustand sind unterschiedlich viele Berechnungen nötig und steuernde Objekte involviert.

Der Agent fragt bei seinem GoapManager die nächste Action an. Wenn der GoapManager noch einen Plan gespeichert hat, gibt er die nächste Action aus diesem zurück. Falls er keine Action mehr vorrätig hat, startet er den GOAP-Planner und übergibt ihm das Goal mit der höchsten Priorität, um für dieses einen Plan zu suchen. Falls das Goal nicht erreichbar ist, lässt der GOAP-Manager den GOAP-Planner mit dem Goal der zweiten Priorität suchen usw..

Im GOAP-Planner wird aus dem Goal der Wurzelknoten gebildet und an den GoapSimpleGraphService weiter gegeben. Der GOAP-Planner lässt sich nun immer wieder den vom A* gewählten Knoten zurückgeben und expandiert danach den Graphen, indem er passende Actions sucht und daraus anzuhängende Kanten und Knoten bildet. Dies wiederholt sich, bis ein Plan gefunden wurde. Falls die maximale Suchtiefe überschritten wurde oder keine weiteren Knoten angehängt werden konnten, wird die leere Action zurück gegeben.

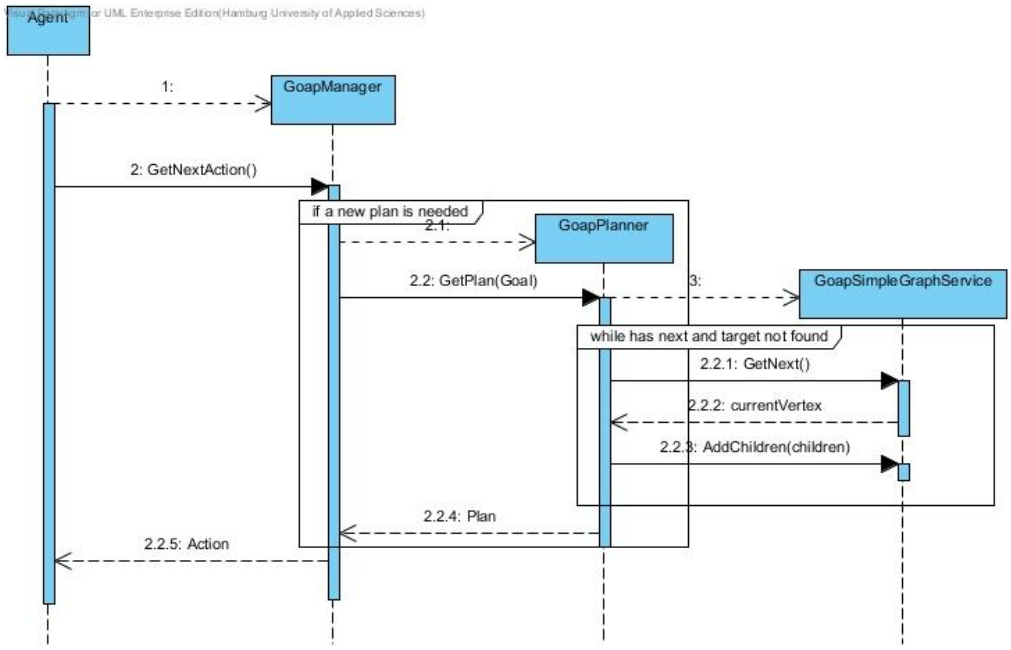


Abbildung 18 Ablaufdiagramm: Anfrage der nächsten Action.

5.2 Blackboard

Weil das Konzept des Blackboards schon lange bekannt ist, wurde nach einer einfachen Fremdimplementierung gesucht. Die Entscheidung ist auf das „Type safe blackboard“[11] von Antonio Nakić Alfirević unter der CPOL 1.02 Lizenz gefallen.

Um diese Implementierung nutzen zu können, müssen erst BlackboardProperties definiert werden. Diese beinhalten einen Typ und einen String der den Namen repräsentiert. Für GOAP sind zwei Properties benötigt und bereits im AbstractGoapSystem vordefiniert.

```
/// <summary>
///     blackboard property
///     The current action.
/// </summary>
public static readonly BlackboardProperty<AbstractGoapAction> ActionForExecution =
    new BlackboardProperty<AbstractGoapAction>("ActionForExecution");

/// <summary>
///     blackboard property
///     The current world state in the goap system
/// </summary>
public static readonly BlackboardProperty<List<WorldstateSymbol>> Worldstate =
    new BlackboardProperty<List<WorldstateSymbol>>("Worldstate");
```

Der Modellprogrammierer kann auf diese Weise die Informationen des Blackboards beliebig erweitern und Werte zwischenspeichern.

5.3 Goap-Graph-Service

Der Ablauf besteht aus einem inkrementellen Graphenaufbau. Der A* wird immer nach dem Entnehmen des aktuellen Knotens mit Prüfung auf „Ziel erreicht?“ aus der Open List unterbrochen. Die Kinder (Kanten, die Actions repräsentieren) des gewählten Knotens werden nun vom GOAP-Planner ausgewählt, und somit wird der Graph erweitert.

Dies bedeutet, dass der Graph nach jedem Schritt des A-Stern erweitert werden muss. Konkret: Der Wurzelknoten und die möglichen Kinder werden initialisiert. Wenn der zu explorierende Knoten feststeht, muss dieser wiederum mit Kindern versehen werden.

5.3.1 Libraries für Graphen und zugehörige Algorithmen

Das präferierte Ziel, eine umfangreiche Library für Graphen und Suchfunktionen zu finden, die auch gleichzeitig für das restliche MARS-System gut geeignet ist, hat sich leider nicht in angemessener Zeit durchführen lassen. Auf den ersten Blick gibt es ein breites Angebot,

[11] http://www.codeproject.com/Articles/451326/Type-safe-blackboard-property-bag

wie z.B. „thinksharp“[12] oder „AForge.NET framework“[13] aber viele Implementierungen haben Schwächen oder sind seit langer Zeit nicht mehr gepflegt worden.

Als vielversprechendste Umsetzung erschien zuerst „QuickGraph“[14]. Durch die Vorgehensweise, den Graphen nach jeder Auswahl eines nächsten zu behandelnden Knotens zu erweitern, ist allerdings eine eigene Ableitung eines A* Algorithmus nötig geworden. Die Realisierung in QuickGraph hat sich aufgrund der Dokumentation, die sich unter anderem durch den Wechsel zwischen den teilweise inkompatiblen Versionen 2.0 und 3.0 (inklusive abweichenden Funktionsaufrufen) auszeichnet, als nicht sinnvoll dargestellt.

Das Ziel, eine durch weitere Komponenten nutzbare Library in MARS zu nutzen, wurde ersetzt durch eine schlanke, genau auf die Bedürfnisse des Planers von GOAP zugeschnittene Lösung. Dabei wurde die zukünftige Möglichkeit, eine umfangreichere Komponente anzubinden, nicht außer Acht gelassen. Interfaces und gemeinsame Typen bieten die Möglichkeit, ein anderes System leicht anzuschließen.

5.3.2 Goap-Simple-Graph-Service

Dieser implementiert das Interface IGoapGraphService, wird von der GOAP-Komponente erzeugt und für die Suche nach einem Plan benutzt. Die Klasse GoapSimpleGraphService arbeitet dabei als Fassade. Der Graph kann am aktuell zu explorierenden Knoten durch den GOAP-Planner erweitert werden. Aufrufe werden zu AStarSteppable und/oder zur Graphenklasse weitergeleitet.

5.4 GOAP Elemente der Simulation

Anhand des Referenzmodells wurden vereinfachte Goals und Actions entwickelt. Nachfolgend sind die konkreten Bezeichnungen und die Vorbedingungen und Effekte genannt.

5.4.1 Goals

GoalName	key	value
BeOutOfDanger	IsOutside	true

Die Agenten wollen den Gefahrenbereich verlassen.

[12] https://code.google.com/p/thinksharp/
[13] http://www.aforgenet.com/
[14] https://quickgraph.codeplex.com/

5.4.2 Actions reaktiver Agent

ActionName	PreCond		Effect	
AimlessWandering	HasTarget	false	IsInExitArea	true

Der Agent wählt eine zufällige Laufrichtung und verfolgt diese. Kommt er nicht weiter, wechselt er die Laufrichtung wieder zufällig. Ziel ist das zufällige Auffinden des Ausgangsbereiches.

ActionName	PreCond		Effect	
ParticipateMassEscape	HasTarget	True	IsInExitArea	true

Diese Action ist verfügbar, wenn ein Anführer einer Massenflucht in der Nähe ist.

ActionName	PreCond		Effect	
AggressiveApproximation	IsInExitArea	true	IsOnExit	true

Der Agent versucht sich dem Ausgang zu nähern und drängelt, wenn er keine Fortschritte bei der Annäherung macht. Durch Drängeln wird Druck ausgeübt und Hindernisse können überrannt oder andere Agenten getötet werden.

5.4.3 Actions deliberativer Agent

ActionName	PreCond		Effect	
ChooseExit	KnowsExitLocation	true	HasTarget	true

Der Agent wählt einen Ausgang anhand seiner vorhandenen Informationen.

ActionName	PreCond		Effect	
FindPathToExit	HasTarget	true	HasPath	true

Der Agent plant eine Route zu dem gewählten Ziel.

ActionName	PreCond		Effect	
FollowPathToExitArea	HasTarget	true	IsInExitArea	true

Der Agent folgt der Route bis zum Ausgangsbereich. Wenn auf der Route ein anderer Agent den Weg versperrt, wird auf die betreffende Zelle Druck ausgeübt und der andere Agent kann getötet werden.

ActionName	PreCond		Effect	
AggressiveApproximation	IsInExitArea	true	IsOnExit	true

Der Agent versucht sich dem Ausgang zu nähern und drängelt, wenn er keine Fortschritte bei der Annäherung macht. Durch Drängeln wird Druck ausgeübt und Hindernisse können überrannt oder andere Agenten getötet werden.

5.4.4 Actions reflektiver Agent

ActionName	PreCond		Effect	
ChooseExit	KnowsExitLocation	true	HasTarget	true

Der Agent wählt einen Ausgang anhand seiner vorhandenen Informationen.

ActionName	PreCond		Effect	
FindPathToExit	HasTarget	true	HasPath	true

Der Agent plant eine Route zu dem gewählten Ziel.

ActionName	PreCond		Effect	
FollowPathToExit	HasPath	true	IsOnExit	true

Der Agent folgt der Route direkt bis zum Ausgang. Wenn auf der Route ein anderer Agent den Weg blockiert, kann er seine Route verwerfen und eine neue planen.

5.4.5 Gemeinsame Actions

ActionName	PreCond		Effect	
GoOut	IsOnExit	true	IsOutside	true

Der Agent benutzt den Ausgang.

5.4.6 Symbole

Aus der Formulierung der Actions sind folgende Symbole für die Verkettung zu Plänen festgelegt worden.

Worldstate	Beschreibung
IsOutside (bool)	Simulationsbereich erfolgreich verlassen und in Sicherheit
IsOnExit (bool)	Befindet sich auf dem Ausgang
IsInExitArea	Befindet sich im Ausgangsbereich (bestimmter Radius um Ausgang)
HasTarget	Das Ziel, dass gewählt wurde (hat aus einer Menge von Zielen eines zum aktuellen gewählt)
HasPath	Kennt einen Weg zu einem Ausgang
KnowsExitLocation	Kennt die Koordinaten eines Ausgangs

5.5 Implementierung der Simulation mit GOAP

Dieser Abschnitt dient gleichzeitig als Strukturhilfe für spätere Modellprogrammierer, welche die GOAP-Komponente nutzen wollen.

Nachdem die Layer, deren Abhängigkeiten und die Formulierung der vereinfachten SimPan Elemente für GOAP bereits entwickelt wurden, können die Projekte der Simulation in MARS angelegt werden.

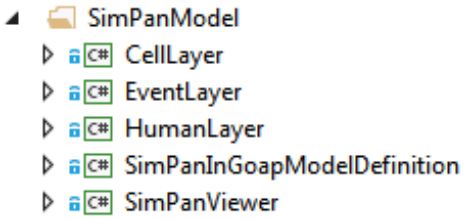


Abbildung 19 Projektmappenstruktur der Simulation

Damit Actions den zugehörigen menschlichen Agent manipulieren bzw. steuern können, ist ein Verweis auf den Menschen-Layer im Projekt der Modelldefinition nötig. Die Visualisierung wird durch den Cell-Layer gestartet und aktualisiert. In der nachfolgenden Abbildung 18 sind die weiteren Abhängigkeiten aufgezeigt.

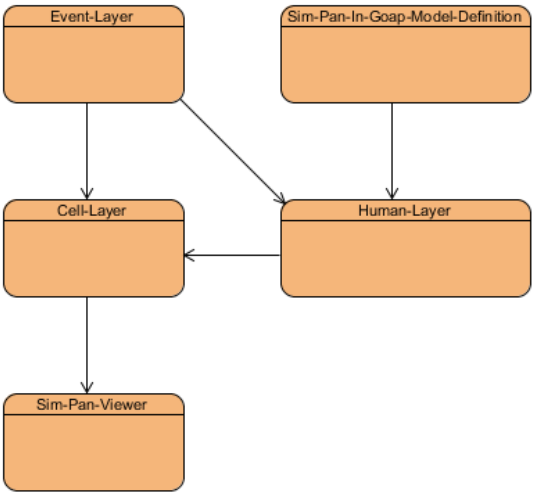


Abbildung 20 Abhängigkeiten der Projekte der Simulation

In Anlehnung an die C4-Architektur wurde die Implementierung des menschlichen Agenten auf folgende fachliche Bereiche aufgespalten:

- Die Human-Klasse, welche den Agenten repräsentiert und das MARS Interface „IAgent“ implementiert um den Simulationstick zu empfangen. Hier werden die nachfolgenden Klassen erzeugt und referenziert.
- Eine separate Klasse für Sensordatenaufnahme und Aufbereitung
- Eine Instanz der Goap-Komponente
- Eine separate Klasse für die Bereitstellung der Bewegungsvarianten und des Managements der zugehörigen Daten auf dem Cell-Layer.

Im Agenten sind anschließend alle Methoden, die für die Interaktion mit dem Cell-Layer, der Verarbeitung von Umweltdaten usw. benötigt werden, implementiert worden. Durch die Actions eines Agenten werden Kombination und Reihenfolge der Aufrufe gesteuert.

Der Human-Layer übernimmt die Initialisierung der menschlichen Agenten. Dabei kann die Anzahl der reaktiven, deliberativen und reflektiven Agenten einzeln bestimmt werden. Die Positionierung auf dem Zellenfeld erfolgt zufällig um möglichst verschiedene Startbelegungen zu erreichen.

Die Angst der menschlichen Agenten wird in jedem Simulationsschritt um einen kleinen Betrag reduziert. Falls eine Beruhigungssphäre vorhanden ist, erhöht sich dieser Betrag. Für die drei Verhaltensmodi sind im Human-Layer Grenzwerte für die Angstt vorgegeben. Sobald ein Agent in einen anderen Angst-Bereich gelangt, wechselt er seinen GoapManager gegen einen neuen mit der passenden Konfiguration aus.

Auf dem Event-Layer wird ein Event-Agent gestartet, der eintretende kritische Ereignisse umsetzen kann. In der umgesetzten Simulation findet das kritische Ereignis sofort statt .

5.5.1 Konfiguration von GOAP

Um die verschiedenen Verhaltensstränge abzubilden, wurden drei Konfigurationen für die Agenten erstellt. In der folgenden Abbildung 20 ist die Konfiguration des reaktiven Agenten zu sehen.

```
public class ReactiveConfig : IGoapAgentConfig {
        private readonly Human _human;
        private readonly Blackboard _blackboard;

        public ReactiveConfig(Human human, Blackboard blackboard) {
            _human = human;
            _blackboard = blackboard;
        }

        #region IGoapAgentConfig Members

        public List<WorldstateSymbol> GetStartWorldstate() {
            return GetUpdatedSymbols();
        }

        public List<AbstractGoapAction> GetAllActions() {
            return new List<AbstractGoapAction> {
                new GoOut(_human),
                new AggressiveApproximation(_human),
                new AimlessWandering(_human),
                new ParticipateMassEscape(_human)
            };
        }

        public List<AbstractGoapGoal> GetAllGoals() {
            return new List<AbstractGoapGoal> {
                new BeOutOfDanger(),
            };
        }

        public int GetMaxGraphSearchDepth() {
            return 20;
        }

        public bool IgnoreActionsIsFinished() {
            return false;
        }

        public bool ForceSymbolsUpdateBeforePlanning() {
            return true;
        }

        public bool ForceSymbolsUpdateEveryActionRequest() {
            return true;
        }

        public bool ForceGoalRelevancyUpdateBeforePlanning() {
            return false;
        }

        public List<WorldstateSymbol> GetUpdatedSymbols() {
            List<WorldstateSymbol> updatedSymbols = new List<WorldstateSymbol> {
                new WorldstateSymbol(Properties.IsOutSide, _blackboard.Get(Human.IsOutSide), typeof (Boolean)),
                new WorldstateSymbol(Properties.IsOnExit, _blackboard.Get(Human.IsOnExit), typeof (Boolean)),
                new WorldstateSymbol(Properties.IsInExitArea, _blackboard.Get(Human.IsInExitArea), typeof (Boolean)),
                new WorldstateSymbol(Properties.HasTarget, _blackboard.Get(Human.HasTarget), typeof (Boolean)),
                new WorldstateSymbol(Properties.HasPath, _blackboard.Get(Human.HasPath), typeof (Boolean)),
                new WorldstateSymbol
                    (Properties.KnowsExitLocation, _blackboard.Get(Human.KnowsExitLocation), typeof (Boolean))
            };
            return updatedSymbols;
        }

        #endregion
    }
```

Abbildung 21 Die Konfigurationsklasse des reaktiven Agenten

In der Simulation sind Actions vorhanden, die mehrere Simulationsschritte nacheinander ausgeführt werden müssen. Dazu gehört z.B. das ziellose Umherwandern. GOAP wurde daher so konfiguriert, dass berücksichtigt wird, ob die Actions bereits beendet oder weiter ausgeführt werden müssen. Alle Actions überschreiben dazu „IsFinished“.

Das Ziel in diesem Szenario ist, dass die Agenten ihre Pläne sofort anpassen, wenn sich Werte ändern. Dazu wurde die Aktualisierung der Symbole für jede Anfrage einer Action konfiguriert. In der letzten Methode werden die bereits aufgearbeiteten Daten des Agenten auf die Symbole der GOAP-Komponente abgebildet und somit aktualisiert.

5.5.2 Design der Actions

Nachfolgend ist die Action der Schrittweisen aggressiven Annäherung zu sehen. Diese Klasse benötigt die Möglichkeit zu zählen um eine Abbruchbedingung zu schaffen. Sobald ein Bewegungsversuch nicht erfolgreich war, wird im darauf folgenden Druck aufgebaut.

```
/// <summary>
    ///     Is used for humans in reactive behaviour mode. If the human fails in moving he will add pressure to
    ///     the cell in the chosen direction.
    /// </summary>
    public class AggressiveApproximation : AbstractGoapAction {
        private const int MaximumAttemps = 30;
        private readonly Human _human;
        private int _previousAttemps;

        public AggressiveApproximation
            (Human human) :
                base(
                new List<WorldstateSymbol> {new WorldstateSymbol(Properties.IsInExitArea, true, typeof (Boolean))},
                new List<WorldstateSymbol> {new WorldstateSymbol(Properties.IsOnExit, true, typeof (Boolean))}) {
            _human = human;
            _previousAttemps = 0;
        }

        public override bool ValidateContextPreconditions() {
            if (_previousAttemps <= MaximumAttemps) {
                return true;
            }
            return false;
        }

        public override void Execute() {
            _previousAttemps += 1;
            if (_human.HumanBlackboard.Get(Human.MovementFailed)) {
                _human.MotorAndNavigation.ApproximateToTarget(aggressiveMode: true);
            }
            else {
                _human.MotorAndNavigation.ApproximateToTarget();
            }
        }

        public override bool IsFinished() {
            return _human.HumanBlackboard.Get(Human.IsOnExit);
        }

        public override AbstractGoapAction GetResetCopy() {
            return new AggressiveApproximation(_human);
        }
    }
```

Abbildung 22 Action für die schrittweise Annäherung

Damit diese Klasse bei der Planung erneut und mit der Startbelegung der Zähler genutzt werden kann, wird die „GetResetCopy“ Methode hier überschrieben. Im Standard gibt diese die Referenz auf die Klasse selbst zurück und erzeugt keine neuen Objekte.

5.5.3 Visualisierung der Simulation

Um die Simulation im Verlauf der Entwicklung zu beobachten und später teilweise auch auszuwerten ist eine Visualisierung sinnvoll. Die Entscheidung ist auf eine einfache Windows Forms Oberfläche, zur Darstellung der Umwelt und Agenten, gefallen.

Ein Argument dafür ist, dass Mono[15] welches von den MARS Nutzern mit MacOS oder Linux häufig gebraucht wird, auch mit Windows Forms lauffähig ist. Die Daten der Visualisierung werden durch den Cell-Layer an das Forms Projekt SimPanViewer, welches in einem eigenen Thread läuft, weiter gegeben.

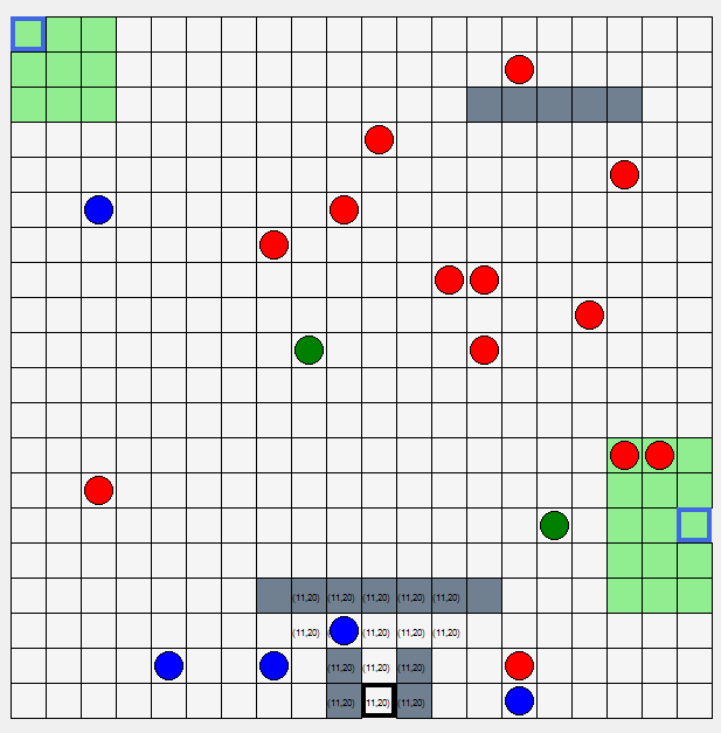


Abbildung 23 Darstellung der Simulation zu Beginn

Die Daten für die Visualisierung werden aus den Zellen und den Einstellungen des Cell-Layer gewonnen.

[15] http://www.mono-project.com/

6 Tests und Ergebnisse

In diesem Kapitel werden Tests in Bezug auf die GOAP-Komponente und Vergleiche zwischen der entwickelten Simulation und SimPan angestellt.

6.1 GOAP

Als Framework für die Tests wird NUnit[16] bereits für MARS eingesetzt und wurde auch weiter verwendet. Die Modelldefinitionen befinden sich in der Regel nicht direkt im MARS-System. Weil GOAP allerdings nicht ohne Actions usw. arbeiten kann, wurde ein spezielles Modell zu Testzwecken untergebracht.

Das GoapModelTest Projekt beinhaltet Actions, Goals, Symbole und verschiedene Konfigurationen. Kontext-Vorbedingungen und –Effekte sind in den Actions nicht benötigt. An diesem Modell können somit die Unit und Komponententests immer wieder ausgeführt werden.

Provozierte Fehlersituationen:

- Initialisierung der GOAP-Komponente und Auffinden der Konfigurationsklasse via Reflektion
- Initialisierung ohne Actions oder Goals
- Initialisierung mit doppelten Symbolschlüsseln in Actions
- Initialisierung mit doppelten Symbolschlüsseln in Goals
- Test der Überschreitung der maximalen Suchtiefe des Graphen
- Rückgabe der leeren Action wegen Überschreitung der Suchtiefe
- Test mit nicht erreichbarem Goal, weil keine zielführenden Actions vorhanden sind
- Prüfung des Wechsels des Goals anhand erwarteter Actions
- Prüfung auf Rückgabe der erwarteten Actions in der korrekten Reihenfolge

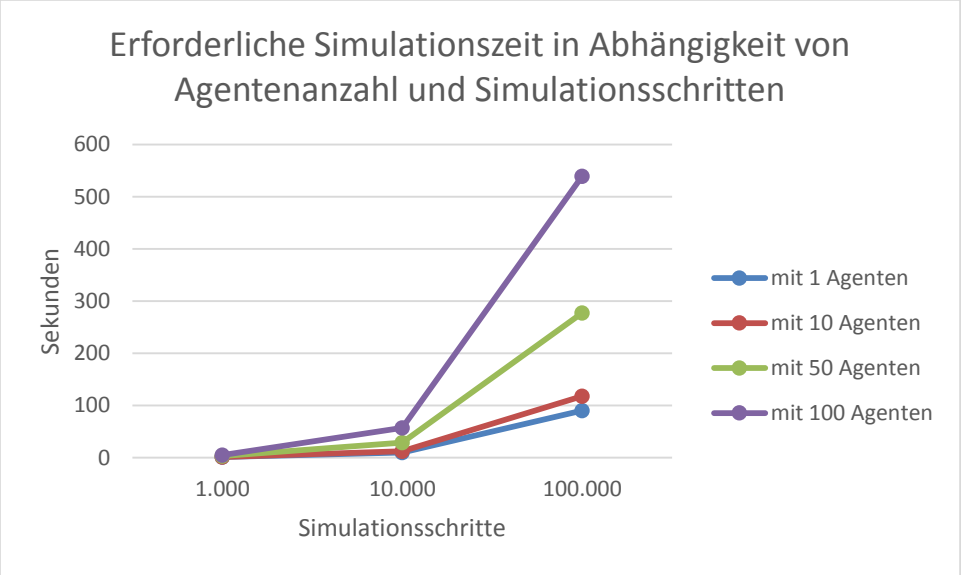
6.1.1 Performance Tests

Wie viel Rechenzeit benötigen x Agenten bei n Ticks, wenn sie Pläne durch die GOAP-Komponente erstellen lassen? Diese Zeiten sind stark von der Komplexität der

[16] http://www.nunit.org/

entstehenden Graphen bei der Planung abhängig. Weiteren Einfluss nimmt die durchschnittliche Länge eines Planes. Je länger die Pläne, desto seltener muss eine Neuplanung durchgeführt werden. Um eine Einschätzung von den benötigten Zeiten zu gewinnen wurde Agenten mit zwei Goals und drei Actions zur Planung erstellt.

Die nachfolgenden Zeitmessungen wurden mit einem AMD Phenom 9650 Quad-Core Prozessor mit 2,3 GHz und 8 GB Arbeitsspeicher durchgeführt. Die Ausführung wurde aus Visual Studio heraus gestartet.



Die Erstellung eines Planes ist immer mit Lastspitzen verbunden. Obwohl die Suche, über die Tiefe des Graphen, begrenzt werden kann, können starke Verzweigungen zu hohem Rechenaufwand führen. Um eine Einschätzung der Rechenzeiten zu erhalten, wurden Testszenarien mit starker Verzweigung erstellt.

Um eine starke Verzeigung zu erhalten, wurden Goals definiert, die Symbole mit booleschen Werten erwarten. Dazu passend wurden Actions formuliert, die jeweils nur ein gefordertes Symbol befriedigen. Ein Goal mit drei Symbolen und drei dazu passenden Actions ergibt Knoten mit drei möglichen ausgehenden Kanten und eine Höhe des resultierenden Baumes von vier. In der nächsten Grafik sind im Wurzelknoten ABC als Symbole gefordert. Die linke Kante repräsentiert immer die Anwendung der Action, welche A erfüllt, die mittlere B und die rechte C.

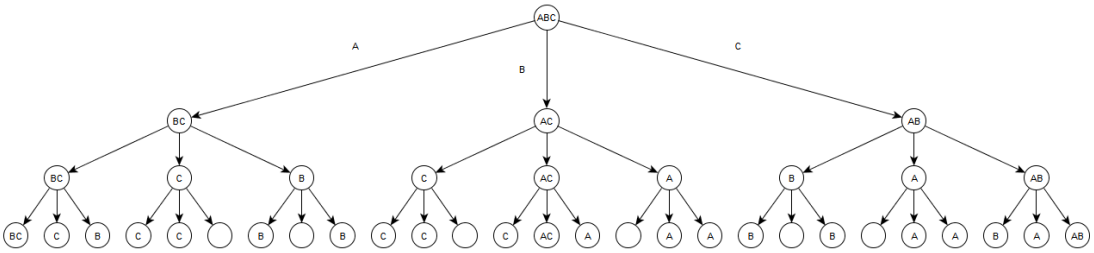


Abbildung 24 Die möglichen Verzweigungen.

Durch die optimierte Suche, die Vorauswahl der Actions und die Erkennung von gleichen Knoten wird der mögliche Graph stark reduziert und optimiert.

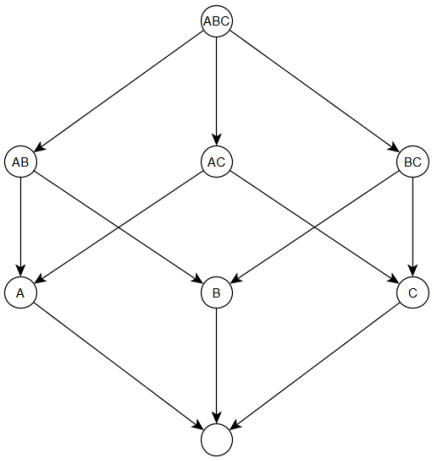
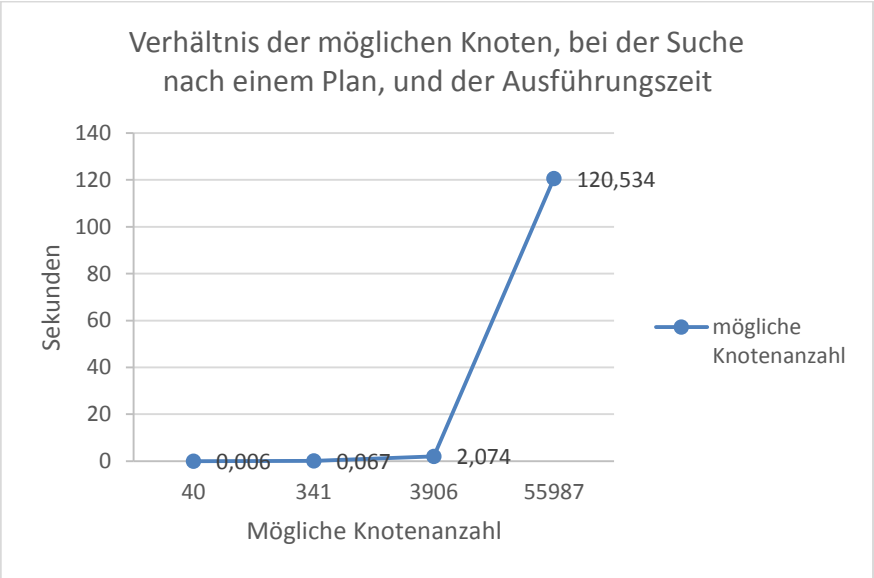


Abbildung 25 Der durch GOAP optimierte Graph.

Für die Zeitmessungen wurden Graphen mit unterschiedlich starker Verzweigung ausgewählt und getestet. Ab einer Verzweigung von mehr als sechs, waren die Ausführungszeiten bereits sehr hoch und wurden nicht mehr im Diagramm verwendet.

Die nachfolgenden Messungen wurden mit einem AMD Phenom 9650 Quad-Core Prozessor mit 2,3 GHz und 8 GB Arbeitsspeicher durchgeführt. Die Zeiten wurden durch Testdurchläufe mit NUnit innerhalb von Visual Studio gemessen.



6.2 GOAP-Komponente im MARS Kontext

Es existiert keine Abhängigkeit der GOAP-Komponente zum Mars-System. Erst in der Formulierung des Agenten werden diese Abhängigkeiten indirekt erzeugt. Die GOAP-Komponente kann theoretisch auch für die Umsetzung eines Spielcharakters genutzt werden. Ein Agent kann die GOAP Komponente nutzen, um Actions zu erhalten, durch die er sich steuern lassen kann.

6.3 Die Simulation anhand von SimPan

Da die Komplexität des Referenzmodells um ein vielfaches höher ist als die testweise Implementierung für die GOAP Komponente, sind direkte Vergleiche zwischen Werten der Testsimulation und den Funktionalitätstest zur Plausibilitätsbetrachtung des Referenzmodells nicht zielführend. Es werden daher einzelne Aspekte der Funktionalitätstests von SimPan mit der entwickelten Simulation vorgenommen.

Die Umgebung wurde, wie in den Funktionalitätstests von SimPan, durch ein Feld von 20 x 20 Zellen dargestellt. Begehbare und nicht begehbare (graue) Zellen wurden umgesetzt. Der Cell-Layer beinhaltet eine Konfiguration um unter anderem die Hinderniszellen anhand der Zellnummern festzulegen.

Das kritische Ereignis tritt sofort ein und tötet Agenten die sich auf den entstehenden Panik- oder Chaos-Zellen befinden. Diese getöteten Agenten werden hellblau dargestellt und können nicht mehr aktiv werden.

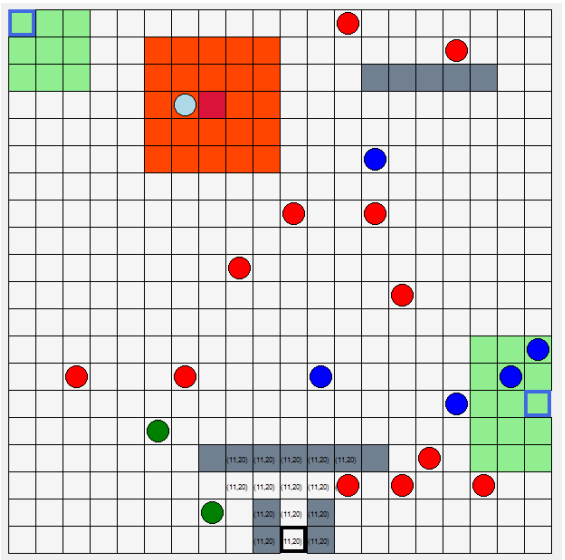


Abbildung 26 Das kritische Ereignis hat einen Agenten getötet

Die Anzahl der Agenten und ihr Startverhaltensmodus können im Human-Layer festgelegt werden. Agenten im reaktiven Modus werden rot, im deliberativen blau und im reflektiven grün gezeichnet. Die Agenten können nur neutrale und Verlustzellen betreten.

Die Entwicklung des Wertes der Angst wird in SimPan durch viele Einflüsse sehr individuell berechnet. In der Umgesetzten Simulation wurde dies stark vereinfacht. Interessant in Bezug auf GOAP ist vor allem der Wechsel zwischen den Verhaltensweisen und die damit verbundene Initialisierung des passenden GoapManager. Die Angst wird in jedem Simulationsschritt stetig verringert. Wenn eine Beruhigungssphäre auf Agenten einwirkt, wird die Angst stärker verringert.

Die Sphären der deliberativen und reflektiven Agenten konnten ähnlich denen in SimPan umgesetzt werden. Die Beruhigungssphäre wird in alle Richtungen um den Agenten gleichmäßig gesetzt. Die Sphäre der Massenfluchtkoordinaten wird nur hinter dem Agenten gesetzt, damit die beeinflussten Agenten diesem nicht den Weg versperren. Dazu wurde anhand der aktuellen und letzten Position die Laufrichtung ermittelt.

In der nachfolgenden Abbildung ist ein reflektiver Agent mit Beruhigungssphäre und den Massenfluchtkoordinaten gezeigt. Die Koordinaten verweisen auf die Position hinter dem Agenten und nicht auf den Ausgang zu dem er sich bewegt.

Deliberative Agenten können nur die Massenfluchtkoordinaten weitertragen. Sobald sie auf ihrem Weg blockiert werden, wird die Sphäre für zwei Simulationsschritte ausgesetzt um Blockierungen durch folgende Agenten zu vermeiden.

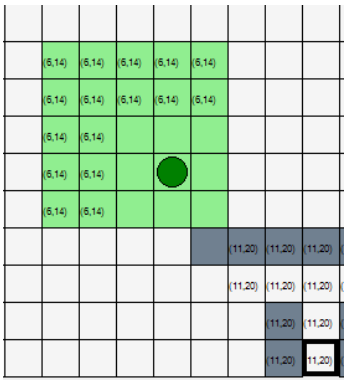


Abbildung 27 Reflektiver Agent mit Beruhigungs- und Massenfluchtsphäre

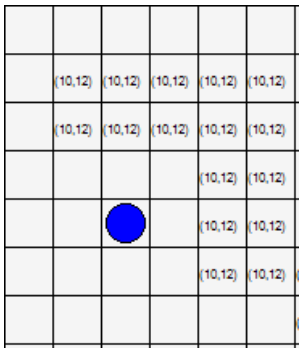


Abbildung 28 Massenfluchtsphäre bei diagonaler Bewegung

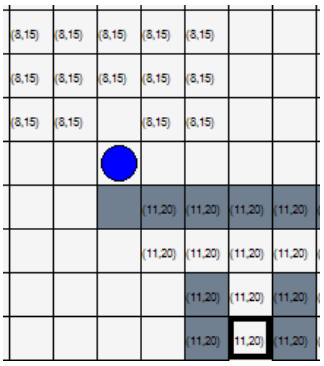


Abbildung 29 Massenfluchtsphäre bei vertikaler Bewegung

Die Funktion und Gestaltung dieser Sphären wurde ebenfalls vereinfacht. Bei SimPan ist z.B. ein Verblassen der Massenfluchtinformationen vorgesehen. Für die Nutzung mittels GOAP war interessanter, dass reaktive Agenten das ziellose Umherirren abbrechen und an der Massenflucht teilnehmen. Dazu mussten diese erneut planen.

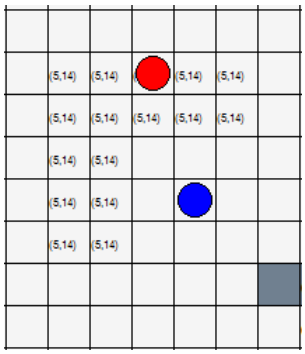


Abbildung 30 Reaktiver Agent nimmt an der Massenflucht teil

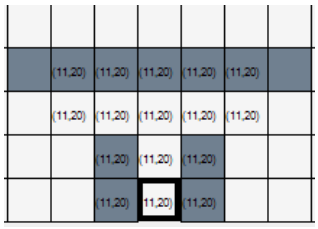


Abbildung 31 Ausgang und Ausgangsbereich

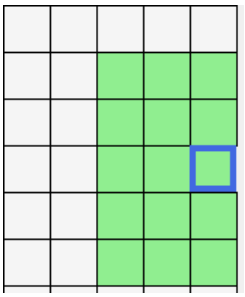


Abbildung 32 Technische Beruhigungsquelle

Ausgänge sind durch eine schwarze Umrandung der Zellen zu erkennen. Um den Ausgang der Simulation sind Hinderniszellen gesetzt und die Koordinaten des Ausgangs werden im Ausgangsbereich bereit gestellt. Ziellos umherirrende Agenten laufen somit zufällig in den Ausgangsbereich und können dann in die nächste Action wechseln.

Die technischen Informationsquellen aus SimPan wurden nur durch eine Beruhigungsquelle umgesetzt. Technische Informationsquellen sind durch eine blaue Umrandung der Zelle und den grünen Ausdehnungsbereich zu erkennen.

7 Diskussion und Ausblick

7.1 Rückblick

Das Konzept des Goal-Oriented Action Planning in eine konkrete und zudem noch möglichst allgemein gehaltene Komponente umzusetzen, gestaltete sich aufgrund fehlender Implementierungsdetails teils sehr aufwändig. Besonders die Identifizierung von möglichen Fehlern bei der Knotenbildung für den Suchgraphen hat viel Zeit in Anspruch genommen. Durch die prototypische Entwicklung von Arne Klingenberg (Klingenberg, 2010) konnten wichtige Orientierungspunkte für die GOAP Komponente gewonnen werden.

Das Ziel eine Komponente zu entwickeln, die sich in das MARS System einpasst und den Modellprogrammierern die grundlegende GOAP Funktionalität bereitstellt, konnte durch schlankes Design und Unabhängigkeit zu den bisherigen Komponenten umgesetzt werden. Die MARS Architektur wurde nicht beeinträchtigt. Der Anschluss an weitere MARS Systeme wird erst im Agenten hergestellt.

Bei der Entwicklung eines Szenarios ging es einerseits darum, den Entwicklungsprozess zukünftiger Nutzer der GOAP-Komponente vorzubereiten und Hilfen zu stellen. Andererseits wurden dadurch vielfältige Problemsituationen aufgedeckt und die deren Behandlung bereits in die GOAP-Komponente eingearbeitet werden konnten.

7.2 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung einer GOAP-Komponente für die Multiagentensimulationsplattform MARS, die im Rahmen eines Projektes an der Hochschule für angewandte Wissenschaften Hamburg entwickelt wurde. Dazu wurden zuerst die grundlegenden Begrifflichkeiten in Bezug auf Multiagentensimulationen, das Konzept von GOAP und das Referenzmodell SimPan geklärt.

Die Inferenzbildung von GOAP wurde ausführlich untersucht und noch offene Möglichkeiten für Fehler konnten identifiziert werden. Daraus wurde ein eigens entwickeltes Regelwerk zusammengestellt. Dieses wurde auf größtmögliche Toleranz und Sicherheit ausgelegt.

Das Referenzmodell SimPan wird anschließend detaillierter untersucht und erste Elemente für die testweise Simulation festgelegt. Der Verlauf der Entwicklung soll dem zukünftigen Modellprogrammierer als Hilfestellung für die Nutzung der GOAP-Komponente dienen.

In den Kapiteln Design und Realisierung wird besonders auf die Details der entwickelten Komponente eingegangen. Diese konnte sehr schlank und ohne Beeinträchtigungen der MARS Architektur entwickelt werden

Die Analyse und Umsetzung eines Szenarios anhand des Referenzmodells SimPan hat dazu beigetragen, die Kernfunktionalität von GOAP weiter zu isolieren und dem Modellprogrammierer Wege für eine Umsetzung aufzuzeigen.

7.3 Ausblick

Da der Freiheitsgrad des Modellprogrammierers in der ersten Variante von GOAP für MARS selbstverständlich hoch sein sollte, wären spezialisierte GOAP Abwandlungen, die dann in Abhängigkeiten zu LIFE Services stehen, gut vorstellbar. Hier könnten z.B. bestimmte Simulationsobjekte, die Interaktionen komfortabler gestalten, standardmäßig integriert werden.

Weiterhin ist auch die Realisierung eines speziellen GOAP-Agenten, der die C4 Architektur berücksichtigt, denkbar.

8 Literaturverzeichnis

Beierle, C., & Kern-Isberner, G. (2008). *Methoden wissensbasierter Systeme - Grundlagen, Algorithmen, Anwendungen*. Retrieved from http://www.springer.com/springer+vieweg/it/programmierung/book/978-3-8348-0504-1

Burke, R., Isla, D., Downie, M., Ivanov, Y., & Blumberg, B. (2001). Creature Smarts: The Art and Architecture of a Virtual Brain. In *Game Developers Conference* (p. 20). San Jose, CA. Retrieved from http://characters.media.mit.edu/additional resources/gdc01.pdf

Colledanchise, M., & Ögren, P. (2014). How Behavior Trees Modularize Robustness and Safety in Hybrid Systems. Retrieved from http://www.csc.kth.se/~miccol/Michele_Colledanchise/Publications_files/IROS14_CO.pdf

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3-4), 189–208. doi:10.1016/0004-3702(71)90010-5

Görz, G., Schneeberger, J., & Schmid, U. (2014). *Handbuch der künstlichen Intelligenz* (5th ed., p. 665). München: Oldenbourg Verlag. Retrieved from http://www.degruyter.com/view/product/228938?rskey=7FxVNd&result=2

Hüning, C. (2013). *Konzeption und Entwurf einer Architektur zum Einsatz von Multi-Agenten-Simulationen in der ökologischen Systemmodellierung*. Hochschule für Angewandte Wissenschaften Hamburg. Retrieved from http://edoc.sub.uni-hamburg.de/haw/volltexte/2013/2166/

Hüning, C., Wilmans, J., Feyerabend, N., & Thiel-Clemen, T. (2014). MARS - A next-gen multi-agent simulation framework. In *Simulation in Umwelt- und Geowissenschaften* (pp. 37–49). Osnabrück: Jochen Wittmann, Dimitris K. Maretis. Retrieved from http://mars-group.org/wp-content/uploads/2014/10/MARS-A-next-gen-simulation-framework.pdf

Klingenberg, A. (2010). *Prototypische Entwicklung eines emotionalen Agenten auf der Basis des Goal Oriented Action Plannings*. Hochschule für Angewandte Wissenschaften

Hamburg. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Prototypische+Entwicklung+eines+emotionalen+Agenten+auf+der+Basis+des+Goal+Oriented+Action+Plannings#0

Klügl, F. (2006). Multiagentensimulation. *Informatik-Spektrum*, *29*(6), 412–415. doi:10.1007/s00287-006-0115-7

Kolbe, F. (2013, September). *Goal Oriented Task Planning for Autonomous Service Robots*. *Mycotoxin research*. Hamburg University of Applied Sciences. Retrieved from http://users.informatik.haw-hamburg.de/~kolbe_j/thesis/RGOAP-Felix_Kolbe-Masterthesis-2013.pdf

Krumke, S. O., & Noltemeier, H. (2012). *Graphentheoretische Konzepte und Algorithmen* (3. Edition.). Wiesbaden: Vieweg+Teubner Verlag. doi:10.1007/978-3-8348-2264-2

Long, E. of A. D. (2007). *Enhanced NPC Behaviour using Goal Oriented Action Planning*. University of Abertay Dundee. Retrieved from http://www.edmundlong.com/Projects/Masters_EnhancedBehaviourGOAP_EddieLong.pdf

Maes, P. (1995). Artificial life meets entertainment: lifelike autonomous agents. *Communications of the ACM*, *38*(11), 108–114. doi:10.1145/219717.219808

Orkin, J. (2003). Applying Goal-Oriented Action Planning to Games. In *AI Game Programming Wisdom 2* (p. 11). Retrieved from http://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf

Orkin, J. (2004). Symbolic representation of game world state: Toward real-time planning in games. In *AAAI Workshop on Challenges in Game AI* (pp. 26–30). Retrieved from http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-006.pdf

Orkin, J. (2005). Agent Architecture Considerations for Real-Time Planning in Games. *Artificial Intelligence*, *38*, 6. Retrieved from http://alumni.media.mit.edu/~jorkin/aiide05OrkinJ.pdf

Orkin, J. (2006). Three States and a Plan: The A.I. of F.E.A.R. In *Game Developers Conference* (p. 18). Retrieved from http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

Schneider, B. (2011). *Die Simulation menschlichen Panikverhaltens*. Wiesbaden: Vieweg+Teubner. doi:10.1007/978-3-8348-8152-6

Siedersleben, J. (2003). Quasar: Die sd&m Standardarchitektur Teil 1. *Auflage, München*, 28. Retrieved from http://www.fbi.h-da.de/fileadmin/personal/b.humm/Publikationen/Siedersleben_-_Quasar_2__sd_m_Brosch_re_.pdf

Siedersleben, J. (2004). *Moderne Softwarearchitektur: umsichtig planen, robust bauen mit Quasar*. (J. Siedersleben, Ed.) (1. ed., p. 282). dpunkt-Verlag. Retrieved from http://books.google.de/books/about/Moderne_Software_Architektur.html?id=ZYKlAQAACAAJ&pgis=1

The 2004 AIISC Report - Working Group on Goal-oriented Action Planning. (2004). Retrieved June 17, 2014, from http://archives.igda.org/ai/report-2004/goap.html

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____