Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Arne Klingenberg

Prototypische Entwicklung eines emotionalen Agenten auf der Basis des Goal Oriented Action Plannings

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Arne Klingenberg

Prototypische Entwicklung eines emotionalen Agenten auf der Basis des Goal Oriented Action Plannings

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thiel-Clemen
Zweitgutachter : Prof. Dr. Zukunft

Abgegeben am 24. Februar 2010

Arne Klingenberg

Thema der Bachelorarbeit

Prototypische Entwicklung eines emotionalen Agenten auf der Basis des Goal Oriented Action Plannings

Stichworte

Emotionale Agenten, GOAP, OCC-Modell, Künstliche Intelligenz, Simulation, Planning, Agenten Systeme

Kurzzusammenfassung

Diese Arbeit beschreit die prototypische Entwicklung eines emotionalen Agenten auf der Basis des Goal Oriented Action Plannings. Es werden grundlegenden Konzepte für die Entwicklung eines emotionalen Agenten erläutert und anhand einer Analyse aufgezeigt, welche Anforderungen an einen solchen Agenten gestellt werden. Anhand der Anforderungen, die die Analyse ergab, wurde ein Design für das Agentensystem erstellt und realisiert. Um den entwickelten Prototpyen zu validieren und die Auswirkungen der Emotionen auf dessen Verhalten zu beurteilen, wurde schließlich eine Testreihe in einem Makrtplatzscenarion durchgeführt

Arne Klingenberg

Title of the paper

Prototypical development of an emotional agent based on Goal Oriented Action Planning

Keywords

Emotional Agents, GOAP, OCC-Model, Artificial Intelligence, Simulation, Planning, Agent Systems

Abstract

This paper describes the prototypical development of an emotional agent based on Goal Oriented Action Planning. Firstly the fundamental concepts, which form the basis for the development of such an Agent, are described. Following an analysis has been performed to identify the requirements for an emotional agent and what is needed to fulfill them. On the basis of these requirements a design an implementation of the agent system has been developed. Finally a series of tests have been performed in a market scenario, to validate the developed prototype and to evaluate the impact of emotions on the agent's behavior.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Da Computerspiele heutzutage nicht mehr nur als Zeitvertreib für Kinder gelten, sondern auch in zunehmendem Maße von Erwachsenen gespielt und für ernsthafte Simulationen genutzt werden, steigt der Anspruch an die Künstliche Intelligenz (KI) in solchen Programmen immer weiter an. Dies führt dazu, dass herkömmliche Methoden, wie Zustandsautomaten (FSM), immer mehr an ihre Grenzen stoßen. Besonders problematisch ist dabei, dass sie mit zunehmender Komplexität immer unüberschaubarer und somit fehleranfälliger werden.
Da jedoch auch eine sehr komplexe KI häufig unnatürlich wirkt, sollten darüber hinaus Methoden geschaffen werden, um eine größere Bandbreite von Verhalten zu erzeugen. Eine Möglichkeit glaubhafte Varianz in das Verhalten einer KI zu bringen besteht darin, die Entscheidungen der KI durch eine Filterfunktion zu beeinflussen. Hierbei werden Emotionen oft vernachlässigt, obwohl sie eine wichtige Rolle in sozialen Umgebungen spielen [GM05].

1.2 Zielsetzung

In dieser Arbeit wird ein Prototyp entwickelt, der die oben angesprochenen Probleme beseitigt. Spiele wie World of Warcraft zeigen, dass manche Entwickler begonnen haben Emotionen in Agenten zu integrieren. Jedoch wurden Emotionen in Spielen wie World of Warcraft hauptsächlich durch die Animation von Gesichtsausdrücken realisiert. Diese Art der Einbindung von Emotionen trägt zwar zur Glaubwürdigkeit der Agenten bei, verändert jedoch nicht deren sich wiederholendes Verhalten [SS07].
Diese Arbeit wählt einen anderen Ansatz, indem die Emotionen intern verwendet werden, um die Entscheidungen der Agenten und somit deren Verhalten zu beeinflussen. Hierzu wird ein relativ neues KI Verfahren namens Goal Oriented Action Planning (GOAP) verwendet und mit einem Emotionsmodell kombiniert.
Die Motivation in der Kombination dieser beiden Modelle liegt darin, emotionale Agenten zu erschaffen, die von ihren Emotionen in Entscheidungen beeinflusst werden und dadurch Spiele und Simulationen lebensechter und glaubwürdiger erscheinen lassen.

1.3 Abgrenzung

Die Entwicklung einer komplexen künstlichen Intelligenz ist eine Aufgabe, die mehrere Jahre in Anspruch nehmen kann. Es muss eine Vielzahl verschiedener Verhalten entwickelt, implementiert und vor allen Dingen lange getestet werden.
Aufgrund dieser Tatsache und der begrenzten Zeit, die für die Erstellung dieser Arbeit zur Verfügung steht, wird wie bereits erwähnt, nur ein Prototyp entwickelt. Dieser Prototyp dient als proof of concept des entwickelten Designs und implementiert nur ein paar grundlegende Verhalten.

1.4 Inhaltlicher Aufbau

Die Arbeit beginnt mit einem Kapitel über die Grundlagen, die zum Verständnis der Arbeit vorausgesetzt werden. Hierzu werden die Konzepte und Techniken vorgestellt, die den Stand der Technik in dem jeweiligen Bereich darstellen und somit als Ausgangspunkt für diese Arbeit dienen.
Nachdem die Grundlagen erläutert wurden, wird eine Analyse durchgeführt, anhand derer gezeigt wird, warum ein neues Verfahren für die Entwicklung von emotionalen Agenten notwendig ist. Des Weiteren wird in dieser Analyse herausgearbeitet, welche Anforderungen an den Prototypen und die Testumgebung gestellt werden.
Anhand der gefundenen Anforderungen wird im darauf folgenden Kapitel das Design des Prototypen und der Testumgebung erstellt.
Im zweiten Teil der Arbeit wird genauer beschrieben, wie die im Analyseteil betrachteten Verfahren in der Realisierung des Prototypen umgesetzt wurden. Besonderes Augenmerk liegt hierbei auf den verschiedenen Komponenten des Planning Systems. Außerdem werden in diesem Abschnitt die Implementationsdetails der Testumgebung erläutert.
Anschließend werden die Ergebnisse betrachtet, die bei der Ausführung des Testscenarios unter verschiedenen emotionalen Zuständen gesammelt wurden. Den Abschluss der Arbeit stellt der Diskussions-Teil dar, in dem die gewonnen Ergebnisse eingeordnet werden und ein Ausblick auf mögliche Erweiterungen des Systems gegeben wird.

2 Methodische Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte für die Entwicklung des Prototypen erläutert. Dazu wird zunächst veranschaulicht was in der Informatik unter einem Agenten zu versteht ist und anschließend eine Architektur zu deren Entwicklung vorgestellt.
Danach wird ein kurzer Blick auf den A* Algorithmus geworfen, der sowohl in der Wegfindung als auch dem Planner (siehe 2.3.2) des Prototypen eine zentrale Rolle spielt.
Des Weiteren wird erläutert weshalb es für eine KI wichtig ist Entscheidungen zu treffen, die für den Betrachter glaubwürdig erscheinen und welche Methoden es gibt um dies zu erreichen.
Der letzte Teil dieses Kapitels beschäftigt sich damit, wie Emotionen in einem Agenten modelliert werden können. Hierzu wird zunächst darauf eingegangen, was unter Emotionen zu verstehen ist und welche Arten von Emotionen es gibt. Anschließend werden zwei gängige Methoden zur Modellierung von Emotionen vorgestellt.

2.1 Agenten

Es gibt viele verschiedene Arten von Agenten in der Informatik und dementsprechend ist es schwer, eine eindeutige Definition für diese zu finden. So ist bei Nwana zu lesen:
"We have as much chance of agreeing on a consensus definition for the word 'agent' as AI researchers have of arriving at one for 'artificial intelligence' itself - nil!"[NH96]
Jedoch werden in dieser Arbeit Eigenschaften genannt, die ein Software Agent zu einem gewissen Grad haben sollte. Laut Nwana sind dies folgende:

1. **Autonomy:** Beschreibt die Eigenschaft, dass Agenten eigenständig operieren und ihre Ziele verfolgen können.
2. **Cooperation:** Besagt, dass Agenten mit anderen Agenten und ggf. auch mit Menschen über eine Art von Sprache kommunizieren können.
3. **Learn:** Sagt aus, dass Agenten in einem bestimmten Maß durch Interaktion mit ihrer Umgebung lernen können.

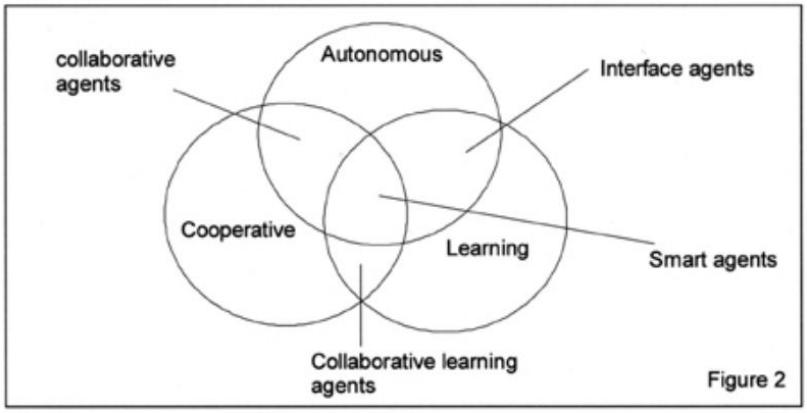


Abbildung 2.1: Nwanas Kategorisierung der verschiedenen Agenten [NH96]

Aus diesen Eigenschaften leitet Nwana vier Agententypen ab (siehe Abbildung 2.1).

Aber auch Nwanas Kategorisierung von Agenten deckt nicht alle Arten von Agenten ab. So gibt es beispielsweise noch die sogenannten Believable Agents, die bei Rizzo so characterisiert werden: „agents are considered „believable“ when viewd by an audience as endowed with behaviors, attitudes and emotions, typicall of different personalities.“ [PMCA99].

Da in dieser Arbeit ein emotionaler Agent entwickelt wird, ist im folgenden Verlauf der Arbeit mit dem Begriff Agent eine Mischung aus Nwanas Smart Agent und einem Believable Agent gemeint.

2.1.1 C4 Architektur

Das C4 Modell (siehe Abbildung 2.2) ist eine vom MIT Media Lab entwickelte Architektur zur Entwicklung von KIs. Das Design der Architektur orientiert sich an biologischen Systemen, wie etwa Hunden, da diese von der Natur perfekt für ihre jeweiligen Aufgaben angepasst wurden.[RB01].
Ziel des Modells ist es, ein virtuelles Gehirn nachzubilden, indem komplexe Vorgänge in kleinere Einheiten unterteilt werden, wie es in menschlichen und tierischen Gehirnen auch geschieht. Das Modell setzt sich aus mehreren Hauptkomponenten zusammen, die über ein internes Blackboard miteinander kommunizieren können.
Folgende Hauptkomponenten sind in dem Modell vertreten:

1. Das Sensor System, welches dafür zuständig ist, die Umwelt wahrzunehmen. Es stellt den einzigen Eingangspunkt für Informationen in das System dar, und leitet diese ungefiltert an das Perception System weiter. Neben den Umwelteinflüssen kann das Sen-

sor System auch noch Reize wahrnehmen, die vom Proprioceptive System kommen und Veränderungen innerhalb des Systems darstellen.

2. Das Perception System ist dafür zuständig, die aufgenommenen Umwelteindrücke zu klassifizieren und schließlich an das Working Memory weiterzuleiten. Die Klassifizierung der Umwelteindrücke ist wichtig, damit keine überflüssigen Daten an das Working Memory weitergeleitet werden. So würde ein Schaf zum Beispiel auch die Konversation von zwei Menschen hören, diese Information jedoch nicht einordnen können, das Bellen eines Schäferhundes jedoch schon [RB01].

3. Das Working Memory speichert die klassifizierten Daten. Es repräsentiert also was der Agent aktuell über seine Umgebung weiß. Dieses Wissen ist jedoch nicht mit dem tatsächlichen Zustand der Umgebung gleich zu setzten, da sich diese natürlich verändern kann.

4. Das Action System ist dafür zuständig, anhand der im Working Memory befindlichen Erinnerungen Entscheidungen zu treffen. Zusätzlich zum Working Memory können auch noch Informationen vom Blackboard mit einbezogen werden. Die getroffene Entscheidung wird anschließend auf dem Blackboard bereitgestellt.

5. Das letzte System besteht aus dem Navigation und Motor System. Diese beiden Systeme sind dafür zuständig, die auf dem Blackboard vorliegenden Informationen in Bewegungen umzuwandeln, die nach außen hin sichtbar sind.

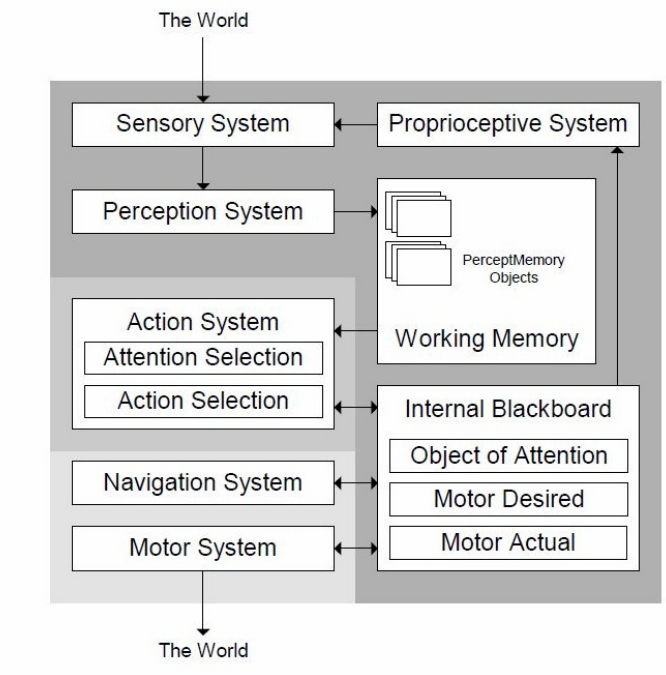


Abbildung 2.2: Architektur des C4 Modells [RB01]

2.2 Künstliche Intelligenz und Entscheidungen

Um die in den folgenden Kapiteln vorgestellten Methoden zur Modellierung von Künstlicher Intelligenz zu verstehen, sollte zunächst einmal geklärt werden, was unter dem Begriff Künstliche Intelligenz zu verstehen ist. John McCarthy von der Stanford University beschreibt Künstliche Intelligenz als die Wissenschaft und Entwicklung von intelligenten Maschinen, insbesondere von Computer-Programmen [McC].
Da Intelligenz jedoch eine komplexes Konstrukt aus einer Vielzahl kognitiver Fähigkeiten ist [WP09], beschränken sich die meisten KIs auf die Nachbildung eines bestimmten Teilbereiches. So beschäftigt sich das GALE Project etwa ausschließlich mit der Erkennung von Sprachen und Stimmen [Coh07].
Bei der Entwicklung von KIs für Computerspiele und Simulationen besteht das Ziel darin, Algorithmen zu entwerfen, die Agenten menschlicher erscheinen lassen [Mil06]. Dies bedeutet hauptsächlich eine KI zu entwickeln, die Entscheidungen trifft, welche für den Betrachter glaubwürdig erscheinen.
Entscheidungen werden hierbei in zwei Hauptkategorien unterteilt. Zum einen gibt es die normativen Entscheidungen, die anhand der gegebenen Fakten entscheiden was in einer Situation getan werden sollte.
Ein Beispiel hierfür ist der A* Algorithmus, der den besten Weg findet, den die KI nehmen sollte. Zum anderen gibt es die deskriptiven Entscheidungen, die anhand einer Beziehung zwischen Ursache und Wirkung getroffen werden und beschreiben, mit welcher Tendenz Individuen zu einer Entscheidung neigen [Mar09]. Da in dieser Arbeit eine KI für Spiele und Simulationen entwickelt wird, liegt das Hauptaugenmerk bei der Entwicklung der künstlichen Intelligenz auf der Entscheidungsfindung.

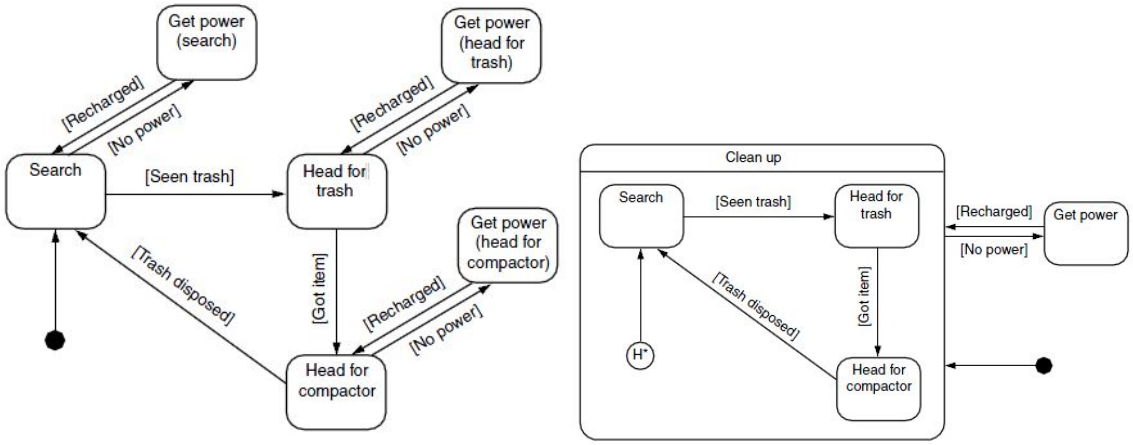
2.3 Bestehende Techniken zur Entscheidungsfindung

In diesem Abschnitt werden Techniken betrachtet, die bestimmen wie eine künstliche Intelligenz in verschiedenen Situationen reagieren soll. Dabei wird zunächst ein Blick auf statische Verfahren geworfen, die deterministisch festlegen, wie das Verhalten eines Agenten aussieht. Danach werden Techniken vorgestellt, die es Agenten erlauben, zur Laufzeit eines Programms eigenständig Entscheidungen zu treffen.

2.3.1 Finite State Machines

Finite State Machines, ((dt.)Zustandsautomaten) (FSM)s sind die am weitesten verbreitete Technik zur Entwicklung von KIs in Computerspielen [MT10]. Ein Zustandsautomat besteht

aus Zuständen und Übergängen, die von einem Zustand in den nächsten führen.
Jeder Zustand des Automaten enthält Anweisungen was der Agent tun soll, solange er sich in diesem Zustand befindet. Die in diesem Zustand enthaltenen Anweisungen werden so lange ausgeführt, bis bestimmte Bedingungen erfüllt sind, die dazu führen, dass ein Übergang in einen anderen Zustand stattfindet. So könnte es zum Beispiel einen Zustand *Umherwandern* geben, in dem der Agent kontinuirlich seine Position ändert, bis er etwas zu Essen findet, was den Übergang in den Zustand *Essen* veranlasst.
Bei herkömmlichen FSMs muss jeder Übergang zwischen zwei Zuständen explizit angegeben werden. Dies führt jedoch zu Problemen, wenn Verhalten modelliert werden sollen, in denen Zustände priorisiert werden sollen. Wenn ein Agent zum Beispiel die Flucht ergreifen soll, sobald Gefahr droht, egal in welchem Zustand er sich gerade befindet, müsste aus jedem Zustand explizit eine Transition zu dem Flucht Zustand angegeben werden. Da dies sehr aufwändig und fehleranfällig ist, wurden sogenannte Hierarchical Finite State Machines, ((dt.)Hirarchische Zustandsautomaten) (HFSM)s entwickelt.
In diesen HFSMs wird das Verhalten nicht in einem großen Zustandsautomaten beschrieben, sondern in mehrere kleine FSMs unterteilt. Diese sind in einer Hirarchie angeordnet, so dass eine in der Hirarchie tiefer liegende FSM nur dann angesprochen wird, wenn keine der höher liegenden FSMs aktiviert wurde [Mil06].



(a) FSM (b) HFSM

Abbildung 2.3: Unterschied zwischen FSM und HFSM [Mil06]

2.3.2 Goal Oriented Behavior

Agenten, deren Verhalten mit den zuvor betrachteten FSMs realisiert wird, haben keine Ziele, auf die sie hin arbeiten. Es wird lediglich anhand einer Reihe von Variablen ein Zustand ausgewählt, der ein bestimmtes Verhalten hervorruft.

Das Goal-Oriented Behavior (GOB) im Gegensatz definiert eine Reihe von Zielen, die das Verhalten der KI bestimmen. GOB ist jedoch keine eigenständige Technik, sondern stellt einen Oberbegriff für alle KI-Techniken dar, deren Verhalten durch das Streben nach der Erfüllung eines Ziels motiviert ist. Unter diesen Oberbegriff fallen daher auch die nachfolgend erklärten Techniken.

2.3.2.1 Planning

„Planning is a key area in Artificial Intelligence. In its general form, planning is concerned with the automatic synthesis of action strategies (plans) from a description of actions, sensors and goals. Planning thus contradicts with two other approaches to intelligent behavior: the programming approach, where action stragies are defined by hand, and the learning approach, where action strategies are inferred from experience.“[Gef02]

Die Suche nach einer Lösung für ein Problem mit Hilfe eines Planners geschieht in den meisten Fällen folgendermaßen:
Zunächst wird dem Planner der aktuelle Zustand der Welt übergeben. Des Weiteren erhält der Planner dazu die Aktionen, die ihm zur Verfügung stehen und den Zustand der Welt wie er sein soll, nachdem das Problem gelöst wurde (Goal State).
Aufgrund dieser Eingaben generiert der Planner aus den zur Verfügung stehenden Aktionen eine Sequenz von Aktionen, die den jetzigen Zustand der Welt in den gewünschten Zustand überführen. Abbildung 2.4 zeigt beispielhaft, wie eine gültige Sequenz von Aktionen aussehen könnte, um einen Gegner anzugreifen.

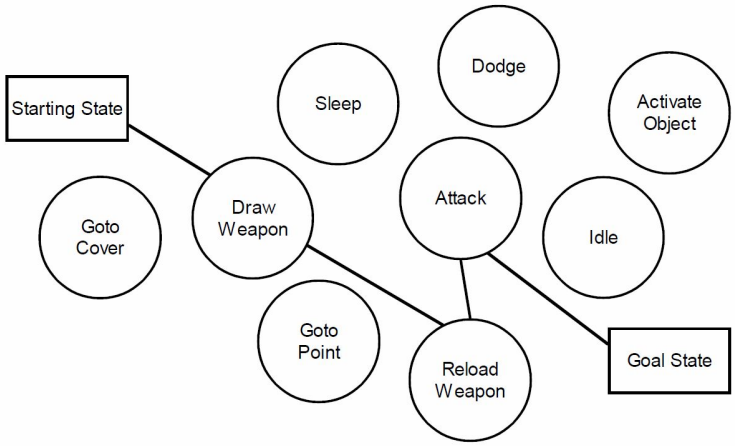


Abbildung 2.4: Beispielhafter Planungsprozess [Ork03]

Bei der Suche nach einer gültigen Sequenz kann entweder ausgehend vom aktuellen Zustand (Progressive Suche) zum Goal State oder umgekehrt gesucht werden (Regressive Suche).

2.3.2.2 GOAP

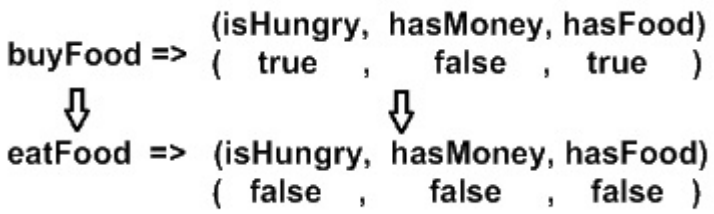
Das Goal Oriented Action Planning oder kurz GOAP ist eine von Jeff Orkin modifizierte Form des Stanford Research Institute Problem Solver (STRIPS) [FN71]. Wie in Kapitel 2.3.2.1 beschrieben, funktioniert auch der GOAP Planner in dem er aus einer Reihe von Aktionen einen Plan formuliert, der den aktuellen Zustand der Welt in einen Zielzustand überführt.
Dieser Plan bestimmt das Verhalten der KI und wird so lange ausgeführt bis das Ziel erreicht wurde, ungültig geworden ist oder ein anderes Ziel eine höhere Priorität erhält.
Anders als in Kapitel 2.3.1 werden Zustände im GOAP System jedoch durch einen Satz von Variablen repräsentiert und nicht von kompletten Codeblöcken, die bestimmte Aktionen ausführen.
So würde der Zustand eines Agenten, in dem er gelangweilt und hungrig ist, einfach durch die Kombination von den Variablen

$$isBored \wedge isHungry$$

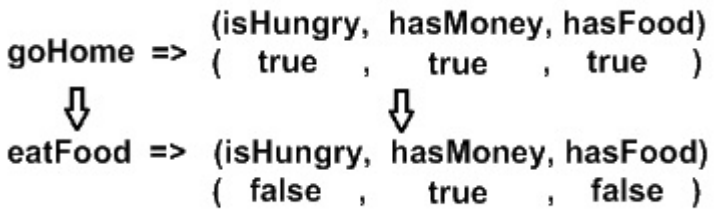
dargestellt werden.
Zwei Ziele dieses Agenten könnten also sein, den Hunger des Agenten zu stillen und gegen seine Langeweile etwas Unterhaltung zu suchen. Für welche dieser beiden Ziele zuerst ein Plan formuliert wird, wird dadurch entschieden, wie wichtig das jeweilige Ziel im Augenblick ist. So würde ein Agent, der seit Tagen nichts mehr gegessen hat sicher das Ziel den Hunger zu stillen priorisieren. Die Funktionsweise von GOAP wird am besten durch ein Beispiel erläutert.
Gehen wir davon aus, dass der Agent sich gerade auf einem Markt befindet und lange nichts gegessen hat, somit sein primäres Ziel *getFood* ist. Um seinen Hunger zu stillen kann der Agent entweder zu einer Bude gehen und sich dort etwas zu essen kaufen, oder er kann sich entscheiden nach Hause zu gehen und dort etwas zu essen. Sich an einer Bude etwas zu Essen zu kaufen ist nur dann eine Option, wenn der Agent genügend Geld hat. Der aktuelle Zustand des Agenten und der gewünschte Zielzustand könnten demnach wie folgt repräsentiert werden:

```
Symbol : ( isHungry, hasMoney, hasFood)
Current : (  true   ,   true   , false  )
Goal    : (  false  ,    -     ,   -    )
```

Ausgehend von diesem aktuellen Zustand und dem gewünschten Zielzustand könnte der GOAP Planner nun folgende Pläne erstellen:



oder



Wie in dem Beispiel zu sehen, wird der Zustand der Welt in GOAP durch eine Menge von sogenannten WorldStateSymbols repräsentiert. Die WorldStateSymbols bestehen aus einem Typ (z.B isHungry) und einem zugehörigen Wert. Um das Beispiel möglichst einfach zu halten wurden hier nur boolean Variablen verwendet. Es können jedoch beliebige Datentypen zur Repräsentation der Welt genutzt werden.
Dies ist natürlich ein triviales Beispiel, jedoch kann in richtigen Anwendungen die Zahl an vorhandenen Aktionen und die sich daraus ergebenden Möglichkeiten sehr viel größer sein. Wenn der Agent zum Beispiel gerade kein Geld hat, könnte er vorher noch zur Bank gehen, oder er entscheidet sich Geld oder Essen zu stehlen.
Der große Unterschied von GOAP zu STRIPS ist, dass GOAP in der Planungsphase auf A* zurückgreift, um für Echtzeitanwendungen einsetzbar zu sein. Hierbei repräsentieren die Worldstates die Knoten in dem Graphen und die Aktionen bilden die Übergänge zwischen den Knoten.
Der Vorteil hierbei ist, dass man anhand von Heuristiken die Suche in bestimmte Richtungen lenken kann. So würde einem Agenten, der einen normalen Marktbesucher darstellt, die Aktion das Essen zu stehlen auch zur Verfügung stehen, dies aber sicherlich keine bevorzugte Option sein. Um den Planner dazu zu bewegen die Aktion Kaufen zu bevorzugen, könnten die Kosten für diese Aktion im A* Algorithmus als sehr viel niedriger als die Kosten für das Stehlen angegeben werden. Für einen Dieb hingegen könnte dies jedoch ganz anders aussehen.
GOAP arbeitet bei der Suche nach einem gültigen Plan vom Zielzustand aus regressiv (siehe 2.3.2.1) bis zum momentan Ausgangszustand. Dabei durchsucht der Planner alle zur Verfügung stehenden Aktionen, um diejenigen zu finden, deren Effekte die Erfüllung des Ziels zur Folge haben.

Ein weiterer Unterschied zwischen STRIPS und GOAP sind die Procedural Preconditions. Da es unpraktisch wäre jeden Aspekt der Umgebung als Variable darzustellen, bietet diese Funktion die Möglichkeit bei Bedarf noch zusätzliche Vorbedingungen zu überprüfen, die nicht durch WorldStateSymbols repräsentiert werden. Zum Beispiel wäre die Option an einer Bude etwas zu Essen zu kaufen nur dann gültig, wenn der Agent überhaupt weiß, wo auf dem Markt sich eine Essensbude befindet.
Ähnlich wie die Procedural Preconditions gibt es in GOAP auch Procedural Effects. Diese Effekte verändern den Zustand der Welt nicht sofort, sondern benötigen dafür einen längeren Zeitraum. So braucht der Agent z.B. eine gewisse Zeit, um sein Essen zu essen und ihn somit zu sättigen. Dieser Effekt wäre mit STRIPS nicht möglich, da dort alle Effekte sofort eintreten [Ork06].
Dieses simple Beispiel zeigt schon, dass in einem GOAP System die KIs nicht mehr nur entscheiden können, was sie machen, sondern auch wie sie etwas machen. Dies führt dazu, dass das Verhalten der KI sich nicht so häufig wiederholt und einmal entwickelte Verhalten wiederverwendet werden können [Ork03].
GOAP wurde das erste mal in dem von Monolith productions entwickelten First Person Shooter F.E.A.R. eingesetzt. Inzwischen findet es jedoch in immer mehr Computerspielen Einsatz (siehe [Ork]) und wurde auch für militärische Zwecke, wie z.B der Routenplanung für Raketen, genutzt (siehe [Dor07]).

2.4 Emotionen

Da eine enge Verknüpfung von Emotionen und der Wahrnehmung besteht, ist es sehr schwer Emotionen objektiv zu beschreiben und ein Modell für diese zu erstellen. Aus diesem Grund gibt es auch viele verschiedene Emotionsmodelle, die sich teilweise stark unterscheiden[MG08]. Um für den weiteren Verlauf dennoch ein Vorstellung dafür zu vermitteln, was unter Emotionen zu verstehen ist, wird der Begriff Emotionen in diesem Kapitel zunächst genauer erläutert.
Nachdem der Emotions-Begriff erläutert wurde, wird anschließend vorgestellt, mit welchen Methoden Emotionen in der Informatik nachgebildet werden.

2.4.1 Was sind Emotionen

Fast jeder kann sich unter dem Begriff Emotionen etwas vorstellen, jedoch gestaltet es sich schwer diese Vorstellung klar zu definieren. So ist etwa im Lehrbuch der Emotionspsychologie folgendes zu lesen: „Bislang ist kein Konsens festzustellen, was man unter einer Emotion zu verstehen hat“ [SA96].
Um den Begriff Emotionen etwas einzugrenzen, sei hier dennoch eine Definition erwähnt.

Laut Kleinginna und Kleinginna ist eine Emotion „ein komplexes Muster von Veränderungen, das physiologische Erregung, Gefühle, kognitive Prozesse und Verhaltensweisen einschließt, die in Reaktion auf eine Situation auftreten, welche ein Individuum als persönlich bedeutsam wahrgenommen hat“ [PGZ08].

Genau wie bei der Frage wie Emotionen zu definieren sind, gibt es auch über die Anzahl der Emotionen, die ein Mensch empfinden kann, viele verschiedene Ansichten. Die Anzahl der verschiedenen Emotionen in den unterschiedlichen Modellen reicht dabei von zwei bis hin zu unendlich [OT90].
Allerdings ist eine weit verbreitete Ansicht, dass es zwei Kategorien von Emotionen gibt [SS07]:

Kategorie Eins: Emotionen, die sich kurz nach der Geburt entwickeln. Hierzu gehören: Wut, Ekel, Angst, Freude, Trauer und Überraschung.

Kategorie Zwei: Die Emotionen in der zweiten Kategorie werden als erlernte, soziale Emotionen angesehen. Hiervon gibt es eine große Bandbreite, zu denen etwa folgende Emotionen gehören: Verachtung, Peinlichkeit, Schuld, Neid, Stolz, Reue und Scham.

Allerdings werden in manchen Theorien die Emotionen aus Kategorie Zwei nicht als eigene Emotionen angesehen, sondern als Vermischung der Emotionen aus Kategorie Eins [SS07]. So kann Lieben zum Beispiel als Kombination von Freude und Vertrauen betrachtet werden. Dies wird auch in *Plutchik's Wheel of Emotions* dargestellt (siehe Abbildung 2.5).

2.4.1.1 Unterschied Stimmungen und Emotionen

Stimmungen werden häufig mit Emotionen verwechselt, aber der Unterschied liegt darin, dass Emotionen nur für eine kurze Zeit anhalten. Emotionen sind nur lang genug, damit ein Individuum auf diesen Impuls reagieren kann und eine Art von korrigierender Maßnahme einleiten kann.
Stimmungen hingegen dauern sehr viel länger an, manchmal sogar bis zu ein paar Tagen. Meist liegt einer Stimmung eine Emotion zugrunde. Wenn ein Individuum in einer bestimmten Stimmung ist, können ähnliche Emotionen schneller ausgelöst werden.
Zum Beispiel neigt eine Person, die in einer schlechten Stimmung ist, eher dazu eine negative Emotion wie etwa Wut zu empfinden[PMCA99].

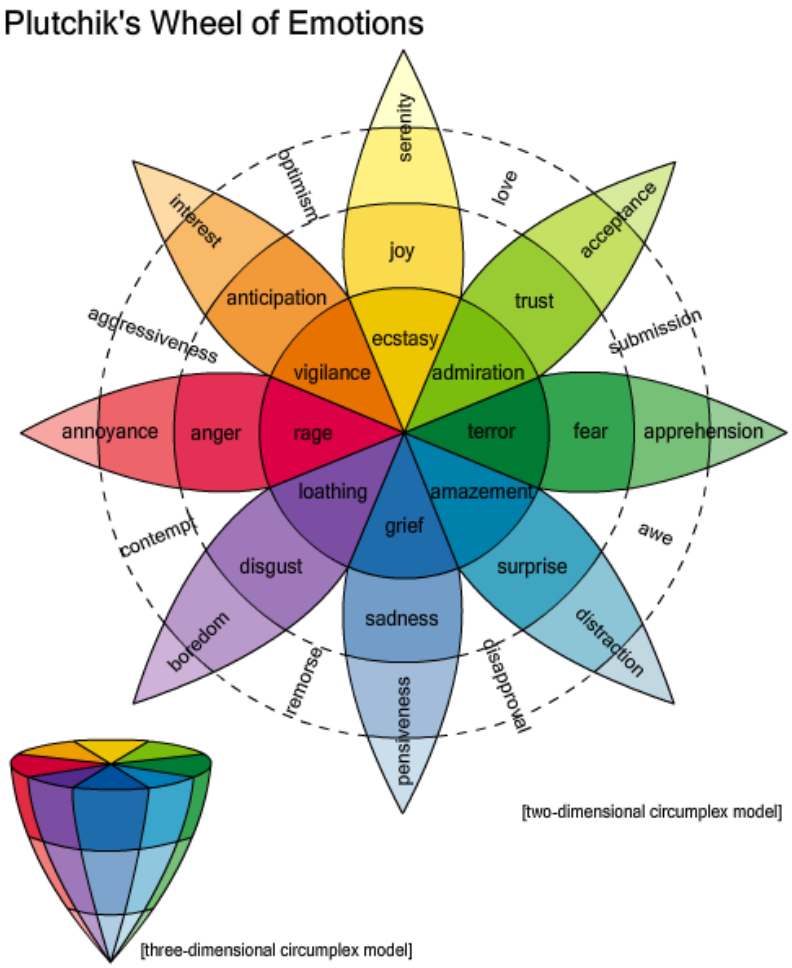


Abbildung 2.5: Plutchik's Wheel of Emotions [Plu]

2.4.2 OCC

Obwohl es in der Psychologie, wie zuvor beschrieben, viele verschiedene Emotionsmodelle gibt, hat sich in der Informatik das OCC (Ortony, Clore, & Collins, 1988) Modell [OCC88] als das Standardmodell für die Emotionssynthese etabliert und wurde bereits von einer großen Anzahl an Studien genutzt [Bar02]. Im OCC Modell werden 22 verschiedene Arten von Emotionen unterschieden, wobei diese 22 Arten in 3 Hauptkategorien eingeordnet werden können.
Diese 3 Kategorien werden in Abbildung 2.6 durch die Zweige *Consequences of events, Actions of Agents* und *Aspects of Objects* dargestellt. Wie in 2.6 zu sehen, sind die Kategorien jedoch nicht strikt voneinander getrennt, sondern gehen teilweise ineinander über [Bar02]. Anhand von goals, standards (Erwartungen wie andere Agenten reagieren) und preferences (was der Agent mag oder nicht mag) werden den Ereignissen Emotionen zugewiesen.

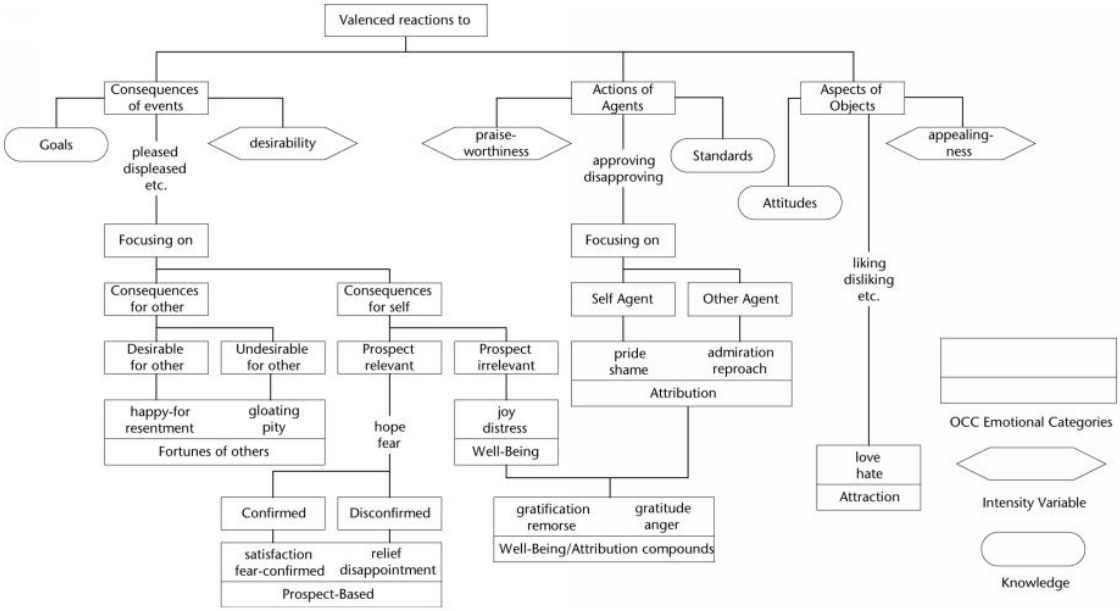


Abbildung 2.6: Das OCC Modell [Bar02]

Zur Verarbeitung von Emotionen im OCC Modell sind laut Bartneck [Bar02] 5 Phasen notwendig, auch wenn diese nicht alle explizit im original OCC Modell vorgesehen sind.
Die Phasen sehen wie folgt aus:

1. **Klassifikation:** In dieser ersten Phase wird eingestuft, welche der Kategorien von dem Ereignis, Aktion oder Objekt betroffen ist.
2. **Quantifizierung:** Berechnung mit welcher Intensität die Kategorien betroffen sind.
3. **Interaktion:** Auswirkungen auf andere Kategorien durch Änderungen an einer Kategorie berechnen.
4. **Mapping:** Da nicht in jeder Implementierung der OCC Modells alle 22 Kategorien verwendet werden, müssen eventuell die Kategorien aus dem OCC auf die tatsächlich vorhandenen Kategorien abgebildet werden.
5. **Ausdruck:** Der emotionale Zustand des Agenten muss durch Veränderungen im Verhalten, Gesichtsausdruck oder Ähnlichem zum Ausdruck gebracht werden.

2.5 Algorithmen und Design Pattern

In diesem Abschnitt werden die Algorithmen und Design Pattern erläutert, die bei der Realisierung des Agenten zum Einsatz kommen.

2.5.1 Observer Patter

Das Observer Pattern definiert „eine 1-zu-n-Abhängigkeit zwischen Objekten, so daß die Änderung des Zustandes eines Objektes dazu führt, daß alle abhängigen Objekte benachrichtigt und automatisch akutalisiert werden“ [GHJJ96].

Jeder Beobachter, der über Änderungen bei einem Subjekt (siehe Abbildung 2.7) informiert werden will, muss sich dafür nur bei diesem anmelden. Sobald der Beobachter dies gemacht hat, wird er automatisch über Änderungen des Subjekts informiert.
Dies bietet den Vorteil, dass Daten zwischen unterschiedlichen Klassen konsistent gehalten werden können, ohne dass eine enge Kopplung zwischen diesen notwendig wäre. Dadurch ist es möglich, Beobachter und Subjekt unabhängig voneinander zu verändern und auch wieder zu verwenden.
Allerdings kann es bei einer großen Anzahl von Beobachtern pro Subjekt auch zu sehr hohen Änderungskosten kommen, da immer alle Beobachter über jede Änderung informiert werden, auch wenn diese die Information zu diesem Zeitpunkt gar nicht benötigen.

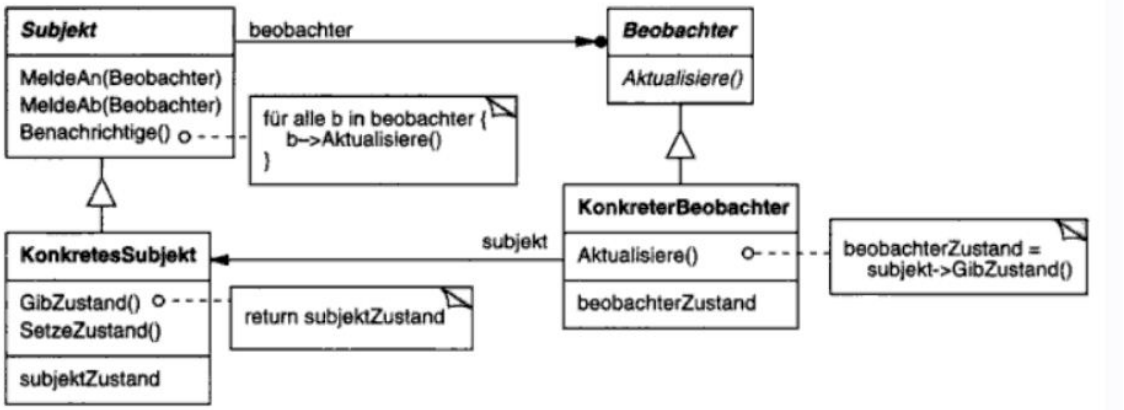


Abbildung 2.7: Observer Pattern UML [GHJJ96]

2.5.2 A*-Algorithmus

Der A*-Algorithmus ist ein weit verbreiteter Suchalgorithmus, der dazu genutzt werden, kann in einem Graphen die kürzeste Verbindung zwischen zwei Knoten zu finden.
So wird die Wegfindung in Spielen fast ausschließlich mit dem A*-Algorithmus realisiert, da er sehr effizient und einfach zu implementieren ist und außerdem viel Raum für diverse Optimierungen lässt [Mil06].
Jedoch lässt sich A* für viel mehr einsetzen, als nur für das Finden eines optimalen Pfades durch eine Menge von Navigationspunkten. So erwähnt Dan Higgins zum Beispiel, dass bei der Entwicklung des Spiels Empire Earth A* zur Terrain Analyse genutzt wurde [Hig02].
Den Algorithmus in aller Tiefe zu diskutieren ist nicht Sinn dieser Arbeit, da dieser jedoch intensiv während der Problemlösungsphase im GOAP Planner genutzt wird (siehe 5.2.2), sei er an dieser Stelle kurz erläutert:

g: Kosten, um vom dem Startknoten zu diesem Knoten zu gelangen
h: Die geschätzten Kosten, um von diesem Knoten zum Ziel zu gelangen (Heuristic)
f: g+h

1. P sei der Startknoten
2. Weise P f, g und h Werte zu
3. Füge P der offenen Liste hinzu. An diesem Punkt ist P der einzige Knoten in der offenen Liste
4. B sei der beste Knoten aus der offenen Liste (beste = geringster f-wert)
   a) Wenn B der Zielknoten ist, wurde der Pfad gefunden
   b) Wenn die offene Liste leer ist, wurde kein gültiger Pfad gefunden
5. C sei ein gültiger Knoten, der mit B verbunden ist
   a) Weise C f, g und h-Werte zu
   b) Prüfe ob C in der offenen oder geschlossenen Liste ist
      i. Wenn dies der Fall ist, prüfe ob der neue Pfad effizienter ist (geringerer f-Wert)
         A. Falls auch dies der Fall ist, aktualisiere den Pfad
      ii. Sonst füge C der offenen Liste hinzu
   c) Wiederhole Schritt 5 für alle gültigen Nachbarn von B
6. Wiederhole Schritt 4 [Mat02]

3 Analyse

In diesem Kapitel wird zunächst erläutert, warum die Kombination von GOAP und Emotionen sinnvoll ist und welche bestehenden Probleme dadurch gelöst werden sollen. Daraufhin wird in der Anforderungsanalyse herausgearbeitet welche Eigenschaften der Prototyp aufweisen muss, um die identifizierten Probleme zu beheben.

3.1 Warum GOAP + Emotionen

Wie schon in Kapitel 1.1 erwähnt, gibt es Zwei Probleme bei der Entwicklung von KIs für Computerspiele und Simulationen. Zum einen werden die KIs immer komplexer, was dazu führt, dass die Größe von Zustandsautomaten immer weiter zu nimmt, was wiederum zur Folge hat, dass diese kaum noch überschaubar sind. Diese unüberschaubare Komplexität der Automaten führt zu Problemen bei der Implementierung, da sie nicht mehr komplett verstanden werden [Isl05].
Zum anderen erwarten die Benutzer von Computerspielen und Simulationen ein immer realistischeres Erlebnis. Dies wurde lange Jahre durch die Verbesserung der Graphik erreicht, doch da diese inzwischen fast fotorealistisch ist, werden andere Methoden benötigt, um diese Anwendungen realitätsnäher zu gestalten.
Aus diesem Grund besteht großes Interesse daran die KI mit künstlichen Emotionen auszustatten, da diese dazu führen, dass die KIs glaubwürdiger erscheinen [Bat94]. Glaubwürdigkeit ist wichtig, da wir uns nur mit Charakteren identifizieren können, deren verhalten wir nachvollziehen können. Nur wenn wir uns mit dem was auf dem Bildschirm passiert identifizieren können, steigert dieses in Spielen die Immersion und macht Simulationen, wie etwa beim Militär, lebensechter.
Deshalb wird in dieser Arbeit ein emotionaler Agent auf Basis des GOAP entwickelt, der diese Probleme beheben soll.

3.2 Anforderungsanalyse

In diesem Kapitel werden zunächst bereits bestehende Methoden zur Entwicklung von Agenten analysiert. Anhand der Analyse werden daraufhin die Anforderungen an den hier zu entwickelnden Prototypen gestellt. Die hier gefundenen Anforderungen werden später die Grundlage für das Design und die Implementierung des Prototypen bilden.

3.2.1 Agenten Architektur

Die Architektur der Agenten bildet die Grundlage für alle weiteren Subsysteme und muss deshalb sorgfältig gewählt werden. Da im Rahmen des Forschungs- und Vertiefungsgebietes „Spieleprogrammierung und strategische Simulation“ eine Sammlung von wiederverwendbaren Softwarekomponenten in Java entstehen soll, ist eine der wichtigsten Anforderungen an die Architektur, dass sie modular aufgebaut ist.
Dies bedeutet, dass einzelne Subsysteme austauschbar sein müssen, ohne dass die anderen Subsysteme durch diesen Wechsel beeinträchtigt werden. Des Weiteren sollte die Architektur auf Performance optimiert sein, da in Echtzeitanwendungen, wie Simulationen, die Rechenzeit mit vielen anderen Subsystemen geteilt werden muss und somit begrenzt ist.

3.2.2 Dynamisches Verhalten

Wie aus der Problemstellung in Kapitel 1.1 hervorgeht, fehlt es den meisten KIs in Computerspielen und Simulationen an Glaubwürdigkeit in bestimmten Situationen.
Dieses Problem kommt zustande, da bei herkömmlichen Techniken zur Modellierung von künstlicher Intelligenz (vgl. Kapitel 2.3.1) jede Situation im Voraus explizit implementiert werden muss. Jede mögliche Situation schon während der Planungsphase vorauszusehen ist jedoch nahezu unmöglich. Dies führt dazu, dass die KIs in Situationen, die nicht explizit bedacht worden sind, der Situation unentsprechendes und somit unnatürliches Verhalten an den Tag legen.
Um dieses Problem zu beheben, soll in dieser Arbeit eine KI entwickelt werden, die dynamisch auf Änderungen in ihrer Umgebung reagieren kann. Das bedeutet, dass die KI eigenständig, aufgrund der ihr zur Verfügung stehenden Mittel, entscheiden können muss, wie ihr weiteres Verhalten aussieht.

Durch diese Verlagerung des Entscheidungsprozesses von der Planungsphase der Künstlichen Intelligenz in die KI selbst, wird es dem Entwickler abgenommen sich in einem immer komplexer werdenden Zustandsautomaten zurecht finden zu müssen (siehe Abbildung 3.1).

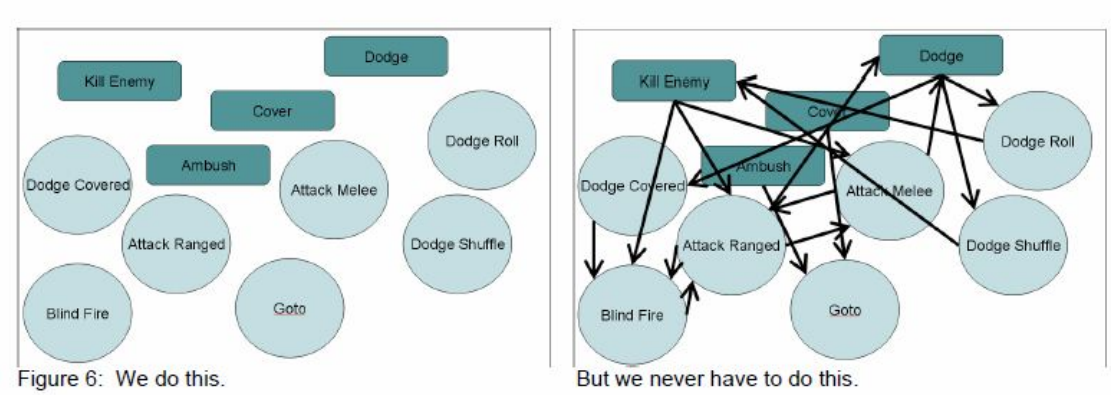


Abbildung 3.1: Vergleich Planning und FSM [Ork06]

Es gibt bereits viele Implementierungen von Agentensystemen, die das zuvor erwähnte Problem mit Hilfe von GOB (vgl. Kapitel 2.3.2) zu lösen versuchen. Ein Beispiel hierfür ist etwa JADE [JAD]. GOB bietet den Vorteil, dass es dem Entscheidungsprozess von Menschen sehr ähnelt. Dadurch kann es auch leicht von Leuten ohne großes Vorwissen in Agententechnologien verstanden werden [YSM08].
Jedoch bieten auch Agentensysteme wie JADE nicht die gewünschte Flexibilität, da sie zwar GOB-Design unterstützen, jedoch für die Erfüllung der Goals Pläne im Vornherein formuliert werden müssen. Diese Flexibilität ist sehr wichtig, da es die Glaubwürdigkeit steigert, wenn Agenten mit verschiedenen Persönlichkeiten die gleichen Ziele auf verschiedene Weise erreichen [PMCA99] (siehe Beispiel aus Kapitel 2.3.2.2).
Da des weiteren, wie in 3.2.1 schon erwähnt, eine Sammlung von wiederverwendbaren Softwarekomponenten in Java entstehen soll, wird in dieser Arbeit eine eigene GOB-Implementation entwickelt.

Des Weiteren soll die hier entwickelte KI auf Veränderungen in ihrer Umwelt sofort reagieren. Wie in Kapitel 2.3.1 beschrieben, wurden FSMs bereits mit einer hirarchischen Struktur erweitert, um das Problem zu umgehen, bei der Modellierung von unterschiedlich priorisierten Verhalten eine Verdopplung der Zustände zu erhalten.
Jedoch muss auch bei diesen HFSMs die Logik in jedem Zustand bis zum Ende durchlaufen werden, bevor der Übergang in einen anderen Zustand erfolgen kann. Dies kann zur Folge haben, dass eine FSM mit höherer Priorität erst verzögert ausgeführt wird. Daraus könnte etwa ein Verhalten resultieren, indem ein Agent, neben dem auf einem Markt eine Bombe explodiert ist, zuerst sein Essen zu Ende isst und erst dann wegläuft. Da dies kein glaubwür-

diges Verhalten darstellt, soll die hier entwickelte KI in der Lage sein, jeder Zeit ihr Verhalten der Umgebung anzupassen.

3.2.3 Emotionen

Emotionen sind ein essentieller Bestandteil für das menschliche Verhalten und sollten deshalb auch bei der Entwicklung von glaubwürdigen Agenten einbezogen werden [VDPBB+06]. Damit das Verhalten eines Agenten als glaubhaft angesehen werden kann, muss der Betrachter nachvollziehen können, warum der Agent in einer bestimmten Situation gerade dieses Verhalten zeigt. Das bedeutet, dass die Motivation des Agenten für den Betrachter nachvollziehbar sein muss.
Neben den Zielen, die ein Agent verfolgt, können auch Emotionen eine wichtige Rolle für die Motivation hinter einem Verhalten spielen. So würde ein mutiger Mensch einen Dieb wohl eher zur Rede stellen, als ein ängstlicher. In den meisten KIs ist das Verhalten jedoch nur von der jeweiligen Situation abhängig, jedoch nicht vom emotionalen Zustand des Agenten. Aus diesem Grund soll der hier entwickelte Agent das GOAP mit einem Emotionsmodell verbinden.
Durch die Kombination von GOAP mit einem Emotionsmodell sollen also Agenten entstehen, die nicht nur dynamisch Entscheidungen treffen können, sondern die Entscheidungen auch noch anhand einer Motivation treffen. Diese Entscheidungen müssen nach außen sichtbar sein, denn wenn die Agenten keine emotionale Reaktion zeigen, wirken sie auf den Betrachter nicht lebensecht [PMCA99].
Die Schwierigkeit in der Modellierung von Emotionen besteht darin, dass in der Literatur kein einheitliches Modell aller menschlicher Emotionen vorhanden ist (siehe Kapitel 2.4.1). Des Weiteren sind die vorhandenen Modelle wie etwa das OCC Modell (siehe Kapitel 2.4.1) sehr allgemein gehalten und somit oft zu komplex für die Entwicklung eines glaubhaften Agenten [Bar02]. Deshalb muss ein Modell entworfen werden, dessen Komplexität der Anwendung entsprechend ist und trotzdem die Simulation von Emotionen erlaubt.

3.2.4 Testscenario und -umgebung

Um die Auswirkungen von Agenten mit verschiedenen Emotionen und Zielen untersuchen zu können, muss ein Testscenario gewählt werden, das es erlaubt eine große Bandbreite dieser darzustellen. Außerdem muss das Scenario eine bekannte Umgebung darstellen, damit die vom Prototypen gezeigten Verhalten in einen vergleichbaren Kontext gesetzt werden können. Ein solches Scenario wird zum Beispiel in *Simulating crowd phenomena in African markets* [TGB09] beschrieben. Das Marktplatz Scenario bietet ausreichend Ziele wie etwa Essen,

Bummeln etc. bietet und es erlaubt eine große Bandbreite verschiedener Agenten auf ihm zu platzieren.

Darstellung: Um einen belebten Markt glaubwürdig darstellen zu können, muss die Realisierung performant genug sein, um eine sich auf dem Markt befindende Menschenmenge flüssig darstellen zu können. Deshalb sollten 25 frames per second nicht unterschritten werden, welches die Standard Bildwiederholungsrate bei einem Fernseher darstellt [Wik09c].
Damit die Grafik die Performance so wenig wie möglich beeinflusst, sollte für die Darstellung der Testumgebung auf eine professionelle Engine zurückgegriffen werden.

Auswertung: Um eine große Bandbreite von verschiedenen Scenarien simulieren zu können, sollte außerdem die Möglichkeit bestehen, die Agenten zu Beginn der Simulation mit unterschiedlichen Emotionen und Zielen initialisieren zu können.
Damit aus den auftretenden Verhalten Schlüsse gezogen werden können, muss des Weiteren die Möglichkeit bestehen, die einzelnen Agenten genauer zu untersuchen. Dies sollte direkt zur Laufzeit möglich sein, indem Agenten individuell ausgewählt werden können, um deren aktuelle Ziele und Emotionen angezeigt zu bekommen. Außerdem sollte nach Beendigung der Simulation eine Statistik über alle Agenten ausgegeben werden können.
Diese Statistik könnte zum Beispiel dazu genutzt werden um zu überprüfen, wie viele der Agenten den Markt glücklicher verlassen haben als sie ihn betreten haben. So könnte die Simulation genutzt werden um zu analysieren, wie sich die Umverteilung der Stände oder die Änderung der Preise auf die Zufriedenheit der Besucher auswirkt.

Navigation: Da die Agenten auf dem Marktplatz an bestimmte Orte, wie zum Beispiel zu einer Essensbude, gelangen müssen um ihre Ziele zu erfüllen, müssen die Agenten auf dem Marktplatz navigieren können. Hierfür wurden in der Fachliteratur zwei verschiedene Verfahren gefunden, die in Frage kommen.
Zum einen die Navigation per Steering Behaviour in Kombination mit Obstacle Avoidance [Rey01] und zum anderen Pathfinding via A* (siehe 2.5.2). Steering Behaviour bietet den Vorteil, dass es ohne vorherige Kenntnis über die Umgebung funktioniert. Auch sehen die Bewegungen häufig realistischer aus, da sich die Agenten nicht nur zwischen Knoten des Pathfinding Graphen bewegen können. Allerdings ist dies auch mit zusätzlichem Entwicklungsaufwand verbunden und könnte bei einer großen Anzahl von Agenten zu Performanceproblemen führen.

4 Design

In diesem Kapitel wird das Design des zu implementierenden Prototypen erläutert. Grundlage für das Design stellen die unter 3 gefundenen Anforderungen dar.
Ein wichtiger Punkt bei dem Design des Agenten ist, dass sowohl das GOAP System als auch das System für die Emotionen so entworfen werden, dass sie auch unabhängig voneinander in anderen Projekten wieder verwendet werden können.

Dieses Kapitel ist in vier Teile gegliedert. Im ersten Teil wird zunächst das Design des Emotionsmodells beschrieben. Der zweite Teil beschäftigt sich mit dem Design des GOAP Systems. Im dritten Teil wird schließlich erläutert, wie die Architektur des Agenten aussieht und wie in dieser Architektur die zuvor entworfenen Systeme miteinander interagieren.
Abschließend werden einige wichtige Aspekte der Testumgebung erläutert, die zur Validierung des entwickelten Agentensystems genutzt wird.

4.1 Emotionen

Die Wahl des Emotionsmodells ist auf das OCC Modell gefallen, da dieses Modell sehr gut dokumentiert ist und somit einen guten Einstiegspunkt bietet. Das OCC Modell wurde jedoch entworfen, um die ganze Bandbreite menschlicher Emotionen zu simulieren. Dies würde jedoch den Rahmen dieser Arbeit sprengen. Aus diesem Grund wurde eine vereinfachte Form des OCC Modells entworfen, die einfach zu implementieren ist und dennoch genutzt werden kann, um die Auswirkungen von Emotionen auf das Verhalten der Agenten zu untersuchen.

Wie in Abbildung 4.1 zu sehen ist, wurde die Grundstruktur des OCC Modells mit seinen Drei Hauptkategorien beibehalten. Allerdings wurden viele der Verzweigungen des Originals entfernt und die Anzahl der Emotionen auf vier beschränkt.
Die Emotionen anger, fear, happyness und sadness wurden ausgewählt, da sie Emotionen der Kategorie Eins darstellen (siehe Kapitel 2.4.1) und zwei Paare von gegensätzlichen Emotionen bilden (siehe Abb. 2.5). Durch diesen Gegensatz der Emotionen sollen die unterschiedlichen Verhaltensweisen, die durch diese hervorgerufen werden, verdeutlicht werden.

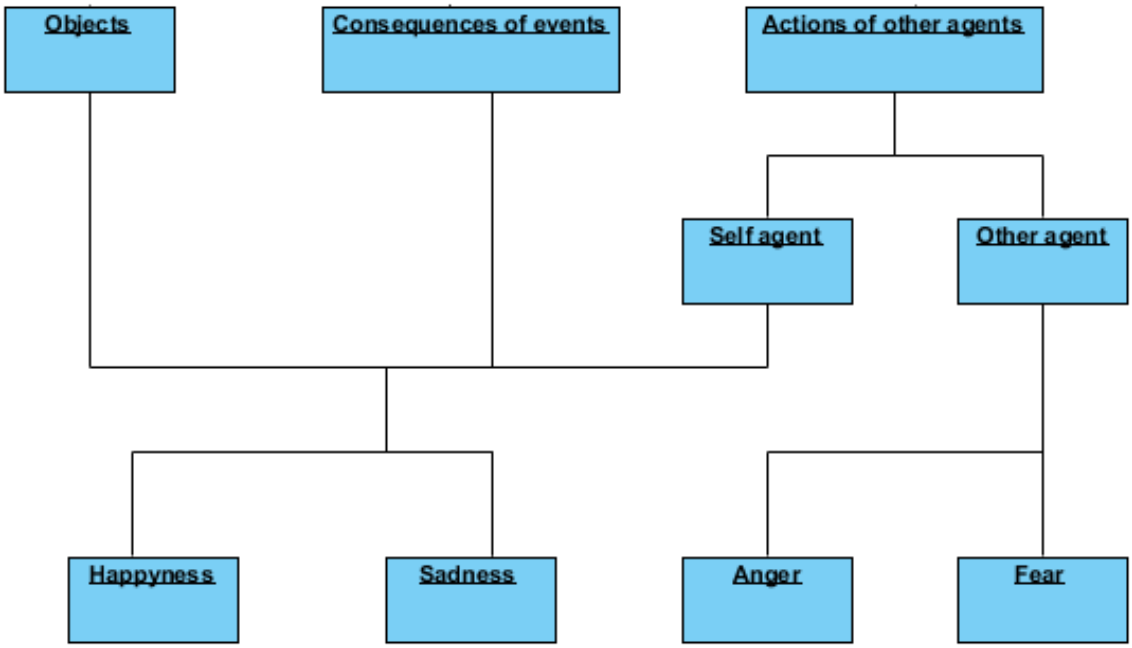


Abbildung 4.1: Vereinfachtes OCC Modell

Damit das entwickelte OCC Modell mit jeder Art von KI kombiniert werden kann, wurde bei dem Design stark auf die Verwendung von Interfaces gesetzt. Diese Interface orientierte Modellierung erlaubt es, dass jede KI, die das IOCCAgent Interface implementiert, sowohl auf emotionale Trigger von anderen Agenten als auch durch actions oder goals intern ausgelöste Emotionen reagieren kann.
Das IEmotionListener Inteface, von dem auch das IOCCAgent Interface erbt, erlaubt es darüber hinaus auch anderen Klassen auf ausgelöste Emotionen zu reagieren. Dies ist zum Beispiel hilfreich, um einen Logger einzuhängen, der die auftretenden Emotionen protokolliert.
Da auch die Erzeugung von Emotionen über das IEmotionTrigger Interface gekapselt ist, können auch Agenten mit verschiedenen KIs in der selben Simulation verwendet werden und dennoch Emotionen auslösen, die von jedem IEmotionListener interpretiert werden können.

Um die Auswirkungen der Emotionen auf den Agenten so einfach wie möglich an- und abschalten zu können, wurde auf das Observer Pattern (siehe Kapitel 2.5.1) zurückgegriffen. Die Verwendung dieses Patterns ermöglicht es, dass die actions und goals weiterhin Emotionen auslösen können, diese vom Agenten jedoch nur wahrgenommen werden, wenn er sich auch als EmotionListener bei dem OCC Model angemeldet hat (siehe Abbildung 4.2).

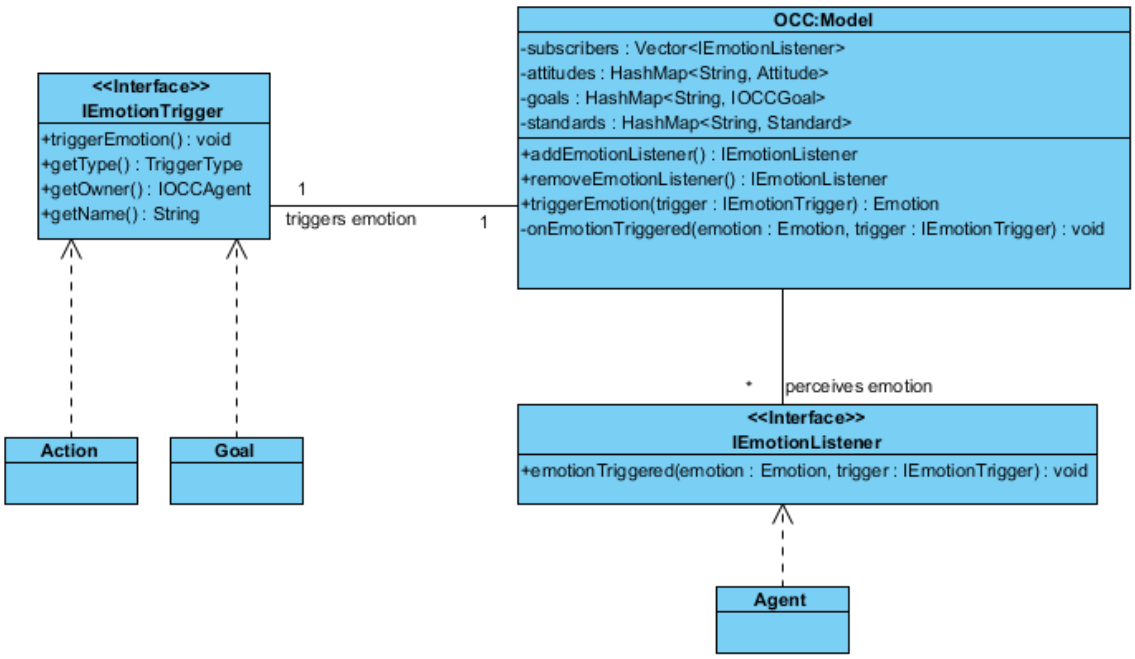


Abbildung 4.2: Emotion trigger und listener

Ein weiterer wichtiger Aspekt des Emotionsmodells ist, dass ein Ereignis bei verschiedenen Agenten unterschiedliche Emotionen auslösen kann. Aus diesem Grund wurden die Emotionen nicht direkt in den Auslöser eines Ereignisses mit eingebunden.
Stattdessen übermitteln die Auslöser für eine emotionale Reaktion (EmotionTrigger) lediglich welche Art von Trigger ausgelöst wurde, wer oder was den Trigger ausgelöst hat und den Namen des Triggers.
Aus diesen Informationen wird dann innerhalb des OCC Modells eines jeden Agenten die entsprechende Emotion ermittelt. Diese Emotion wird dann an alle EmotionListener, die sich bei dem OCC Modell registriert haben, übermittelt (siehe Abbildung 4.2).
Damit die Zuordnung von Triggern zu Emotionen funktionieren kann, muss das OCC Modell nach der Erstellung mit den goals, attitudes und standards, die dieser Agent hat, initialisiert werden.
Die Zuordnung einer Emotion zu einem Trigger wird anhand des TriggerTypes und des Namens des Triggers ermittelt. Bei goals ist festgelegt, dass die Erfüllung eines goals positive

Emotionen (Happyness) und das nicht Erfüllen eines goals negative Emotionen (Sadness) auslöst.
Attitudes hingegen müssen selbst spezifizieren, welche Emotion sie auslösen. So kann Beispielsweise das Essen einer Wurst bei dem einen Agenten Freude hervorrufen, weil seine attitude angibt, dass er Wurst mag und bei einem anderen Agenten Traurigkeit, weil er diese nicht mag. Ähnlich verhält es sich bei Standards, jedoch muss hier auch noch beachtet werden, von wem der Trigger ausgelöst wurde.
Dies ist notwendig, da standards unterschiedliche Emotionen auslösen können, je nachdem ob das Ereignis von dem Agenten selber oder von einem anderen Agenten ausgelöst wurde (siehe Abbildung 4.2). Ein Beispiel hierfür wäre etwa die StealFood Aktion. Diese könnte bei dem Agenten, der das Essen stiehlt, Angst auslösen, da er fürchtet erwischt zu werden. Bei einem Agenten, der dies beobachtet, könnte dieses aber Wut über das Verhalten des anderen Agenten hervorrufen.
Jedoch ist nicht jede Aktion eines Agenten von allen anderen Agenten wahrzunehmen. Gründe hierfür könnten sein, dass es sich um eine Aktion handelt, die nicht nach außen sichtbar ist, oder dass sie zwar sichtbar ist aber der andere Agent zu weit entfernt ist, um diese wahrzunehmen. Um dieses simulieren zu können, wurde das IActionTrigger Interface designt, welches von dem IEmotionTrigger Interface erbt. Jede Aktion, die auch bei anderen Agenten Emotionen auslösen können soll, muss daher dieses Interface Implementieren.
Damit nicht alle Agenten auf den ActionTrigger reagieren, sondern nur diejenigen, die es auch sollen, müssen diese Agenten bei dem ActionProvider eines Agenten angemeldet sein. Hierbei kommt wieder das Observer Pattern zum Einsatz (siehe 2.5.1), das bedeutet, dass der ActionProvider alle ActionTrigger an alle Agenten, die bei ihm registriert sind, weiterleitet. Dieser ActionTrigger löst bei den anderen Agenten wiederum einen EmotionTrigger aus, um eine emotionale Reaktion zu generieren.

4.2 GOAP

Um das in 3.2.2 geforderte dynamische Verhalten umsetzen zu können, wird hier eine KI auf Basis des Goal Oriented Action Plannings designt. Die Wahl fiel auf GOAP, da dieses mittels Planning sowohl dynamisches Verhalten ermöglicht, als auch Goal Oriented Behavior. Das GOAP System wurde als vollständig eigenständige Komponente designt. Das bedeutet, dass es nicht wie das zuvor beschriebene Emotionsmodell auf andere Komponenten aufsetzt. Aus diesem Grund wurde auch überwiegend auf den Einsatz von Interfaces verzichtet.

4.2.1 Actions und goals

Die goals und actions, die dem Agenten zur Verfügung gestellt werden, bilden die Grundbausteine für sein Verhalten. Goals repräsentieren was der Agent erreichen will und die actions stellen die Mittel dar, die dem Agenten zur Verfügung stehen, um seine Ziele zu erreichen.
Jedem goal ist ein Wert zwischen 0 und 1 zugeordnet, der dessen aktuelle Relevanz darstellt. Jede action hat bestimmte Kosten, die repräsentieren, wie schwer oder aufwendig eine Aktion ist. Diese Kosten werden jedoch nur bei der Formulierung eines neuen Plans zur Erfüllung eines goals in Betracht gezogen (Genaueres siehe Kapitel 5.2.3).
Da eine zu große Anzahl verschiedener goals und actions es erschweren würde die Auswirkungen der Emotionen auf das Verhalten der Agenten zu analysieren, wurde deren Anzahl möglichst gering gehalten. Um jedoch trotzdem ansatzweise einen belebten Markt simulieren zu können, wurden folgende goals und actions identifiziert:

4.2.1.1 Goals

Explore: Das Explore goal führt dazu, dass der Agent seine Umgebung erkundet. Dieses goal hat eine sehr niedrige Priorität und wird somit nur dann ausgewählt, wenn der Agent seine Umgebung noch nicht gut genug kennt, um eins seiner anderen goals zu erfüllen. Da die Wahl für die Wegfindung auf den A* Algorithmus gefallen ist (siehe 4.3.1.5) werden hierbei vordefinierte Wegpunkte abgelaufen. (Fulfillment condition: *AtDestination = true*)

GetFood: Das GetFood goal dient dazu, den Hunger eines Agenten zu stillen. Seine Priorität wird daher direkt durch den Hunger des Agenten bestimmt. Wird dieses goal aktiviert sucht der Agent den nächsten Ort auf, an dem es etwas zu essen gibt, um dort seinen Hunger zu stillen. (Fulfillment condition: *Hunger = 0*)

GetEntertainment: Ähnlich wie das GetFood goal wird auch die Priorität von GetEntertainment direkt von einer Variable beeinflusst. Jedoch ist der bestimmende Faktor hier die Langeweile des Agenten. Ist diese größer als jedes andere Verlangen des Agenten wird das goal aktiviert und der Agent versucht etwas interessantes zu finden, um seiner Langeweile Abhilfe zu verschaffen. (Fulfillment condition: *Boredom = 0*)

GoHome: Dieses goal veranlasst den Agenten den Markt zu verlassen. Die Priorität des goals hängt von der Erschöpfung des Agenten ab, als auch von seinem emotionalen Zustand. Je unzufriedener ein Agent ist, desto wahrscheinlicher ist es, dass er den Markt verlässt. (Fulfillment condition: *HasLeft = true*)

4.2.1.2 Actions

Eat: Befriedigt den Hunger des Agenten und macht den Agenten glücklich. (Effect: *Hunger = 0*, Precondition: *HasFood = true*).

BuyFood, StealFood: Führen beide dazu, dass der Agent etwas zu essen hat, jedoch hängen die Kosten für die jeweilige Aktion vom aktuellen emotionalen Zustand des Agenten ab, um so das Verhalten des Agenten zu beeinflussen (siehe 5.2.3). StealFood steigert zusätzlich die Angst des Agenten. (Effect: *HasFood = true*, Precondition: *AtDestination = true*)

WatchEntertainment: Lindert die Langeweile des Agenten, verringert also den Wert des WorldStateSymbols *Boredom*. Die Zeit, die diese Aktion in Anspruch nimmt, ist proportional zur Langeweile des Agenten. Macht den Agenten glücklich.(Effect: *Boredom -= EntertainmentValue * Time*, Precondition: *AtDestination = true*)

GoToLocation: Diese Aktion wird genutzt, um den Agenten zu bewegen. Jede Bewegung lässt die Erschöpfung des Agenten ansteigen. Allerdings stellt diese Aktion nur den nächsten Navigationspunkt auf dem Blackboard (siehe 4.3.1.1) bereit. Die eigentliche Bewegung wird dann vom MotorSystem ausgeführt. (siehe Abbildung 4.4) (Effect: *AtDestination = true, Exhaustion + 0.01 * Time*, Precondition: *HasDestination = true*)

Exit: Diese Aktion führt dazu, dass der Agent vom Marktplatz entfernt wird. (Effect: *HasLeft = true*, Precondition: *AtDestination = true*)

GetBestLocation: Diese Aktion sucht im WorkingMemory nach der Essens/Entertainment/Exit Position, die dem Agenten am nächsten ist. Diese Aktion hat keine normale precondition. Jedoch muss im WorkingMemory eine entsprechende Positionsinformation vorhanden sein, damit diese Aktion ausführbar ist. Dieses wird durch eine contextPrecondition geprüft (siehe procedural preconditions 2.3.2.2). (Effect: *HasDestination = true*)

4.2.2 GOAPManager

Das zentrale Element des GOAP Systems bildet der GOAPManager. Alle goals und actions, die dem Agenten zur Verfügung stehen sollen, müssen deshalb bei diesem registriert werden.
Um das Verhalten des Agenten jedoch auch zur Laufzeit beeinflussen zu können, wurde der GOAPManager so designt, dass goals und actions dynamisch hinzugefügt und entfernt werden können. Dieses dynamische Hinzufügen und Entfernen erlaubt es beispielsweise den Agenten lernen zu lassen.
So kann sich der Agent in einem Scenarion beispielsweise durch Beobachtung eines anderen Agenten dessen Fähigkeiten aneignen. Andersherum sind auch Scenarien denkbar, indem die Fähigkeiten eines Agenten zur Laufzeit reduziert werden. So könnte etwa die Aktion *Rennen* entfernt werden, nachdem der Agent eine Verletzung erlitten hat.

Der GOAPManager hat folgende zentrale Aufgaben:

1. **Relevanz der goals updaten:** Damit immer für das goal, welches gerade die höchste Relevanz hat, ein Plan formuliert wird, muss diese stetig vom GOAPManager neu berechnet werden.

2. **Auswahl des aktuellen goals:** Das goal mit der höchsten Relevanz kann nicht automatisch als das aktive goal gesetzt werden, da es sein kann, dass momentan keine Möglichkeit besteht dieses goal zu erfüllen. Deshalb muss der GOAPManager dieses prüfen und ein valides goal wählen.

3. **Formulierung eines Plans einleiten:** Wurde das aktuelle goal geändert, muss ein Plan formuliert werden, der dieses goal erfüllen kann. Der GOAPManager ist dafür zuständig, den A*-Algorithmus mit der Suche nach einem solchen Plan zu beauftragen.

4. **Auswahl der aktuellen action:** Damit der Agent sein aktuelles goal erfüllen kann, müssen die Aktionen abgearbeitet werden, die der aktuelle Plan enthält. Das bedeutet, dass der GOAPManager überprüfen muss, ob die aktuelle Aktion bereits zu Ende ausgeführt wurde. Ist dies der Fall, muss diese Aktion aus dem Plan entfernt werden und die nächste Aktion aktiviert werden. Andernfalls muss die aktuelle Aktion durchgeführt werden.

Jedes mal wenn das aktive goal oder die aktive action wechselt, wird ein Event ausgelöst. Dieses Event übermittelt das neue goal bzw. action, sowie das goal bzw. action das ersetzt wurde. Auf diese Weise können auch Instanzen außerhalb des GOAP Systems einfach auf Verhaltensänderungen reagieren.
Instanzen, die sich für diese Events interessieren, müssen sich dafür beim GOAPManager anmelden und das IGOAPListener Interface implementieren.

Der zeitliche Ablauf dieser Vorgänge ist im Sequenzdiagramm in Abbildung 4.3 dargestellt.

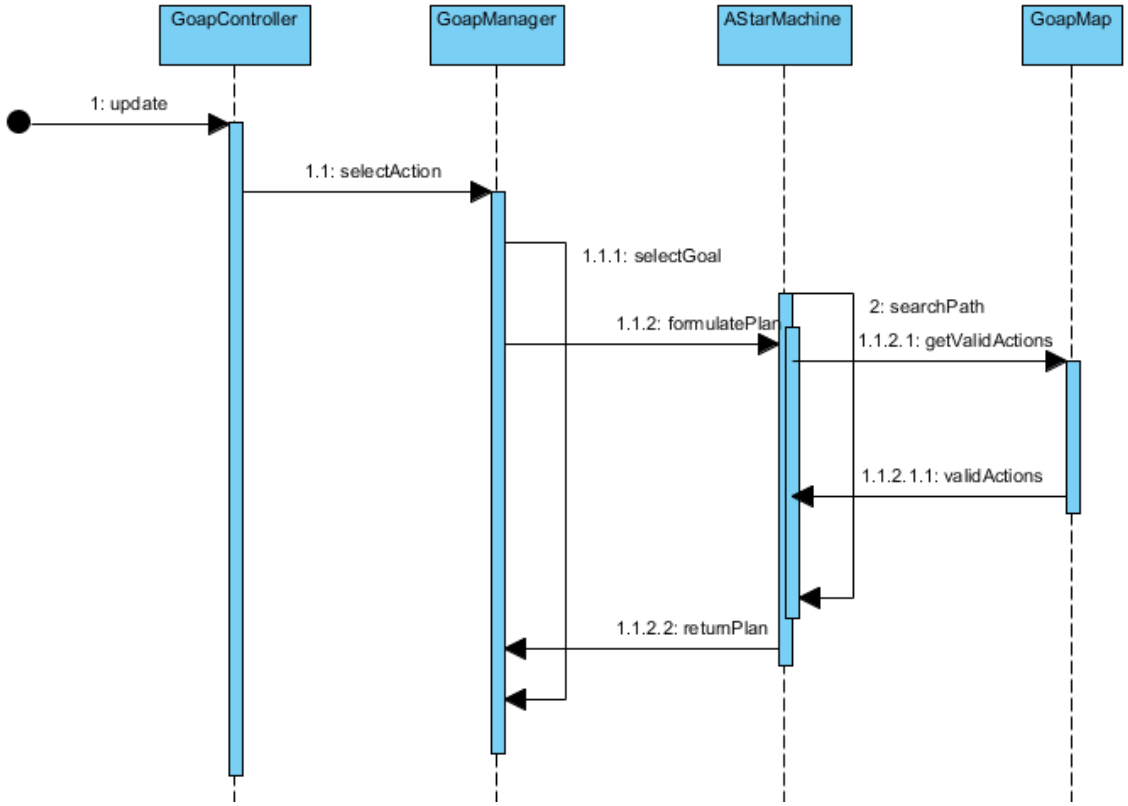


Abbildung 4.3: Ablauf zur Auswahl der aktuellen Aktion

Der in 4.3 zu sehende GOAPController ist die einzige am Planungsprozess beteiligte Instanz, die nicht im GOAP Package gekapselt ist. Der Grund dafür ist, dass es der Implementierung des jeweiligen Agenten überlassen werden soll, in welchen Abständen dieser die Aktualität der goals und actions überprüfen will. Dies ist notwendig, um die CPU Zeit, die das GOAP System erhält, auf die Anforderungen des jeweiligen Agentensystems anzupassen.

4.2.3 WorldState

Wie in Kapitel 2.3.2.2 beschrieben, wird der Zustand der Welt, in dem der Agent sich bewegt, im GOAP System durch einen Vektor von WorldStateSymbols beschrieben. Da zur Formulierung eines gültigen Plans die WorldStateSymbols vielfach miteinander verglichen werden müssen (für details siehe 5.2.3), wurden sie in der WorldState Klasse gekapselt.
Diese stellt Funktionen zur Verfügung um auf einfache Weise festzustellen, welchen Wert bestimmte WorldStateSymbols gerade haben und ob ein WorldStateSymbol gerade dem angestrebten Ziel entspricht.

4.3 Agenten Design

In diesem Kapitel wird das Design der Architektur, die dem Agenten zugrunde liegt, erläutert. Dabei wird detailliert auf die einzelnen Subsysteme eingegangen und deren Interaktion untereinander beschrieben.

4.3.1 Subsysteme

Als Basis für die Architektur der Agenten wurde das C4 Modell (siehe 2.1.1) gewählt, da in diesem Modell die einzelnen Subsysteme sauber voneinander getrennt sind und somit die in 3.2.1 geforderte Modularität bietet. Des Weiteren erlauben WorkingMemory und Blackboard das Zwischenspeichern von internen Daten. Dies erlaubt es rechenintensive Prozesse, wie etwa das Sammeln von Daten per Sensoren (siehe 4.3.1.2), über mehrere Frames zu verteilen, um CPU Zeit zu sparen.
Um die gewünschte Funktionalität zu erhalten, muss das C4 Modell jedoch modifiziert werden, da in ihm kein Subsystem für die Simulation von Emotionen vorgesehen ist. Darüber hinaus muss das im C4 vorgesehene Action System durch das GOAP System ersetzt werden (siehe Abbildung 4.4).

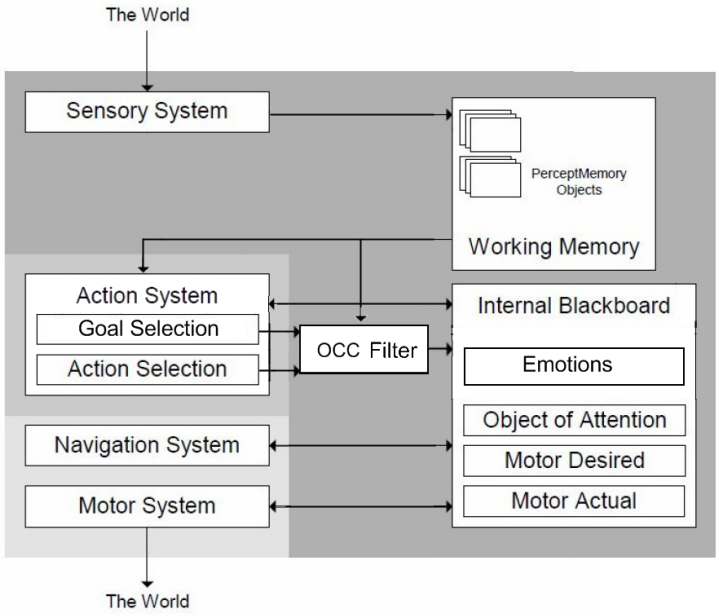


Abbildung 4.4: C4 Modell mit integriertem Emotionsfilter (Original: [Bar02])

4.3.1.1 Blackboard

Das Blackboard stellt die zentrale Kommunikationsschnittstelle zwischen den verschiedenen Subsystemen dar. Alle Subsysteme, die Informationen haben, die sie mit anderen Systemen teilen wollen, müssen diese auf dem Blackboard bereitstellen.
Die Kommunikation der Subsysteme über diese eine Schnittstelle laufen zu lassen hat den Vorteil, dass alle anderen Systeme voneinander entkoppelt bleiben. Würde es das Blackboard nicht geben, müssten etwa das ActionSystem und das MotorSystem miteinander direkt kommunizieren, um die Bewegung des Agenten zu realisieren. Dies hätte zur Folge, dass MotorSystem und ActionSystem nicht mehr ohne weiteres unabhängig voneinander ersetzt werden könnten.
Da das Blackboard nur als Datenspeicher fungiert, enthält es bis auf getter und setter Funktionen auch keine weitere Funktionalität.

4.3.1.2 Sensor System

Das Sensor System dient dazu, alle Sensoren, die dem Agenten zur Verfügung stehen, in regelmäßigen Abständen dazu zu veranlassen, aktuelle Daten zu sammeln. Da das Sammeln von Daten bei verschiedenen Sensoren unterschiedlich viel Zeit in Anspruch nehmen kann, wurde dieses System so entworfen, dass jeder Sensor eine eigene Updatefrequenz haben kann.
Die Sensoren dienen dazu, die Umwelt des Agenten wahrzunehmen und die gesammelten Daten dem Agenten über das WorkingMemory bereit zu stellen. Drei verschiedene Sensortypen wurden designt.

1. **FoodSensor:** der FoodSensor sucht den Sichtbereich des Agenten nach Objekten ab, welche die Bedingung *HasFood* auf true setzen können. Dies geschieht in regelmäßigen Abständen von etwa 200ms.
2. **EntertainmentSensor:** ähnlich wie der FoodSensor wird auch der EntertainmentSensor etwa alle 200ms erneuert und sucht den Sichtbereich des Agenten ab. Allerdings registriert dieser Sensor nur Objekte, welche die Bedingung *Entertainment* auf true setzen können.
3. **ActionSensor:** der ActionSensor ist dafür zuständig, actions, die Emotionen auslösen können, aufzunehmen, damit der Agent auf diese entsprechend reagieren kann. Anders als die anderen beiden Sensoren ist dieser Sensor event gesteuert. Das bedeutet, dass er automatisch alle actions, die im Sichtbereich des Agenten auftreten aufzeichnet, ohne dass explizit ein Update durch das SensorySystem erfolgen muss.

4.3.1.3 Action System

Das Action System ist, wie schon in 2.1.1 beschrieben, dafür zuständig anhand der Informationen, die im WorkingMemory und dem Blackboard bereit stehen, Entscheidungen zu treffen. Diese Entscheidungen werden wiederum auf dem Blackboard bereitgestellt und bilden die Grundlage für das Verhalten des Agenten.
In dieser Arbeit wird das Action System als das in 4.2 beschriebene System realisiert. Damit die Agentenarchitektur aber auch mit einem anderen Action Systems ausgestattet werden kann, wurde das IActionSystem Interface entworfen. Dieses Interface legt die Kommunikationswege zwischen ActionSystem, Blackboard und WorkingMemory fest und stellt somit sicher, dass kein Action System entworfen wird, das das modulare C4 Design umgeht.

4.3.1.4 OCC Filter

Wie in Abbildung 4.4 zu sehen ist, wird der OCC Filter, der für die Simulation der Emotionen verantwortlich ist, in den Kommunikationskanal von Blackboard und Action System eingehängt. Goals und actions können über das IEmotionTrigger Interface (siehe 4.1 interne Ereignisse auslösen, die von dem OCC-Filter in konkrete Emotionen umgewandelt werden, die auf dem Blackboard gespeichert werden. Diese Emotionen können wiederum Einfluss darauf nehmen, welches goal ausgewählt wird, als auch darauf, wie ein goal erreicht wird.
Die Auswahl von goals wird durch die Emotionen beeinflusst, indem sie in die Berechnung der Relevanz eines goals mit einfließen. So wird die Relevanz für das GoHome goal etwa so berechnet:

$$Relevanz = \begin{cases} 0 & Exhaustion + Fear < 0 \\ 1 & Exhaustion + Fear > 1 \\ Exhaustion + Fear & \text{sonst} \end{cases} \tag{4.1}$$

Dies hat zur Folge, dass ein Agent den Markt nicht nur verlässt, wenn er erschöpft ist, sondern auch wenn er zu ängstlich ist.

Die Auswahl von actions kann auf zwei verschiedene Weisen beeinflusst werden. Zum einen können die Kosten für eine Aktion durch Emotionen verändert werden, um diese Aktion bei der Formulierung eines Plans zu bevorzugen, bzw. zu benachteiligen (siehe 5.2.3). Zum anderen können Aktion in einem bestimmten emotionalen Zustand komplett ausgeschlossen werden. Dies geschieht, indem manche Aktionen in den proceduaral preconditions (siehe 2.3.2.2) auf einen bestimmten emotionalen Zustand prüfen und nur dann melden, dass die Aktion valide ist, wenn dieser emotionale Zustand gerade vorliegt.

Des weiteren erhält der OCC Filter auch Informationen aus dem WorkingMemory, um auch emotionale Ereignisse zu verarbeiten, die durch Sensoren wahrgenommen worden sind (siehe 4.3.1.2).

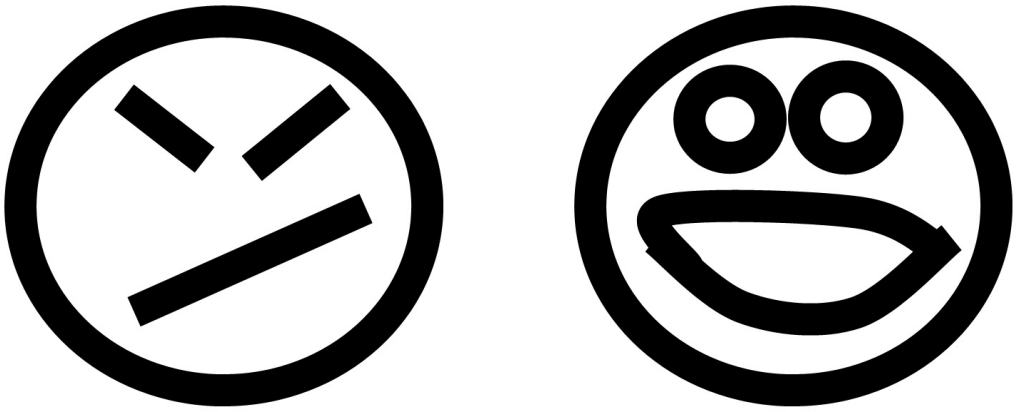
4.3.1.5 Navigation und Motor System

Das Navigation System ermöglicht es, dem Agenten eine Route von seiner jetzigen Position zu einer gewünschten Position zu ermitteln. Die Informationen über Start und Ziel für die Berechnung der Route wird dem Navigation System auf dem Blackboard bereitgestellt. Für die Planung einer Route wurde auf den A* Algorithmus zurückgegriffen (siehe Kapitel 2.5.2), da dieser ohnehin für das GOAP System implementiert werden musste.
Die ermittelte Route wird wiederum auf dem Blackboard bereitgestellt. Das Motor System ist schließlich dafür zuständig, die tatsächliche Bewegung des Agenten auszuführen und die gefundene Route abzuarbeiten.

4.4 Testumgebung

Zur Verifikation des entwickelten emotionalen Agenten wurde eine Testumgebung auf der Basis der JmonkeyEngine [JMo10] entworfen. Die Testumgebung stellt einen in 3D modellierten Marktplatz dar. Dieses Scenario wurde gewählt, um die Ergebnisse der Verifikation mit den in *Simulating crowd phenomena in African markets* [TGB09] gefundenen Ergebnissen vergleichen zu können. Um die Performance Anforderungen so gering wie möglich zu halten, wird alles auf dem Marktplatz lediglich durch farbige Quader repräsentiert.
Auf dem Marktplatz befinden sich Essensstände, die es dem Agenten erlauben seinen Hunger zu stillen und Unterhaltungsstände, die die Langeweile des Agenten verringern. Um eine gute optische Erkennbarkeit zu gewährleisten, sind Essensstände rot gehalten und Unterhaltungsstände blau.
Die Agenten selber ändern ihre Farbe je nach emotionalem Zustand.
Dabei repräsentiert die Farbe Grün, dass der Agent glücklich ist (Happyness > 0) und die Farbe Grau, dass der Agent traurig ist (Happyness <= 0). Wut und Angst werden durch zwei

Graphiken repräsentiert, die auf jeder Seite des Quaders, der den Agenten darstellt, einmal abgebildet sind. (sieh Abbildung 4.5)



(a) Angry (b) Frightened

Abbildung 4.5: Bilder zur Repräsentation des Emotionszustandes eines Agenten

Der Markt verfügt über vier Ein- und Ausgänge, die sich jeweils in eine der vier Himmelsrichtungen befinden. Über diese Ein- und Ausgänge können Agenten jeder Zeit die Simulation verlassen, oder neu hinzugefügt werden. Um das rege Kommen und Gehen eines belebten Marktes darzustellen, werden über die Eingänge in regelmäßigen Abständen neue Agenten in der Simulation platziert.
Die Geschwindigkeit, mit der sich die Agenten über den Marktplatz bewegen und Aktionen ausführen, wurde als ein Vielfaches der Geschwindigkeit festgelegt, die sie in der Realität benötigen würden. Diese Entscheidung wurde getroffen, um die Simulationsdauer zu verringern und Änderungen in den Verhaltensweisen der Agenten besser beobachten zu können.
Da es unmöglich ist das Verhalten von allen Agenten auf dem Marktplatz gleichzeitig zu beobachten, wurde ein Logger entworfen, mit dem das Verhalten eines Agenten protokolliert werden kann, um es später auszuwerten. Dieser Logger implementiert sowohl das IGOAP-Listener als auch das IEmotionListener Interface, um einfach an jeden Agenten angehängt zu werden.

5 Realisierung

In diesem Kapitel werden wichtige Aspekte der Realisierung erläutert. Das Augenmerk liegt hierbei auf der Implementierung des GOAP-Systems.
Darüber hinaus wird aber auch auf die Implementierung des Emotionsmodells und einiger Aspekte der Agenten Architektur eingegangen.

5.1 Agenten-Architektur

Die einzelnen Komponenten der Agenten-Architektur bieten relativ wenig interessante Implementationsdetails. Deshalb sei im Folgenden nur das Workingmemory näher erklärt, da dessen Realisierung einige interessante Aspekte aufweist.

5.1.1 Workingmemory und Blackboard

Das Workingmemory stellt das Gedächtnis des Agenten dar und ist deshalb im Wesentlichen eine Collection aus sogenannten MemoryObjects. Diese MemoryObjects stellen eine einzelne Erinnerung dar, und bestehen aus einem String, der den Typ der Erinnerung angibt (z.B „FoodLocation“), einer Position, an der die Erinnerung gesehen wurde und einem beliefe Wert, der repräsentiert, wie sicher sich der Agent dieser Erinnerung ist.
Dieser beliefe Wert kann ein float zwischen 1.0f und 0.0f sein und wird für jede gespeicherte Erinnerung kontinuierlich durch das Workingmemory erneuert. Verschwindet ein Agent z.B aus dem Sichtfeld eines anderen Agenten, nimmt der beliefe Wert über die Position des Agenten immer weiter ab, bis er schließlich 0.0f erreicht und die Erinnerung gelöscht wird. Taucht der Agent jedoch innerhalb dieser Zeit wieder im Sichtfeld auf, wird der belief Wert wieder auf 1.0f zurückgesetzt.
Die MemoryObjects werden innerhalb des Workingmemory in einer Hashmap gespeichert, um schnell alle Erinnerungen des selben Typs finden zu können. Es können beliebig viele Erinnerungen von einem Typ gespeichert werden, jedoch nicht dieselbe Erinnerung mehrmals. Zwei Erinnerungen werden als gleich angesehen, wenn sie vom gleichen Typ sind und die gleiche Position haben. Wird versucht eine doppelte Erinnerung dem Workingmemory hinzuzufügen, wird lediglich der beliefe Wert der bereits vorhandenen Erinnerung auf den

aktuelleren Wert gesetzt.

Das Blackboard sollte ursprünglich ähnlich wie das Workingmemory ein Container für jegliche Art von Objekten sein, die über einen String identifiziert werden können. Aus Zeitgründen wurde dies jedoch nicht mehr implementiert, so dass das Blackboard momentan nur vordefinierte Variablen speichern kann.

5.2 GOAP

Das GOAP-System ist die zentrale Komponente bei der Generierung des Verhaltens des Agenten und wird deshalb im folgenden ausführlich beschrieben.

5.2.1 Actions

Actions repräsentieren die atomaren Aktionen, die ein Agent ausführen kann um ein bestimmtes Goal zu erreichen. Jede Action leitet von der abstrakten Action Klasse ab, welche sowohl die Kosten für das Ausführen der Action, als auch je eine Liste von WorldStateSymbols für die Effekte und Vorbedingungen der Action definiert.
Während der Planungsphase wird anhand der Effekte, die diese Aktion auf die Welt hat, festgestellt, ob die jeweilige Action den aktuellen Zustand der Welt auf eine Art und Weise verändern kann, die den Agenten seinem aktuellen Ziel ein Stück näher bringt.
Eine Aktion ist jedoch nur zulässig, wenn die contextPreconditionsMet() Funktion den Wert true zurück liefert. Diese Funktion muss von jeder Action individuell implementiert werden und prüft anhand von Variablen im Workingmemory und dem Blackboard, ob die Action zum aktuellen Zeitpunkt überhaupt zulässig ist. Solch eine context precondition könnte z.B. sein, dass ein Agent erst einmal die Position seines Ziels kennen muss, bevor die Aktion GoToLocationXY ausgeführt werden kann.
Des Weiteren muss jede Action die performAction() Methode implementieren, die in jedem Update Zyklus der Anwendung vom GOAPManager aufgerufen wird und die eigentliche Aktion ausführt. Dies geschieht so lange, bis die Aktion über die isValid() Methode mitteilt, dass sie nicht mehr aktuell ist.
Dies kann entweder bedeuten, dass die Aktion zu Ende ausgeführt wurde und die nächste Aktion ausgeführt werden kann, oder dass etwas die korrekte Ausführung unterbrochen hat und ein neuer Plan gefunden werden muss.
Ob die Aktion erfolgreich beendet wurde oder nicht, kann durch die isFinished() Methode erfragt werden, die im Falle der erfolgreichen Ausführung abschließend noch Informationen für darauf folgende Aktionen auf dem Blackboard platzieren kann.

5.2.2 A*

In diesem Abschnitt werden einige Aspekte der A* Implementierung genauer erläutert, da sie fundamental für das Verständnis des GOAP Systems sind. Wie schon in Kaptiel 2.5.2 erwähnt, ist A* ein Suchalgorithmus, der nicht nur für die Wegfindung genutzt werden kann. In dem Prototypen wird der Algorithmus sowohl für die Wegfindung, als auch für das Finden einer gültigen Sequenz von Actions zur Erfüllung eines gegebenen Goals verwendet.
Um für mehrere Zwecke genutzt werden zu können, wurde die A* Maschine deshalb generisch gestaltet. Bei der Erzeugung einer Instanz der A* Maschine muss daher spezifiert werden, welcher Knotentyp mit der A* Maschine benutzt werden soll.
Des Weiteren erwartet der Konstruktor der AStarMachine Klasse eine Map, die für die Wegfindung als auch für das GOAP System, die gleichen Methoden zur Verfügung stellt, diese jedoch unterschiedlich implementiert. So arbeitet die NavigationMap für die Wegfindung zum Beispiel auf einem vorher bekannten Graphen aus Navigationsknoten, während die GOAP-Map den Graphen dynamisch während der Suche erstellt (siehe 5.2.2.2).

```
1 AstarMachine<GOAPNode> astar = new AStarMachine<GOAPNode>(goapMap);
```

Die eigentliche Suche wird angestoßen, indem die findPath(IMover mover, Node start, Node end) Methode aufgerufen wird. Diese Methode hat als Parameter zunächst einen mover, der es erlaubt für den gleichen Typ von Nodes unterschiedliche Verhalten zu definieren. So kann z.B. bei der Wegfindung ein Flugzeug bestimmte Nodes betreten, die ein Schiff nicht betreten kann.
Die nächsten beiden Parameter repräsentieren den Startpunkt der Suche, und den Endpunkt, zu dem die Verbindung gefunden werden soll. Hierbei sei erwähnt, dass bei der GOAP Suche vom Zustand, der erreicht werden soll, rückwärts zum aktuellen Zustand gesucht wird, also der eigentliche Ziel-Node als Start-Node übergeben wird und umgekehrt.

Da die Wegfindung nicht das Hauptaugenmerk dieser Arbeit ist und auch in großem Umfang in der Fachliteratur behandelt wird (siehe etwa [Mat02]), werden im Folgenden nur die für das GOAP System relevanten Aspekte erläutert.

5.2.2.1 GOAPNode

Die GOAPNode Klasse ist von der abstrakten Node Klasse abgeleitet und repräsentiert einen Zustand der Welt, also eine Menge von WorldStateSymbols (siehe 2.3.2.2). Darüber hinaus speichert der Node die f,g und h Werte der aktuellen A* Suche.
Der heuristische Wert h (siehe 2.5.2) wird in der GOAPNode durch die Anzahl WorldState-Symbols berechnet, deren Wert nicht dem angestrebten Goalstate entsprechen. Um es der A* Maschine zu ermöglichen die Sequenz von Actions und Nodes zu speichern, enthält jeder Node eine Referenz auf seinen Vorgänger-Node und auf die Action, die den Übergang

zwischen diesen darstellt.
Des Weiteren stellt jeder Node auch eine Funktion bereit, die prüft, ob dieser Node das Ende der Suche darstellt und A* somit terminieren kann.

5.2.2.2 GOAPMap

Die GOAPMap bildet neben dem A* Algorithmus selbst das Herzstück der Suche für eine geeignete Sequenz von Aktionen. Sie ist dafür verantwortlich zu ermitteln, welche Aktionen momentan ausgeführt werden können und konstruiert daraus die benachbarten Nodes.
Dies geschieht, indem alle dem Agenten zur Verfügung stehenden Aktionen daraufhin betrachtet werden, ob sie einen der unbefriedigten WorldStateSymbols des aktuellen Nodes befriedigen können. Wurde eine Aktion gefunden, die ein WorldStateSymbol befriedigen kann, wird zunächst geprüft, ob auch die context preconditions für diese Aktion erfüllt sind (siehe 5.2.1). Wird eine Aktion gefunden, welche alle Vorbedingungen erfüllt, wird ein neuer Node angelegt und der Zielzustand des aktuellen Nodes auf den neuen übertragen.
Des Weiteren werden jedoch zu dessen Zielzustand noch die Vorbedingungen der Aktion hinzugefügt, die den Übergang zwischen der aktuellen Node und dem neu erstellten Nachbarn darstellt. Zuletzt werden die unbefriedigten WorldStateSymbols gemäß der Auswirkungen der Aktion angepasst und auch in dem neuen Node gespeichert.
Wenn alle, dem Agenten zur Verfügung stehenden Aktionen betrachtet wurden, wird eine Liste mit den neu erstellten Nachbar-Nodes zurückgegeben, welche die Basis für die weitere A* Suche bilden.
Darüber hinaus ist die Map noch dafür zuständig, die Kosten für den Übergang von einer Node zur Anderen zu berechnen. Im Falle der GOAPMap besteht dies jedoch lediglich darin, die Kosten der Aktion, die den Übergang zwischen den Nodes darstellt zurückzugeben.

5.2.3 Planning

Die Formulierung eines neuen Plans funktioniert in seinen Grundlagen ähnlich wie das im Kapitel 2.3.2.1 vorgestellte Verfahren. Jedoch bringt die Integration des A*-Algorithmus und Emotionen in das Verfahren einige Änderungen mit sich, weshalb hier die Formulierung eines Plans noch einmal an einem Beispiel erläutert wird.
Gehen wir davon aus, dass der Agent, den wir betrachten, gerade Hunger hat. Das bedeutet, dass das goal GetFood zur Zeit die höchste Priorität hat und somit vom GOAPManager als aktives goal ausgesucht wird. Sobald das aktive goal wechselt, wird automatisch die Suche nach einem neuen Plan angestoßen, vorausgesetzt der WorldState, der zur Erfüllung des goals benötigt wird, liegt nicht schon vor.
In dem WorldState, der zur Erfüllung des GetFood goals notwendig ist, muss das WorldStateSymbol isHungry auf false gesetzt sein. Wie schon in 5.2.2 erwähnt, verläuft die Suche

nach einem validen Plan vom Ziel- zum Ausgangszustand, ist also regressiv.
Die Suche beginnt damit, dass der A* Algorithmus die geschätzte Distanz zum Ziel (Heuristik) berechnet. Da nur ein WorldStateSymbol unbefriedigt ist, nämlich isHungry, wird diese mit 1.0 bemessen (vgl 5.2.2.1).
Als nächstes werden die zur Verfügung stehenden Aktionen nach passenden Möglichkeiten durchsucht. Dabei wird nur die Aktion Eat gefunden, da dies die einzige Aktion ist, die den Effekt hat das WorldStateSymbol isHungry auf false zu setzen. Eat hat jedoch die Vorbedingung, dass der Agent etwas zu essen hat. Gehen wir davon aus, dass dies gerade nicht der Fall ist, sieht der daraus resultierende WorldState wie folgt aus:

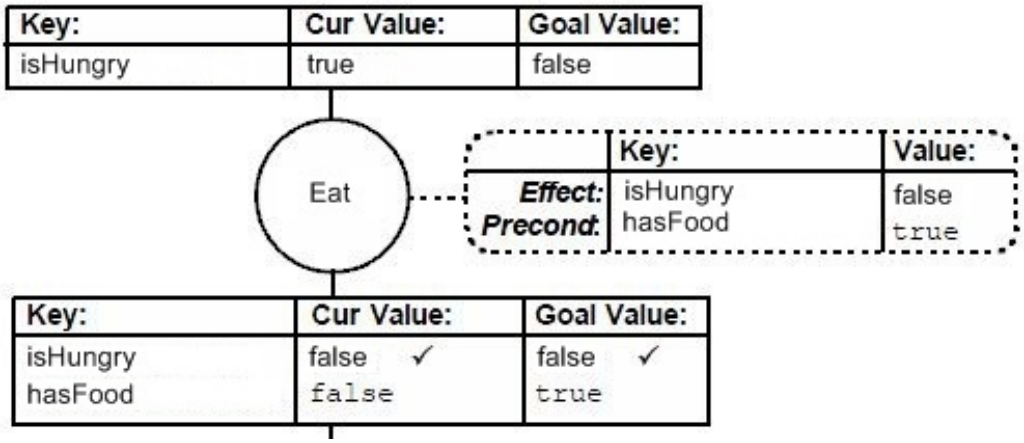


Abbildung 5.1: WorldState des Planners nach der 1. A* Iteration (Original:[Ork03])

Bei der nächsten Iteration des A* Algorithmus wird zunächst geprüft, ob bereits ein valider Plan formuliert wurde. Da jedoch noch unbefriedigte WorldStateSymbols vorhanden sind, ist dies nicht der Fall (siehe Abbildung 5.1). Deshalb wird erneut nach gültigen Aktionen gesucht. Diesmal werden StealFood und BuyFood gefunden.
Welche der beiden Aktionen der A* Algorithmus bevorzugt, entscheidet sich jetzt anhand der Kosten für diese Aktion. Wenn wir davon ausgehen, dass unser Agent gerade nicht unglücklich und wütend ist, werden sich die Kosten für die BuyFood Aktion auf 1.0f belaufen. Die Kosten für die StealFood Aktion werden jedoch einen höheren Wert haben, da diese in direktem Zusammenhang mit der Wut des Agenten stehen.

Auf diese Weise können Emotionen die Art und Weise, wie ein Agent ein Ziel erfüllt, beeinflussen. Der WorldState des Planners nach der Auswahl von der BuyFoodAktion wird in Abbildung 5.2 dargestellt.

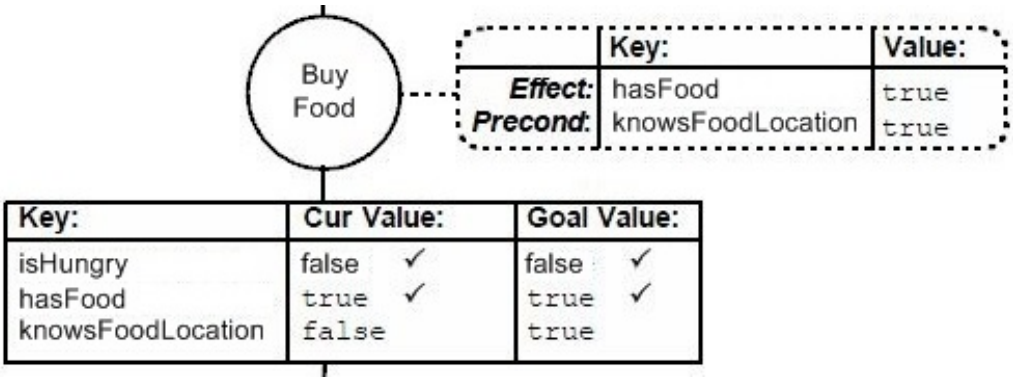


Abbildung 5.2: WorldState des Planners nach der 2. A* Iteration (Original:[Ork03])

Da jedoch auch BuyFood noch die Vorbedingung KnowsFooLocation hat, wird der beschriebene Prozess noch einmal wiederholt. Diesmal wird mit SearchForFood wieder nur eine Aktion gefunden, die den gewünschten Effekt hat. Weil SearchForFood jedoch keine neuen Vorbedingungen mit sich bringt, wird die Suche an diesem Punkt als erfolgreich angesehen und die gefundene Abfolge von Aktionen als Plan zurückgegeben.

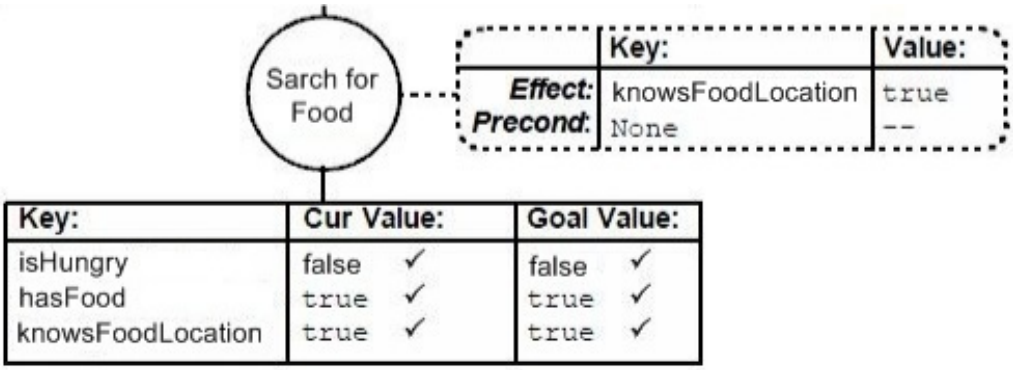


Abbildung 5.3: WorldState des Planners nach der 3. A* Iteration (Original:[Ork03])

5.3 Emotionsmodell

In diesem Kapitel werden die wichtigsten Aspekte bei der Realisierung des Emotionsmodells erläutert. Da das Emotionsmodell jedoch stark auf den Gebrauch von Schnittstellen setzt, die keine nennenswerten Implementationsdetails haben, wird im Folgenden nur die Implementation des OCC Modells selbst betrachtet.

5.3.1 OCC Modell

Das OCC Modell, welches zur Simulation von Emotionen zum Einsatz kommt, speichert die goals (was will der Agent erreichen), die attitudes (was mag der Agent) und die standards (was erwartet der Agent) jeweils in einer Hashmap.
Wird durch ein Objekt, welches das IEmotionTrigger Interface implementiert eine emotionale Reaktion angestoßen, wird dieser Trigger innerhalb des Modells verarbeitet. Dabei wird zunächst überprüft, von welchem Typ der Trigger ist (goal, attitude oder standard). Nachdem der Trigger einem Typ zugeordnet wurde, muss ermittelt werden, welche Emotion dieser auslöst.
Dies geschieht, indem in der zugehörigen Hashmap für den Trigger Typ die passende Emotion und deren Intensität anhand des Namens des Triggers ermittelt wird.
Dies ist im folgenden Codelisting beispielhaft für den TriggerType goal dargestellt.

```
1  if (trigger.getType() == TriggerType.Goal) {
               if (goals.containsKey(trigger.getName())) {
3                  if (goals.get(trigger.getName()).isFullfilled())
                       e = new Emotion(EmotionType.Happyness, goals.get(
5                              trigger.getName()).getDesirability());
                   else
7                      e = new Emotion(EmotionType.Sadness, goals.get(
                               trigger.getName()).getDesirability());
9
                   OnEmotionTriggered(e, trigger);
11                 return e;
               }
13         }
```

Wie bereits in 4.1 erwähnt, löst die Erfüllung oder das Scheitern von einem goal immer happyness beziehungsweise sadness hervor. Bei anderen Trigger Typen muss die passende Emotion zusätzlich anhand des Namen des Triggers ermittelt werden.
Wurde die passende Emotion generiert, wird diese mittels der OnEmotionTriggered Methode an alle Objekte weitergeleitet, die sich bei dem OCC Modell dafür registriert haben (siehe 4.1).

6 Test und Ergebnisse

In diesem Kapitel werden zunächst die Tests beschrieben, die mit dem entwickelten Prototypen durchgeführt wurden. Nach der Beschreibung eines typischen Testverlaufs wird in dem nächsten Abschnitt auf die Ergebnisse eingegangen, die sich aus den Tests ergeben haben. Die Daten, die bei den Testläufen gesammelt wurden, werden hinsichtlich des Verhaltens der Agenten bei der Simulation mit und ohne Emotionen untersucht. Darüber hinaus wird auf die Glaubwürdigkeit der Simulation bezüglich der Darstellung eines belebten Marktes eingegangen.

6.1 Testverlauf

Es wurden Zwei Arten von Tests durchgeführt. Bei dem ersten Test wurde das Verhalten der Agenten allein durch das entwickelte GOAP-System gesteuert, während bei dem zweiten Test auch die Simulation von Emotionen aktiviert wurde.
Zu Beginn eines jeden Testlaufs, werden 40 Agenten auf einem virtuellen Marktplatz platziert (siehe Abbildung 6.5). Die Startposition der Agenten ist dabei zufällig.
Jeder Agent in der Simulation besitzt das gleiche Set von actions und goals, hat also die selben Fähigkeiten und Ziele. Jedoch wird jeder Agent mit zufälligen Werten für die WorldState-Properties *Boredom, Hunger* und *Exhaustion* initialisiert, was dazu führt, dass die Agenten unterschiedliche Ziele priorisieren.
Wird die Simulation mit Emotionen durchgeführt, wird jeder Agent zusätzlich noch mit einem von vier Charaktertypen (siehe Tabelle 6.1) ausgestattet, welcher die Stimmung repräsentiert, mit der der Agent den Markt betritt.

	Anger	Happyness
Angry	0,8	-0,5
Frightened	-0,5	-0,2
Happy	0,2	0,5
Sad	0,2	-0,8

Tabelle 6.1: Die unterschiedlichen Charaktertypen

Zusätzlich zu den 40 Agenten, die sich bereits zum Start der Simulation auf dem Marktplatz befinden, werden in regelmäßigen Abständen (zwischen 0 und 2,5 sek) neue Agenten über die vier Eingänge auf dem Markt platziert (siehe Abbildung 6.1). Befinden sich bereits 80 Agenten auf dem Markt, wird das Platzieren von neuen Agenten kurzzeitig ausgesetzt, bis andere Agenten den Markt wieder verlassen haben.

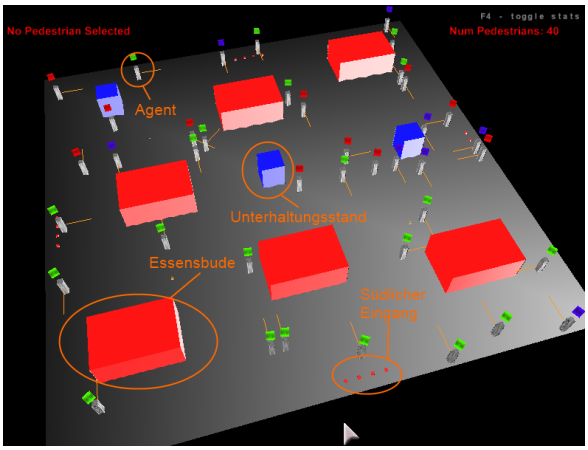


Abbildung 6.1: Beschreibung des Marktes

Immer wenn ein Agent auf dem Marktplatz platziert wird, wird ein Timer gestartet, der solange läuft, bis der Agent den Marktplatz wieder über einen der Eingänge verlässt. Nach dem Verlassen werden der CharacterType und die Zeit, die der Agent auf dem Markt verbracht hat, für die spätere Auswertung in einer Excel Tabelle gespeichert.
Jeder Testlauf dauert vier Minuten und wurde fünf mal wiederholt, um Schwankungen, die durch die zufällige Zuordnung der Variablen *Boredom, Hunger* und *Exhaustion* hervorgerufen werden, auszuschließen.

Alle Tests wurden auf einem PC mit Intel(R) Core(TM)2 Duo CPU mit 1,60GHz, 2GB Ram und einer Intel GMA x3100 Grafikkarte durchgeführt.

6.2 Ergebnisse

Im folgenden Abschnitt werden die Ergebnisse dargestellt, die während der Tests gesammelt wurden. Dabei wird zunächst auf das Verhalten der Agenten selbst eingegangen. Danach werden die gesammelten Ergebnisse daraufhin untersucht, wie sehr sich der entwickelte Prototyp dazu eignet einen belebten Markt zu simulieren.

6.2.1 Verhalten der Agenten

Da Agenten autonome Entitäten bilden, dessen Verhalten nach dem Start der Simulation nicht mehr beeinflusst werden kann, ist die objektive Beurteilung der auftretenden Verhalten schwierig.
Des Weiteren erschwerte die große Anzahl gleichzeitig agierender Agenten auf dem Marktplatz die Untersuchung einzelner Agenten. Diese Probleme konnte auch durch das Testen von einzelnen Agenten nicht umgangen werden, da für viele Verhaltensweisen die Interaktion zwischen Agenten notwendig ist.
Aus diesem Grund wurde das Verhalten der Gesamtheit der Agenten auf dem Markt als Bewertungsmaßstab für die Simulation gewählt.

Bei der Simulation der Agenten ohne zusätzliche Emotionen wird das Verhalten der Agenten allein durch das GOAP-System gesteuert. Eine der primären Anforderungen an dieses System war, leicht dynamische Verhaltensweisen realisieren zu können (vergleiche 3.2.2). Dieses dynamische Verhalten konnte auch während des Simulationsverlaufes beobachtet werden.
So war es den Agenten zum Beispiel möglich, auf einer beliebigen Position des Marktes zu starten (siehe 6.5) und anhand seiner Umgebung und Bedürfnisse sein Verhalten zu definieren.
Dies ist auch anhand des folgenden Auszugs aus dem Logfile eines Agenten zu sehen:

1: New Goal: GetFood relevance: 0.46020627
2: New Action: GetFoodLoc
3: Action GetFoodLoc finished
4: New Action: GoToLocation
5: New Action: BuyFood
6: New Action: Eat
7: Goal GetFood fullfilled relevance: 0.0
8: New Goal: Explore relevance: 0.2
9: New Action: GetUnvisitedLoc
10: Goal Explore aborted relevance: 0.2
11: New Goal: GetEntertainment relevance: 0.63680923
12: New Action: FindEntertainment
13: Action FindEntertainment finished
14: New Action: GoToLocation

Des Weiteren zeigt das Logfile, dass der Agent sein Verhalten dynamisch ändern kann. So wird das goal Explore abgebrochen und auf das GetEntertainment goal gewechselt (Zeile 10 und 11), sobald ein Unterhaltungsstand im Sichtbereich des Agenten auftaucht und das

GetEntertainment goal eine entsprechend hohe Relevanz hat.
Jedoch fällt im Vergleich zur Simulation mit Emotionen auf, dass es weniger Vielfalt im Verhalten der Agenten gibt. Dies führt dazu, dass sich das Verhalten aller Agenten in Intervallen ändert. So ist zum Beispiel nach etwa zwei Minuten Simulationszeit ein Massenabwandern der Agenten zu verzeichnen (siehe 6.2), was nicht dem Verhalten einer Menschenmenge auf einem realen Markt entspricht.

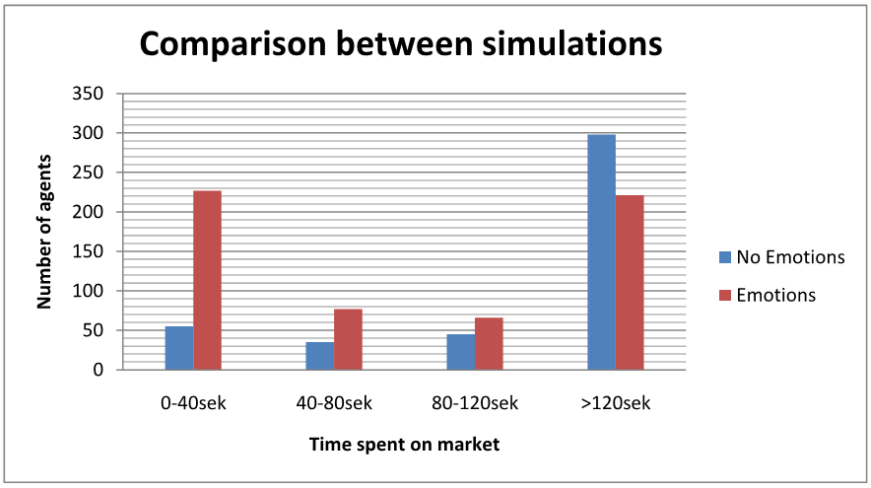


Abbildung 6.2: Auf dem Markt verbrachte Zeit der Agenten

Wie in dem Diagramm 6.2 zu sehen ist, tritt dieses Problem bei der Simulation mit Emotionen nicht auf, da die Zeit, die die Agenten auf dem Markt verbringen, viel unterschiedlicher ist.
Betrachtet man die Dauer des Aufenthalts für jeden CharacterType einzeln (siehe 6.3), wird auch deutlich, dass die in 6.2 zu sehende Varianz durch die verschiedenen CharacterTypes hervorgerufen wird.

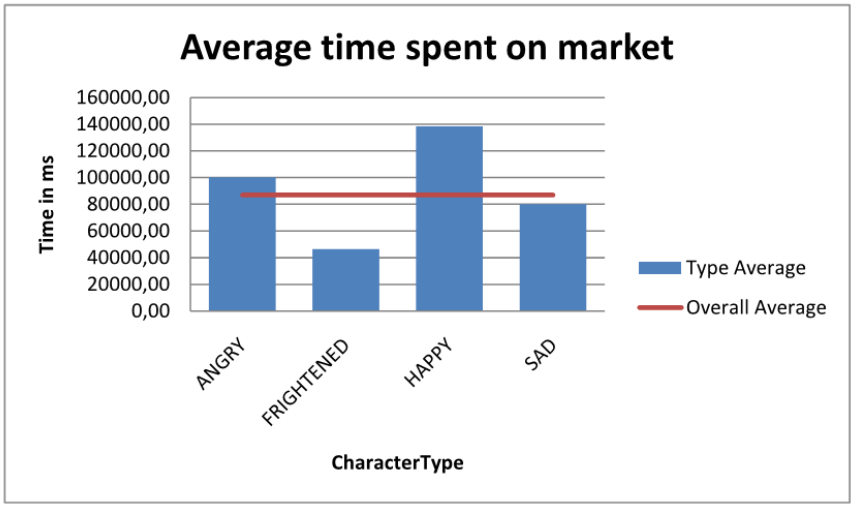


Abbildung 6.3: Vergleich der verschiedenen CharacterTypes 1

In 6.3 ist deutlich zu sehen, dass ein Agent, der den CharacterType Frightened hat, im Durchschnitt sehr viel weniger Zeit auf dem Markt verbringt, als ein Agent mit dem CharacterType Happy.
Wie im Diagramm 6.4 zu sehen ist, bedeutet dies jedoch nicht, dass alle Agenten mit dem gleichen CharacterType sich genau gleich verhalten, sondern nur, dass der CharacterType eine Tendenz vorgibt. Ein Agent mit dem CharacterType Frightened kann also auch eine lange Zeit auf dem Markt verweilen, wenn in seiner Umgebung zunächst nur positive Dinge geschehen, so dass der Agent seine anfängliche Angst verliert.

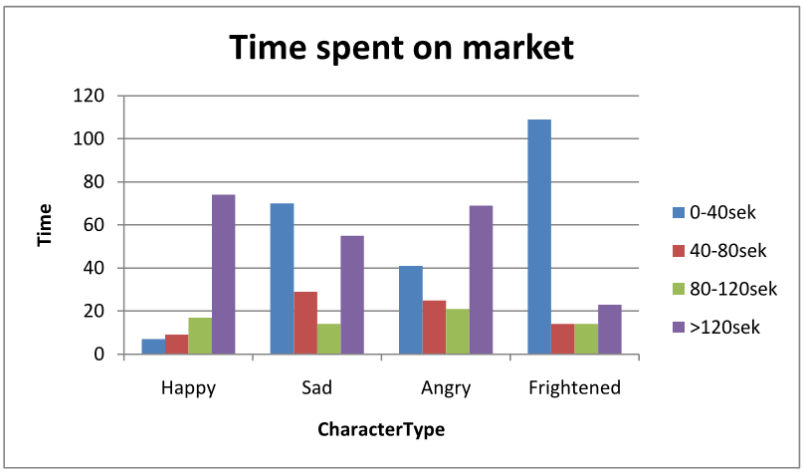


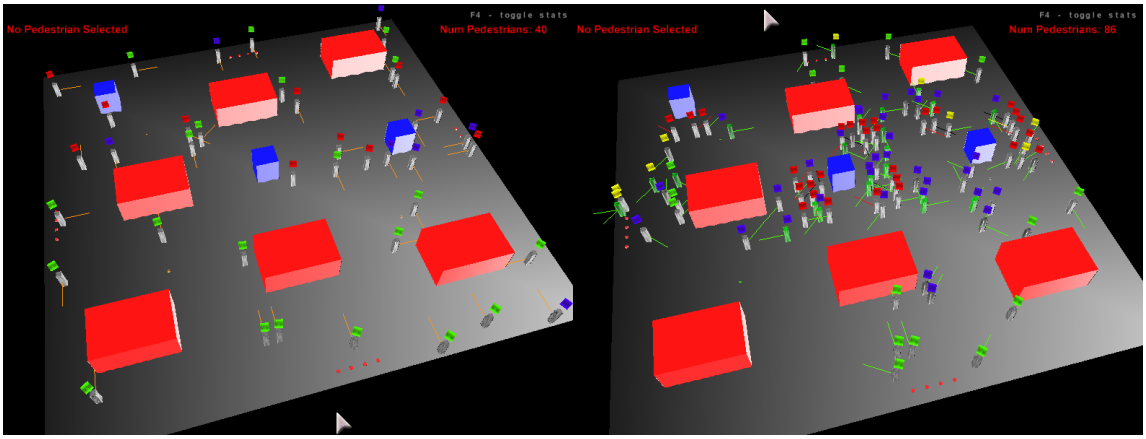
Abbildung 6.4: Vergleich der verschiedenen CharacterTypes 2

Ein weiterer Unterschied, der zwischen den beiden Simulationstypen beobachtet werden konnte ist, dass das Verhalten eines Agenten bei der Simulation ohne Emotionen keinen Einfluss auf die anderen Agenten hat. Bei der Simulation mit Emotionen spielt dieses jedoch eine große Rolle, da es zum Beispiel viele Agenten komplett verängstigen kann, wenn nur ein paar Agenten in ihrer Umgebung sich dazu entscheiden ihr Essen zu stehlen. Dies Wiederum kann dazu führen, dass es zu einer kleinen Massenpanik kommt und viele Agenten in der unmittelbaren Umgebung den Markt verlassen.
Da die Geschwindigkeit, mit der sich Agenten auf dem Markt bewegen, direkt von dem emotionalen Zustand des Agenten abhängt, unterscheidet sich in den beiden Simulationen auch die Art und Weise, wie sich die Agenten auf dem Markt bewegen. Bei der Simulation ohne Emotionen bewegen sich alle Agenten mit der selben Geschwindigkeit, solange ihnen nichts im Weg steht. Dies führt dazu, dass die Agenten auf dem Markt eher eine homogene Menge bilden, während bei der Simulation mit Emotionen eine sehr viele höhere Dynamik in den Bewegungen der Agenten ist.
Durch die Kombination des dynamischen Verhaltens, welches durch GOAP ermöglicht wird, mit dem Emotionsmodell konnten auch emergente Verhaltensweisen beobachtet werden, die in der Design-Phase nicht vorausgesehen wurden. So entstanden zum Beispiel Situationen, in denen manche Agenten ihr Essen gestohlen haben, was jedoch dazu führte, dass sie durch ihre Tat selbst so verängstigt wurden, dass sie sofort den Markt verließen, ohne das gestohlene Essen zu verzehren.
Jedoch hat die hohe Dynamik des GOAP-Systems auch zu Problemen geführt. So können Situationen auftreten, in denen sich das aktuelle goal des Agenten andauernd ändert, wenn etwa die Werte der WorldStateProperties *Boredom* und *Exhaustion* nahezu identisch sind.
In dieser Situation kann es passieren, dass ein Agent gerade auf dem Weg nach Hause ist, sich jedoch auf dem Weg sofort langweilt und umdreht, um Unterhaltung zu suchen. Wieder bei dem Unterhaltungsstand angekommen, ist jedoch das Bedürfnis nach Unterhaltung sofort wieder gedeckt und der Agent will wieder nach Hause gehen, wobei ihm wiederum langweilig wird.
Diese Art der Schwankung konnte durch die Einführung von Emotionen vermindert, jedoch nicht vollkommen behoben werden.

6.2.2 Darstellung des Marktes

Die Ergebnisse, die bezüglich der glaubwürdigen Darstellung eines belebten Marktes erzielt wurden, konnten größtenteils nur visuell durch Beobachten der Simulation gesammelt werden und können somit nicht streng objektiv an einem bestimmten Maßstab gemessen werden.
Wie in Abbildung 6.5 zu sehen ist, bilden sich auf dem Markt nach einer gewissen Zeit Menschenmengen an interessanten Stellen (Unterhaltungsstände). Dieses Verhalten wurde auch

in dem paper *Simulating crowd phenomena in African markets* [TGB09] beobachtet und wird dort als eines der wichtigsten Kriterien für die Simulation eines Marktes genannt.



(a) Simulationsstart (b) Simulationsende

Abbildung 6.5: Verteilung der Agenten auf dem Marktplatz

Des Weiteren ist in 6.5 zu erkennen, dass Stände, die nicht zentral gelegen sind, weniger Zulauf haben. Besonders deutlich ist dies bei dem Essensstand in der linken unteren Ecke von 6.5(b) zu sehen. Dieser Stand ist vollkommen leer, da er weder an einem der vier Eingänge platziert ist, noch an einen Unterhaltungsstand grenzt, der Besucher anziehen würde. Auch dieses Phänomen wurde in Tasses paper [TGB09] beschrieben.
Wie die Menge von Agenten sich auf dem Markt bewegt, ist, wie schon in 6.2.1 beschrieben, je nach Simulationsmodus sehr unterschiedlich. Welches der beiden Bewegungsmuster glaubwürdiger ist, hängt sicher von der Situation ab, die modelliert werden soll. Jedoch können die homogenen Bewegungen der Menge in der Simulation mit Emotionen dadurch erreicht werden, dass die Bewegungsgeschwindigkeit nicht mehr von dem emotionalen Zustand abhängig ist.
Ein Problem, welches bei beiden Simulationsmodi zu beobachten ist, ist, dass die Bewegungen der Agenten sehr abrupt und eckig sind. Dies ist auf die geringe Dichte des A* Graphen zurückzuführen, der für die Wegfindung genutzt wird. Auch funktioniert die Kollisionsvermeidung nicht 100%ig, was die Glaubwürdigkeit der Simulation zusätzlich schmälert.
Betrachtet man jedoch nicht die Gesamtheit des Marktes, sondern jeden Agenten für sich, werden weitere Probleme erkennbar. Dies hat mehrere Gründe.
So machen es die abstrakte Form und die fehlenden Animationen schwer, sich unter dem Agenten einen Marktbesucher vorzustellen. Außerdem ist es schwer nachzuvollziehen, warum der Agent gerade ein bestimmtes Verhalten zeigt, da nicht ersichtlich ist, welche Aktionen er momentan konkret ausführt. Die einzigen Indikatoren hierfür sind das aktuelle goal, welches über dem Agenten angezeigt wird und die Indikatoren für den emotionalen Zustand des Agenten (siehe 4.4). Diese reichen zwar nicht aus, um dem Agenten eine

hohe Glaubwürdigkeit zu verleihen, tragen jedoch dazu bei, das Verhalten der Agenten im Vergleich zur reinen GOAP-Simulation nachvollziehbarer zu machen.

Die Performance der Simulation hingegen kann objektiv beurteilt werden. Wie in 6.1 bereits erwähnt, wurde die Anzahl der Agenten, die sich gleichzeitig auf dem Markt befinden können, auf 80 beschränkt. Diese Beschränkung wurde eingeführt, da bei mehr als 80 Agenten die in 3.2.4 geforderten 25FPS nicht mehr aufrecht erhalten werden können.
Jedoch scheint hierbei die entwickelte KI nicht der limitierende Faktor zu sein, da der GoapController und der MovementController (zuständig für die Wegfindung) zusammen nur 17,7% der Rechenleistung ausmachen (siehe Abbildung 6.6).

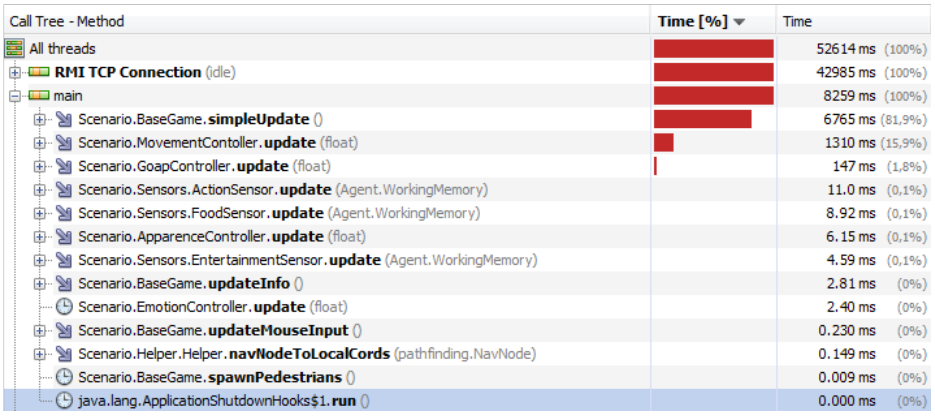


Abbildung 6.6: Profiling Ergebnis der Simulation

7 Diskussion und Ausblick

In diesem Kapitel wird zunächst betrachtet, welche Probleme während der Entwicklung des Prototypen aufgetreten sind und welche Erkenntnisse dabei gesammelt wurden. Im Weiteren wird dann noch einmal der Verlauf der Arbeit zusammengefasst und schließlich ein Ausblick darüber gegeben, wie der Prototyp weiterentwickelt werden könnte und welche Einsatzmöglichkeiten es für diesen geben könnte.

7.1 Rückblick

Im Laufe der Entwicklung des Prototypen wurden viele Erkenntnisse gewonnen. So war ursprünglich geplant, alle WorldStateSymbols nur als Boolean Variablen zu implementieren, um möglichst einfach überprüfen zu können, ob ein bestimmter Zustand vorherrscht oder nicht. Allerdings stellte sich sehr schnell heraus, dass es äußerst umständlich und einschränkend ist, die gesamte Welt nur auf wahr oder falsch zu reduzieren.
Die Erweiterung der WorldStateSymbols auf unterschiedliche Datentypen brachte einige Probleme mit sich, stellte jedoch bei weitem nicht die größte Herausforderung bei der Entwicklung des GOAP-Systems dar. Besonders schwierig war es, den planning Prozess selbst zu implementieren.
Da die Paper über das GOAP-Verfahren größtenteils nur das Prinzip des planning Prozesses behandeln, ohne auf Implementierungsdetails einzugehen, waren viele Iterationen von Tests und Weiterentwicklung notwendig, bis der GOAP-Planner voll funktionsfähig war.
Die Implementierung des Emotionsmodells hingegen gestaltete sich recht einfach. Hierbei lag die Schwierigkeit darin, herauszufinden, welche Emotionen simuliert werden sollen und vor allen Dingen, wie diese Emotionen das Verhalten beeinflussen sollen. So ist zum Beispiel nirgendwo klar definiert, ob ein Agent, der traurig ist, eher mehr isst, oder doch weniger.
Die Zusammenfügung der beiden zunächst unabhängig entwickelten Systeme stellt aufgrund des modularen Designs kein Problem dar. Was jedoch nach der Entwicklung des eigentliches Systems noch sehr viel Zeit in Anspruch genommen hat war, die Auswirkungen der einzelnen Variablen, die die Bedürfnisse und Emotionen des Agenten darstellen, aufeinander abzustimmen. Dies stellte eine besondere Herausforderung dar, da es kein objektives Maß für das Verhalten der Agenten gibt und die Änderung nur einer Variable zu einem radikal anderen Verhalten führen kann.

7.2 Zusammenfassung

Wie in der Einleitung (siehe 1.2) beschrieben, war das Ziel dieser Arbeit, emotionale Agenten zu erschaffen, die von ihren Emotionen in Entscheidungen beeinflusst werden. Die Motivation dafür liegt in der Absicht, das Verhalten der Agenten vielfältiger zu gestalten und dadurch Spiele und Simulationen lebensechter und glaubwürdiger erscheinen zu lassen.
Dafür wurden zunächst die grundlegenden Konzepte erläutert, welche die Basis für die Entwicklung des Agenten bilden. So wurde die C4-Agentenarchitektur erläutert, die das Grundgerüst für die Entwicklung einer Künstlichen Intelligenz darstellt. Darüber hinaus wurden Methoden zur Entwicklung von KIs, wie etwa das Planning vorgestellt. Den Abschluss der Grundlagen bildet ein Kapitel, welches sich damit beschäftigt was unter Emotionen zu verstehen ist und wie diese in einer KI nachgebildet werden können.
Nachdem die Grundlagen für das Agentensystem erläutert wurden, wurde eine Analyse durchgeführt um festzustellen, welche Anforderungen an einen emotionalen Agenten gestellt werden und welche Eigenschaften der Agent aufweisen muss, um diesen Anforderungen zu erfüllen. Im Zuge dieser Analyse wurde herausgearbeitet, dass moderne Spiele und Simulationen immer höhere Ansprüche an die Künstliche Intelligenz und deren Glaubwürdigkeit stellen.
Dies führt dazu, dass neue Methoden zur Entwicklung von KIs entworfen werden müssen, um die wachsende Komplexität der KIs handhabbar zu machen. Darüber hinaus besteht ein großes Interesse darin, KIs glaubwürdiger zu machen, indem deren Entscheidungen für den Betrachter nachvollziehbarer werden, was durch die Integration von Emotionen erreicht werden kann.
Anhand der Anforderungen, die in der Analyse herausgearbeitet wurden, wurde ein Design für das Agentensystem erstellt, welches es dem Agenten durch das GOAP erlaubt, zur Laufzeit dynamisch Entscheidungen über sein Verhalten zu treffen. Diese Entscheidungen werden anhand des aktuellen Zustands der Welt getroffen und beziehen zusätzlich den emotionalen Zustand des Agenten mit ein. Die Simulation der Emotionen wurde durch eine modifizierte Version des OCC-Modells realisiert und zusammen mit dem GOAP-System in das C4-Modell integriert.
In der Testphase wurde schließlich der entwickelte Prototyp in einem Marktplatzscenario validiert. Hierbei zeigte sich, dass die Agenten, wie gefordert, in der Lage sind, dynamisch zur Laufzeit ihr Verhalten der Umgebung anzupassen und dass die simulierten Emotionen das Verhalten der Agenten vielfältiger gestalten.
Jedoch wurde auch deutlich, dass die Verhaltensänderungen, die durch die Emotionen erzeugt werden, alleine nicht ausreichen, um die Agenten wirklich glaubhaft erscheinen zu lassen.

7.3 Ausblick

Für den hier entwickelte Prototyp bieten sich viele Möglichkeiten zur Verbesserung und Erweiterung. So wurde zum Beispiel bei der Umsetzung des Emotionsmodells aus Zeitgründen nur eine geringe Anzahl verschiedener Emotionen implementiert. Um die Glaubwürdigkeit der emotionalen Agenten zu erhöhen, könnte dieses Modell mit zusätzlichen Emotionen erweitert werden.
Auch das GOAP-System bietet noch Raum für Erweiterungen. Hierbei ist vor allem die Erweiterung des Planungsvorgangs für mehrere Agenten interessant, um es zu ermöglichen, dass komplexere Ziele von einer Gruppe von Agenten, in Zusammenarbeit, erreicht werden. Ein Ansatz hierfür wären beispielsweise Hierarchical Task-Networks (siehe [HMA06]).
Um die Entwicklung von emotionalen Agenten zu vereinfachen, wäre es darüber hinaus sinnvoll Tools zur Generierung von goals und actions zu erstellen. Auch die standards, goals and preferences, die die Basis für die Emotionen eines Agenten bilden, könnten durch Tools generiert werden.
Des Weiteren wäre es denkbar, die entwickelte KI mit einem Animationssystem zu verknüpfen, um die Aktionen und Emotionen eines Agenten für den Betrachter besser nachvollziehbar zu machen.
Mit einigen dieser Erweiterungen könnten die entwickelten emotionalen Agenten in einer Vielzahl von Szenarien eingesetzt werden, bei denen es darauf ankommt, glaubhafte autonom agierende Agenten zu simulieren. Solche Szenarien könnten etwa Simulationen im militärischen Bereich umfassen, in denen Soldaten trainieren können, wie Menschenmengen sich in unterschiedlichen Situationen und Kulturen verhalten.
Darüber hinaus sind Anwendungen in Computerspielen denkbar. Hier bietet sich etwa die Simulation von Menschenmengen in nichtlinearen Open World Games (siehe [Ope10]) an.

8 Anhang

8.1 Installationshinweise

1. Um das Programm ausführen zu können, muss Eclipse installiert sein. Sollte dies nicht der Fall sein, ist unter dem Ordner eclipse eine Version auf der CD mit beigelegt. Kopieren Sie diese an einen beliebigen Ort auf Ihrer Festplatte und starten Sie Eclipse.
2. Zunächst muss die JMonkeyEngine in Eclipse importiert werden. Dazu klicken Sie in Eclipse mit der rechten Maustaste in den Package Explorer und wählen Import -> General -> Existing Project into Workspace -> Browse und dort den *jme* Ordner auf der CD. Der Haken bei *Copy pojects into workspace* sollte gesetzt sein.
3. Jetzt muss das eigentliche Programm in Eclipse importiert werden. Dazu klicken Sie erneut in Eclipse mit der rechten Maustaste in den Package Explorer und wählen Import -> General -> Existing Project into Workspace -> Browse und dort den Ordner *Bachelorarbeit* auf der CD. Der Haken bei *Copy pojects into workspace* sollte gesetzt sein.
4. Sollten Fehler in dem *jme*-Projekt auftreten, müssen noch zusätzliche Umgebungsvariablen gesetzt werden. Dazu machen Sie einen Rechtsklick auf das *jme*-Projekt -> Properties, navigieren dort zum Java Build Path und wählen den Libraries tab. Expandieren Sie den *lwjgl.jar* node. Wählen Sie dann *native library location* und drücken Sie Edit. Wählen Sie jetzt ihren Workspace und wählen dort *jme/lib/lwjgl/native/<os>* wobei <os> für ihre Betriebssystem steht (Anwendung wurde jedoch nur unter Windows getestet).
   Wiederholen Sie diesen Vorgang für den *swt.jar* node, wobei Sie dieses Mal jedoch *jme/lib/swt/<os>* als native library location wählen.

8.2 Bedienung des Programms

Das Programm kann in Eclipse gestartet werden, indem die Datei BaseGame.java in dem Project Bachelorarbeit -> src -> Scenario ausgeführt wird. Dazu klicken Sie mit der rechten Maustaste auf diese Datei und wählen Run As -> Run Configurations. Dort können Sie unter dem Tab arguments zwei Argumente mitgeben. Um die Simulation mit Emotionen zu starten, muss als erstes Argument das Wort *emotions* mitgegeben werden. Mit dem zweiten Argument regeln Sie die maximale Anzahl der Agenten, die sich gleichzeitig auf dem Markt befinden können. Falls performance Probleme auftreten sollten, verringern Sie diese Zahl.

Nach dem Start des Programms ist es zunächst pausiert. Drücken sie *P*, um die Simulation zu starten.
Die Kamera kann mit der Maus und den Tasten *W,A,S,D* gesteuert werden.
Um Informationen über einen Agenten zu erhalten, klicken Sie mit der Maus auf diesen. Ist ein Agent ausgewählt, kann per *Leertaste* in den Verfolger-Modus gewechselt werden.
Mit *Escape* wird das Programm beendet. Die Zeit die, jeder Agent auf dem Markt verbracht hat, und dessen emotionaler Zustand kann nach Beendigung des Programms anhand der log.xls, welche im aktuellen Workspace angelegt wird, analysiert werden.

9 Glossar

WorldStateSymbol: Ein key/value Paar, welches den Zustand der Welt beschreibt

Emergenz: Emergence „is the process of deriving some new and coherent structes, patterns and properties (or behaviors) in a complex system"[Eng07] from a set of simpler structes, patterns and properties (or behaviors).

Engine: „Mit Engine (von engl. Antrieb, Motor) wird in der Informationstechnologie ein eigenständiger Teil eines Computerprogramms bezeichnet. Eine Engine ist oft für gewisse in der Regel komplexe Berechnungen oder Simulationen zuständig. Oft läuft eine Engine selbsttätig quasi im Hintergrund, ohne unmittelbar von einer Steuerung durch den Benutzer abhängig zu sein“ [Wik09a].

Immersion: „Immersion ist ein Bewusstseinszustand, bei dem der Betroffene auf Grund einer fesselnden und anspruchsvollen (künstlichen) Umgebung eine Verminderung der Wahrnehmung seiner eigenen Person erlebt. Damit beschreibt der Begriff „Immersion“ - ähnlich der filmischen Immersion - im Kontext der virtuellen Realität das Eintauchen in eine künstliche Welt“ [Wik09b].

Validierung: Prüfung der Entwicklungsergebnisse auf Erfüllung ihrer Spezifikationen.

Heuristik: Ein Lösungsverfahren, welches anhand von Abschätzungen schnell Ergebnisse liefert, die jedoch nicht 100%ig korrekt sein müssen.

10 Abkürzungsverzeichnis

KI	Künstliche Intelligenz
GOAP	Goal Oriented Action Planning
FSM	Finite State Machines, ((dt.)Zustandsautomaten)
HFSM	Hierarchical Finite State Machines, ((dt.)Hirarchische Zustandsautomaten)
STRIPS	Stanford Research Institute Problem Solver
GOB	Goal-Oriented Behavior

Literaturverzeichnis

[Bar02] BARTNECK, Christoph: Integrating the OCC Model of Emotions in Embodied Characters. In: *Proceedings of the Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges*, 2002

[Bat94] BATES, Joseph: The role of emotion in believable agents. In: *Commun. ACM* 37 (1994), Nr. 7, S. 122–125. http://dx.doi.org/http://doi.acm.org/10.1145/176789.176803. – DOI http://doi.acm.org/10.1145/176789.176803. – ISSN 0001–0782

[Coh07] COHEN, J: The GALE project: A description and an update. In: *Automatic Speech Recognition & Understanding* IEEE, 2007. – ISBN 978–1–4244–1746–9, S. 237–237

[Dor07] DORIS, D. K.; S. K.; Silvia: Improved Missile Route Planning and Targeting using Game-Based Computational Intelligence. In: *Computational Intelligence in Security and Defense Applications* (2007), April, Nr. 1-5, S. 63 – 68

[Eng07] ENGELBRECHT, Andries: *Computational Intelligence: An Introduction*. New York, NY, USA : Halsted Press, 2007. – ISBN 0470035617

[FN71] FIKES, Richard E. ; NILSSON, Nils J.: Strips: A new approach to the application of theorem proving to problem solving. In: *Artificial Intelligence* 2 (1971), Nr. 3-4, 189–208. http://dx.doi.org/10.1016/0004-3702(71)90010-5. – DOI 10.1016/0004–3702(71)90010–5

[Gef02] GEFFNER, Hector: Perspectives on artificial intelligence planning. In: *Eighteenth national conference on Artificial intelligence*. Menlo Park, CA, USA : American Association for Artificial Intelligence, 2002. – ISBN 0–262–51129–0, S. 1013–1023

[GHJJ96] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; JOHN, Vlissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl. Bonn : Addison-Wesley, 1996. – ISBN 3–89319–950–0. – Design Patterns, 1995, Deutsche Übersetzung von Dirk Riehle

[GM05] GRATCH, Jonathan ; MARSELLA, Stacy: Evaluating a Computational Model of Emotion. In: *Autonomous Agents and Multi-Agent Systems* 11 (2005), Nr. 1, S. 23–43. http://dx.doi.org/http://dx.doi.org/10.1007/s10458-005-1081-1. – DOI http://dx.doi.org/10.1007/s10458–005–1081–1. – ISSN 1387–2532

[Hig02] *Kapitel* 3.2. In: HIGGINS, Dan: *Generic A* Pathfinding*. Charles River Media, 2002, S. 114–121

[HMA06] *Kapitel* Coordinating Teams of Bots with Hierarchical Task Network Planning. In: HECTOR MUNOZ-AVILA, Hai H.: *AI Game Programming Wisdom 3*. Charles River Media, 2006

[Isl05] ISLA, Damian: Managing Complexity in the Halo 2 AI System. In: *Proceedings of the Game Developers Conference*, 2005

[JAD] *Java Agent Develeopment Framework*. http://sharon.cselt.it/projects/jade/. – [Online; Stand 22. November 2009]

[JMo10] *JMonkey Engine*. http://www.jmonkeyengine.com. Version: 2010. – [Online; Stand 12. Januar 2010]

[Mar09] MARK, Dave ; HURLEY, Heather (Hrsg.): *Behavioral Mathematics for Game AI*. Charles River Media, 2009. – ISBN 1–58450–684–9

[Mat02] MATHEWS, James: Basic A* Pathfinding Made Simple. In: RABIN, Steve (Hrsg.): *AI Game Programming Wisdom*. Charles River Media, 2002, Kapitel 3.1, S. 104–113

[McC] MCCARTHY, John: *WHAT IS ARTIFICIAL INTELLIGENCE?* http://www-formal.stanford.edu/jmc/whatisai/node1.html

[MG08] MORGADO, Luís ; GASPAR, Graça: Towards background emotion modeling for embodied virtual agents. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC : International Foundation for Autonomous Agents and Multiagent Systems, 2008. – ISBN 978–0–9817381–0–9, S. 175–182

[Mil06] MILLINGTON, Ian: *Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2006. – ISBN 0124977820

[MT10] MILES, Jere ; TASHAKKORI, Rahman: Improving believability of simulated characters. In: *J. Comput. Small Coll.* 25 (2010), Nr. 3, S. 32–39. – ISSN 1937–4771

[NH96] NWANA, Hyacinth S. ; HEATH, Martlesham: *Software Agents: An Overview*. 1996

[OCC88] ORTONY, Andrew ; CLORE, Gerald L. ; COLLINS, Allan: *The Cognitive Structure of Emotions*. Cambridge University Press, 1988 http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0521353645. – ISBN 0521353645

[Ope10] *Open World*. http://en.wikipedia.org/wiki/Open_world. Version: 2010. – [Online; Stand 16. Februar 2010]

[Ork] ORKIN, Jeff: *GAMES USING GOAP ARCHITECTURES*. http://web.media.mit.edu/~jorkin/goap.html

[Ork03] *Kapitel* 1. In: ORKIN, Jeff: *Applying Goal-Oriented Planning to Games*. first. Charles River Media, 2003

[Ork06] ORKIN, Jeff: Three States and a Plan: The A.I. of F.E.A.R. In: *Game Developers Conference 2006*, 2006

[OT90] ORTONY, A. ; TURNER, T. J.: What's basic about basic emotions? In: *Psychol Rev* 97 (1990), July, Nr. 3, 315–331. http://view.ncbi.nlm.nih.gov/pubmed/1669960. – ISSN 0033–295X

[PGZ08] PHILIP G. ZIMBARDO, Richard J. G.: *Psychologie*. 18. PEARSON STUDIUM, 2008. – 864 S. – ISBN 3827372755

[Plu] *Plutchik's Wheel of Emotions*. http://upload.wikimedia.org/wikipedia/commons/e/e9/Plutchik's_Wheel_of_Emotions.png. – [Online; Stand 07. November 2009]

[PMCA99] P., Rizzo ; M., Veloso ; CESTA A., Miceli M.: Goal-based personalities and social behaviour in believable agents. In: *Applied Artificial Intelligence Journal* 13 (1999), S. pp.239–271

[RB01] ROBERT BURKE, Marc Downie Yuri Ivanov Bruce B. Damian Isla I. Damian Isla: Creature smarts: The art and architecture of a virtual brain. In: *Proceedings of the Game Developers Conference*, 2001, S. 147–155

[Rey01] REYNOLDS, Craig: *Steering Behaviors: Autonomous Characters in Three Worlds*. http://www.research.scea.com/research/pdfs/CWR2001_04_23_UCB.pdf. Version: April 2001. – [Online; Stand 22. Dezember 2009]

[SA96] SCHMIDT-ATZERT, Lothar: *Lehrbuch der Emotionspsychologie*. Kohlhammer, 1996. – 18 S. – ISBN 3170118471

[SS07] STUART SLATER, Kevan B. Kamal Bechkoum B. Kamal Bechkoum: Foundation of Emotion Research for Games & Simulations. In: *AISB'07*, 2007

[TGB09] TASSE, Flora P. ; GLASS, Kevin ; BANGAY, Shaun: Simulating crowd phenomena in African markets. In: *AFRIGRAPH '09: Proceedings of the 6th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. New York, NY, USA : ACM, 2009. – ISBN 978–1–60558–428–7, S. 47–52

[VDPBB+06] VAN DYKE PARUNAK, H. ; BISSON, Robert ; BRUECKNER, Sven ; MATTHEWS, Robert ; SAUTER, John: A model of emotions for situated agents. In: *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–303–4, S. 993–995

[Wik09a] WIKIPEDIA: *Engine*. http://de.wikipedia.org/wiki/Engine. Version: 2009. – [Online; Stand 17. Dezember 2009]

[Wik09b] WIKIPEDIA: *Immersion (virtuelle Realität)*. http://de.wikipedia.org/wiki/Immersion_(virtuelle_Realität). Version: 2009. – [Online; Stand 20. Dezember 2009]

[Wik09c] WIKIPEDIA: *Phase Alternating Line*. http://de.wikipedia.org/wiki/Phase_Alternating_Line. Version: 2009. – [Online; Stand 15. Dezember 2009]

[WP09] WIESNER-POMMER, Andrea: *Einfluss genetischer Polymorphismen im DISC1 Gen auf kognitive Phaenotypen*, Klinik und Poliklinik fuer Psychiatrie und Psychotherapie der Ludwig-Maximilians-Universitaet Muenchen, Diss., 2009

[YSM08] YU, Han ; SHEN, Zhiqi ; MIAO, Chunyan: A goal-oriented development tool to automate the incorporation of intelligent agents into interactive digital media applications. In: *Comput. Entertain.* 6 (2008), Nr. 2, S. 1–15. http://dx.doi.org/http://doi.acm.org/10.1145/1371216.1371227. – DOI http://doi.acm.org/10.1145/1371216.1371227. – ISSN 1544–3574

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. Februar 2010	
Ort, Datum	Unterschrift