Entwurf und Implementierung einer Multi-Agentensimulation zur Steuerung virtueller Charaktere

Diplomarbeit

im Fachbereich Knowledge Engineering
Studiengang Informatik
der
Technischen Universität Darmstadt

Alexander Skrinjar

30.09.2008

Erstprüfer: Prof. Dr. Johannes Fürnkranz

Bearbeitungszeitraum:

01. April 2008 bis 30. September 2008

Alexander Skrinjar
Matrikelnummer: 1663551
Studiengang: Informatik

Diplomarbeit
Thema: „Entwurf und Implementierung einer Multi-Agenten Simulation zur Steuerung virtueller Charaktere“

Eingereicht: 30.09.2008

Prof. Dr. Johannes Fürnkranz
Fachgebiet Knowledge Engineering
Fachbereich Informatik
Technische Universität Darmstadt
Hochschulstraße 1
64289 Darmstadt

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum	Unterschrift

Kurzfassung

Die manuelle Steuerung einer großen Zahl von virtuellen Charakteren ist effizient nicht möglich. Automatisierte Methoden auf Basis von künstlicher Intelligenz bieten verschiedene Ansätze für dieses Problem. Diese Arbeit untersucht den Einsatz eines Verhaltensmodells zur Steuerung dieser Charaktere. Es wird eine Architektur vorgestellt, welche das Modell auf Funktionalität prüft, insbesondere den Vergleich zu Versuchen mit Regelsystemen. Dazu wurden die Möglichkeiten aktueller Software auf dem Gebiet geprüft und ein Plug-In für die Animationssoftware Autodesk Maya 8.5 implementiert. Es wird gezeigt, dass sich das BDI-Modell erweitern lässt, um die Problemstellung zu lösen.

Schlüsselwörter:

Crowd Simulation, Animation, Verhaltensmodell, Computergrafik, Filmcharakteren, autonome Agenten, Multi-Agenten Systeme, Gruppenverhalten, Künstliche Intelligenz, Künstliches Leben, Maya, MEL, C++

Abstract

Manual control of a high amount of virtual characters is feasibly not possible to accomplish. Methods for automatic simulation on basis of artificial intelligence provide different approaches to this problem. This thesis evaluates the utility of a behaviour model for control of these characters. The introduced architecture examines the functionality of the model, especially in regard to rule based systems. To this end the possibilities of actual implementation are studied and a plug in for Autodesk Maya 8.5 is implemented. It will be shown that an expanded BDI Model is sufficient for solving the problem.

Keywords:

Crowd Simulation, animation, behavioural models, computergenerated imagery, movie characters, autonomous agents, multi-agent systems, group behaviour, artificial intelligence, artificial life, Maya, MEL, C++

Für wen auch immer.

Er war immer für mich da.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACM SIGGRAPH	Association for Computing Machinery's Special Interest Group on Graphics and Interactive Techniques
API	application programming interface
BDI	belief-desire-intention
CGI	computer generated imagery
DG	dependecy graph
JPP	jungian personality profile
KI	Künstliche Intelligenz
MAS	Multi-Agenten-System
MASSIVE	Multiple Agent Simulation System in Virtual Environment
MBTI	Meyer-Briggs Type Indicator
MEL	Maya Embedded Language
NDM	natural decision making
OpenGL	Open Graphics Library
TCPI/IP	Transmission Controll Protocoll/Internet Protocoll

1 Einleitung

Multi-Agentensimulationen eignen sich hervorragend zur Lösung verteilter Probleme. Sie werden häufig dann eingesetzt, wenn voneinander unabhängige, aber in Beziehung stehende Komponenten eines dynamischen Systems untersucht werden müssen. Der Bereich der „Crowd Simulation“ ist ein solches System und wurde bisher über rein reaktive Ansätze beschrieben. Insbesondere im Bereich der Tricktechnik, in der Geschwindigkeit einer der wichtigsten Faktoren ist, werden diese Multi-Agentensysteme aufgrund ihrer oft hohen Simulationsdauer nur zögerlich eingesetzt. Häufig sind dies nur als separate Anwendungen programmierte Insellösungen und lassen sich nur mit erhöhtem Aufwand in vorhandene Arbeitsabläufe einbinden.

Diese Arbeit zeigt, dass trotz der Komplexität des Themas ein naiverer Ansatz zur Steuerung virtueller Charaktere durch die Verwendung eines BDI-Modells effizient möglich ist und dass dieses System zur Erzeugung völlig autonomer Agenten erweiterbar ist. Darüber hinaus wird eine Architektur vorgestellt, die die Vorteile einer ausgelagerten Software mit denen einer integrierten Anwendung als Plug In kombiniert.

1.1 Motivation

Eine Vielzahl von Fernseh- und Kinofilmen sowie Dokumentationen und Reportagen benötigt heutzutage computergestützte Bilderzeugung zur Darstellung von Prozessen, irrealen Szenerien, Verbildlichung von Simulationen und vielem mehr. Die wenigsten dieser Bilder lassen sich durch alternative Konzepte, wie Miniaturen oder Kameraeffekte, realisieren oder sind schlichtweg zu kostspielig oder unflexibel. Um diese Hürden zu umgehen, entstanden in den 1970er Jahren die ersten Versuche mit 2D- und 3D-Simulationen wie z.B. Future World (1976), dessen Drahtgitterdarstellung eines Gesichts ein Novum in der Tricktechnik war. Die Software dazu entstand an der University of Utah und wurde von den Studenten Edwin Cutmull (heute Präsident von Walt Disney Animation Studios und Pixar Animation Studios) und Fred Parke entwickelt. Bis zum heutigen Tage sind die Forschungen der Universitäten die Grundlage von Innovationen im Bereich der Tricktechnik. Neben dem Fachbereich der grafischen Datenverarbeitung werden auch Gebiete der künstlichen Intelligenz

und Physik sowie der Numerik und Berechenbarkeit gestreift, wie eindrücklich auf der jährlich stattfindenden ACM SIGGRAPH vorgestellt wird.

Zur Gestaltung dieser Effekte ist eine Vielzahl von Softwarepaketen auf dem Markt erhältlich. Für den Bereich der dreidimensionalen Gestaltung haben sich im professionellen Segment die Lösungen Maya und Studio Max, beide vom Hersteller Autodesk, durchgesetzt. In diesen Anwendungen lassen sich Objekte modellieren und in Szenen anordnen, animieren und letztlich rendern, d.h. die 3D-Objekte in eine zweidimensionale Form für die Darstellung auf einer Kinoleinwand oder dem Fernseher überführen (siehe Kapitel 4.3.1 für eine tiefergehende Beschreibung der Software Maya).

Ein spezieller Bereich dieser computergenerierten Bilderzeugung (computer generated imagery – cgi) beschäftigt sich mit der Simulation großer Menschenmengen, „Crowd Replication“, „Crowd Generation“ oder „Crowd Simulation“ genannt.

Crowd Simulation ist ein relativ neuer Zweig der Tricktechnik und umfasst viele Gebiete der Forschung. Neben der visuellen Komponente, der Darstellung und des Rendering einer großen Menge von 3D-Daten und der Animation der Charaktere ist die Simulation der Charaktere und damit die Auswahl der Animation ein wichtiger Aspekt. Da die manuelle Erzeugung und Animation von Tausenden von Charakteren immens aufwendig und kostspielig ist, wird in der Regel eine separate Software entworfen, welche dies automatisiert. Dadurch wird dem Benutzer eine große Zeit- und Arbeitsersparnis und letztendlich auch eine Kostenreduzierung ermöglicht.

Zum Zeitpunkt der Entstehung dieser Arbeit ist nur ein Softwarepaket erhältlich, welches die Vielzahl von Einsatzmöglichkeiten von Crowd Simulation abdeckt. Das special effects-Unternehmen Weta Digital entwickelte im Rahmen des Kinospielfilms „Herr der Ringe“ die Software MASSIVE, welche mittlerweile in vielen weiteren Filmen wie „Die Chroniken von Narnia“ oder „King Kong“ verwendet wurde.

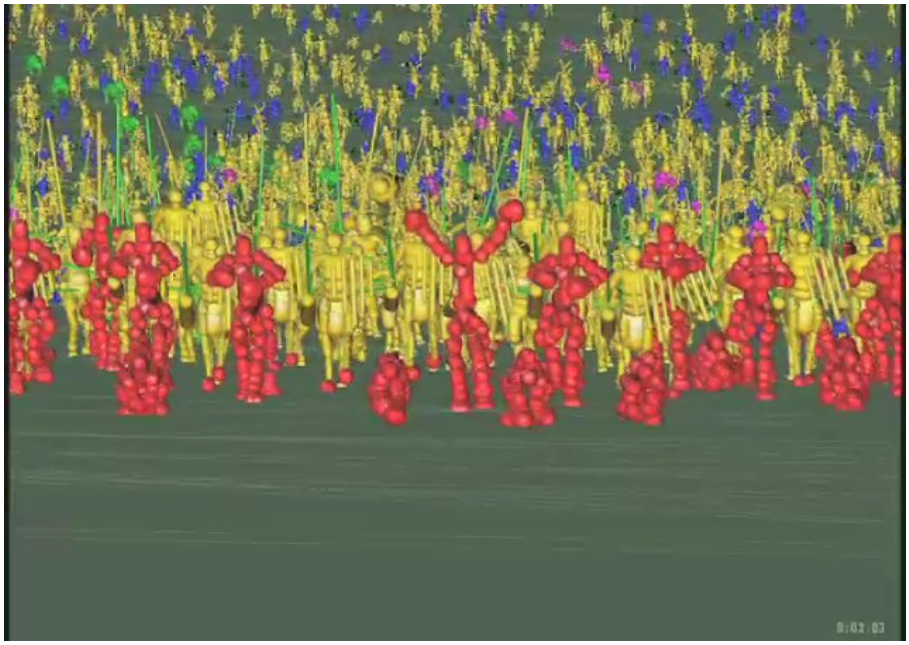


Abbildung 1: Massive Software: Simulation

Abbildung 2: Massive Software: Finaler Schuss

Mit MASSIVE lassen sich Verhaltensweisen festlegen, welche die Charaktere steuern. Jedoch liegt dieser Steuerung meist ein einfaches Regelwerk zugrunde und keine echte Intelligenz, da dies einen zu hohen Rechenaufwand zur Folge hätte. Dies zeigt sich auch in den repetitiven Verhaltensweisen der Charaktere.

Ein weiteres Manko aller bisherigen Implementierungen ist die fehlende Integration in eine bestehende Animationssoftware - es entsteht ein Bruch in der Produktionspipeline. Neben dem erhöhten Zeitaufwand des Exportierens und Importierens wird ein weiterer Spezialist benötigt, welcher sich mit der zusätzlichen Software auskennt.

Diese Arbeit soll neben der Optimierung des Arbeitsablaufs innerhalb der Produktionspipeline im Gegensatz zu bisherigen Entwürfen **[Kol00] [Pat05] [Dur07] [Lim04]** auch einen neuen Ansatzpunkt zur Verhaltenssimulation bei Animationssoftware und darüber hinaus untersuchen, wie in **[Tor03]** vorgestellt.

1.2 Aufgabenstellung

Ziel der Diplomarbeit ist die Entwicklung einer künstlichen Intelligenz zur Steuerung (Entscheiden und Handeln) von Agenten. Das System soll als Motor für virtuelle Charaktere in der 3D-Software Autodesk Maya Version 8.5 (nachfolgend Maya genannt) zur Visualisierung und Animation großer Mengen von Agenten dienen.

Die in dieser Arbeit entworfene Architektur hat daher zum Ziel, einen autonom handelnden Softwareagenten zu entwickeln, der, ähnlich einem simulierten Menschen, einzig auf Basis seiner Wahrnehmung, seiner Erfahrungen, seines Handlungsspielraums und nicht zuletzt seiner individuellen Neigungen eine Handlungsabsicht trifft. Die Arbeit bedient sich hierfür einiger psychologischer Modelle, welche am Prozess der menschlichen Entscheidungsfindung beteiligt sind.

Für eine Verhaltenssimulation werden mehrere Komponenten benötigt: Sensorik, Perzeption, Gedächtnis, Intention, Lokomotion, Planen, Emotionen, Interaktion, Sozialverhalten, um nur einige zu nennen. Diese Arbeit konzentriert sich auf den Aspekt der Intention. Es wird anhand von externen Daten (Umwelt) und internen Daten (Persönlichkeit) eine Entscheidung für eine Aktion getroffen. Dieser Vorgang soll simuliert werden. Für den Anwendungszweck der automatisierten Animation von Filmcharakteren ist dies ausreichend, wie in Kapitel 2 ausgeführt.

Das System muss folglich Umgebungsdaten sammeln und anhand dieser und der eigenen Disposition entscheiden, welche Handlung nachfolgend ausgeführt werden soll und wie die Parameter dieser Handlung zu bestimmen sind.

Alle Entscheidungen sollen dabei über ein Persönlichkeitsprofil bestimmt werden, welches sich über Parameter dynamisch anpassen lässt und somit den Wesenszug des Agenten modifiziert - sowohl in psychologischer (Angst, Freude, Gier etc.) als auch physiologischer (Sichtweite) Hinsicht. Das System muss skalierbar (Anzahl der Agenten) sein und Interaktion zwischen den Agenten berücksichtigen.

Es ist zu berücksichtigen, dass der Agent allein aufgrund seiner Eigenschaften eine Entscheidung trifft und kein Regelsystem oder Entscheidungsbaum mit vorgefertigten Handlungsweisen verwendet wird. Jede Situation wirkt sich in einer bestimmten Weise auf die Motivationen und Bedürfnisse des Agenten aus, auf deren Basis der Agent seine Entscheidungen trifft.

Am Ende der Arbeit steht eine Softwareimplementierung eines solchen Agenten, welcher in der Anwendung Autodesk Maya, innerhalb der die Welt und die zu steuernden Objekte modelliert sind, die Kontrolle über die virtuellen Charaktere übernimmt und diese selbständig in der Welt agieren lässt.

Ein typischer Einsatzzweck dieser Implementierung in einer Filmszene wäre beispielsweise die Kontrolle eines Zuschauers in einem Stadion oder eines Kriegers einer Schlacht. Dieser Agent würde seine Umgebung und die ihn umgebenden Nachbarn wahrnehmen und klassifizieren, sein Verhaltensmodell auswerten und anhand seines individuellen Charakters eine Handlung für den nächsten Simulationsschritt auswählen. Durch ausreichende Simulationsschritte lässt sich so das Verhalten eines Agenten für einen Filmeffekt automatisiert generieren.

1.3 Gliederung

Nachdem die Motivation ausführlich beschrieben wurde, wird im folgenden Kapitel 2 die Problemstellung auf spezifische Aspekte untersucht und ein Anforderungskatalog erstellt. Kapitel 3 beleuchtet die vorhandene Lösung der proprietären Software Massive und zeigt Unterschiede und Gemeinsamkeiten auf.

In Kapitel 4 werden Grundlagen für die durch den Anforderungskatalog vorgegebenen Kriterien und die in Kapitel 5 erarbeitete Architektur dargelegt. Die psychologi-

schen Grundlagen befassen sich mit der Erklärung von Verhaltens- und Persönlichkeitsmodellen, die Physiologie mit Modellen der Wahrnehmung. Des Weiteren sollen insbesondere die technischen Rahmenbedingungen erläutert werden, unter denen das KI-System implementiert werden soll.

Darauf aufbauend wird in Kapitel 5 eine Architektur entworfen, welche die Verhaltensmodelle in einer für eine Implementierung notwendigen Form aufbereitet. Die Implementierung und ihre Komponenten werden in Kapitel 6 beschrieben.

Kapitel 7 untersucht die Funktionalität der Implementierung und der Architektur und letztendlich die Tauglichkeit des Verhaltensmodells für den beabsichtigten Anwendungszweck. Abschließend wird in Kapitel 8 auf Erweiterungen und weitere Verwendungszwecke von Multi-Agenten-Systemen eingegangen.

1.4 Begriffsdefinitionen

Für das bessere Verständnis werden folgend einige Begriffe und ihre Bedeutung im Zusammenhang mit dieser Arbeit erläutert.

Agent / Charakter

Ziel der Arbeit ist das Steuern von virtuellen Lebewesen. Diese Objekte werden Agenten (in Anlehnung an das Konzept des Softwareagenten und der Definition des Wortes Agent in **[Rus95])** oder eben Charaktere (in Anlehnung an Figuren einer Geschichte) genannt. Der Begriff Agent wird häufiger auf der Seite der technischen Realisation genannt, während der Begriff Charakter häufiger die Funktion beschreibt. Jedoch sind die Begriffe wechselseitig austauschbar.

Intention

Der Begriff Intention stammt aus dem in dieser Arbeit verwendeten BDI-Modell (siehe Kapitel 4.1.2) und beschreibt die gewählte Aktion des Agenten, welche direkt ausgeführt wird.

Desire / Bedürfnis / Motivation

Wird von Bedürfnissen gesprochen, so ist die Motivation eines Agenten für die Wahl einer Intention gemeint. Hier ist der Begriff durch sein englisches Pendant desire austauschbar. Es ist die übliche Bezeichnung, hervorgerufen durch die Verwendung innerhalb des BDI-Modells (siehe Kapitel 4.1.2). Auch hier sind die Begriffe wechselseitig austauschbar.

Verhalten

Das Verhalten bezeichnet die räumliche Ausprägung der Intention. Die Arbeit entwirft ein Modell, welches den Agenten eine Tätigkeit auswählen und diese auch in der virtuellen Welt (innerhalb der Software Maya) umsetzen lässt. Maya bietet dazu die Möglichkeit der Animation und der räumlichen Position und Orientierung des Charakters. Das Setzen dieser Parameter beschreibt ein Verhalten.

Wissen / Knowledge Database

Das Wissen eines Agenten ist definiert durch die Situationen, die er wahrgenommen hat. Sie werden in einer sogenannten Wissensdatenbank (Knowledge Database) gespeichert.

2 Problemanalyse und Anforderungsspezifikation

Für die Steuerung der virtuellen Charaktere muss ein Verhaltensmodell entwickelt werden. Sie müssen innerhalb von Maya die virtuelle Welt wahrnehmen und daraufhin eine Entscheidung treffen. Die Welt setzt sich aus dreidimensionalen Objekten zusammen, welche in Maya erstellt werden. Die getroffene Entscheidung oder auch Intention beschreibt eine Aktion, welche den Agenten in der virtuellen Welt steuert (siehe Abbildung 3).

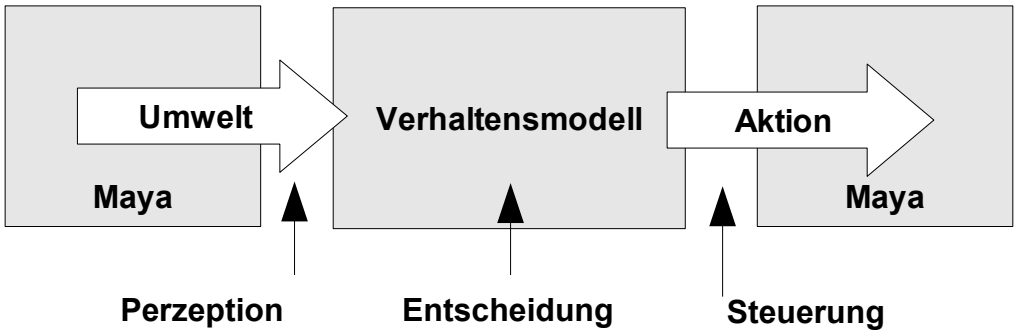


Abbildung 3: Problemstellung

In diesem Kapitel soll analysiert werden, welche Funktionen das Verhaltensmodell für eine natürliche Entscheidung (natural decision making - NDM) bereitstellen und welchen Anforderungen es genügen muss.

Neben den funktionalen Bestandteilen muss sich das System für die Implementierung als Plug In eines technischen Rahmens bedienen. Zu diesem gehören nicht nur die Voraussetzungen für eine Integration für Maya, sondern insbesondere auch die Laufzeit der Simulation.

2.1 Funktionale Anforderungen an den Agenten

Bei einer Crowd Simulation müssen verschiedene, auch widersprüchliche, Aspekte miteinander verknüpft werden. Der Agent muss autonom handeln, jedoch für den Anwendungszweck der Computeranimation eine bestimmte Handlung durchführen. Denn die Szene wird in Aufbau und Gestaltung, mit mehr oder weniger Freiheitsgraden, durch den Regisseur vorgegeben. Im Unterschied zu den meisten Agentensimulationen, bei denen der Ausgang nicht bekannt ist oder nach einem Optimum, wie bei künstlicher Intelligenz in Computerspielen, gesucht wird, soll hier durch ein KI-System nur die Automatisierung und Variation für ein vorgegebenes Ziel erreicht werden.

Um aber dieses gewisse Maß an Variation zu erreichen und insbesondere die Interaktion zwischen mehreren Charakteren abzugleichen, ist der Einsatz eines Regelsystems nicht zwingend optimal. Zwar wird bei einem Regelsystem das Verhalten modelliert anstatt der Agent, aber es muss jede mögliche Situation vorhergesehen und betrachtet werden, um ein natürliches Verhalten des Agenten zu erwirken.

Um das Verhalten der Agenten beeinflussen und anpassen zu können, muss der Agent durch Parameter so konfigurierbar sein, dass aufgrund gleicher Situation und Hintergrundwissen eine andere Entscheidung getroffen wird. Ein häufig auftretendes Szenario für Crowd Simulation ist die Bewegung einer Gruppe von Agenten von einem Punkt A zu einem Punkt B. Sollte zwischen Punkt A und B ein Gefahrenmoment bestehen, so sollen die Agenten entsprechend ihrer Möglichkeiten (Aktionen) und ihrer Persönlichkeit (beispielsweise „mutig“) eine Entscheidung treffen, der Gefahr zu trotzen oder den Rückzug anzutreten. Durch eine Anpassung des Agententypen kann man die Wahrscheinlichkeit einer Aktion (hier: Rückzug oder Kampf) beeinflussen und so mehr oder weniger Agenten kämpfen oder fliehen lassen.

Ein möglicher Lernalgorithmus zur Bestimmung von Verhalten und Parametern würde gegen die Anforderung der Vorhersagbarkeit verstoßen, da nicht ein gewolltes, sondern ein optimales[1] Verhalten erzielt wird. Darüber hinaus ergäbe sich eine zeitliche Verzögerung, bis das Ergebnis vorliegt, welche unter allen Umständen vermieden werden soll. Eine Möglichkeit wäre das durch den Regisseur vorgegebene Verhalten als Ziel zu definieren und zu erlernen, doch auch hier ergäbe sich eine nicht notwendige zeitliche Verzögerung. Durch die häufig sehr unterschiedlichen Zielset-

[1] Optimal in Bezug auf den eingesetzten Lernalgorithmus

zungen bietet auch ein Offlin-Learning-Verfahren keine Lösung. Neben dem Agenten muss auch das System durch den Nutzer so konfigurierbar sein, dass es das gewünschte Verhalten simuliert.

Für die Umsetzung der geforderten Funktionalität werden für den Agenten vier Komponenten benötigt.

1. Sensorik

Um das Verhalten des Agenten zu bestimmen, werden Schnittstellen benötigt, über die der Agent mit der Umgebung interagieren kann. Die Sensorik muss innerhalb der virtuellen Umgebung, welche durch die Anwendung erzeugt wurde, Objekte wahrnehmen und ihre Eigenschaften identifizieren können. Die Wahrnehmung des Agenten beschränkt sich auf die visuellen Komponenten, da Maya nur eine visuelle Auswertung erlaubt. Jedoch soll eine Erweiterung um eine zum Beispiel auditive Schnittstelle in Betracht gezogen und berücksichtigt werden. Dieses Perzept muss für die Auswertung und in Form einer Gedächtnisfunktion für spätere Verwendung vorgehalten werden. Wie beim menschlichen Organismus muss eine Zeitfunktion die Wertig- und damit Nützlichkeit einer Information bewerten. Aufgrund der so gewonnenen Daten kann der Agent eine Wissensdatenbank aufbauen und aufgrund dieser und der aktuell wahrgenommenen Situation passende Entscheidungen treffen.

2. Persönlichkeitsmodell

Für die Entscheidungsfindung wird neben externen Umweltdaten der interne Status des Agenten in Betracht gezogen. Dieser Status besteht aus einer Struktur, welche Wesenszüge für den Agenten beschreibt. Wie bei einem Menschen wird die Persönlichkeit eines Agenten durch eine Wertedisposition bestimmt: Wie wichtig sind einem Agenten bestimmte Bedürfnisse, wie ablehnend steht er bestimmten Handlungen gegenüber. Durch die Variation der Persönlichkeit soll eine Veränderung des Verhaltens erwirkt werden, da der Agent bei der Bewertung der Handlungsmöglichkeiten während des Entscheidungsprozesses die Faktoren, die die Entscheidung determinieren, anders bewertet. Das Modell für diese Persönlichkeit muss eine Vergleichsfunktion bieten, mit deren Hilfe man Aktionen für bestimmte Agententypen prädestinieren kann. Dadurch wird die geforderte Streuung des Verhaltens erreicht, da, falls gewollt, jeder Typ einen anderen Aktionspool besitzt.

3. Verhaltensmodell

Das Verhaltensmodell ist der Kern des Systems. Es muss aufgrund der gesammelten Daten aus Sensorik und Persönlichkeit eine Aktion auswählen. Der Agent muss eigenständig aufgrund seiner Bedürfnisse agieren. Der Einfluss des Benutzers beschränkt sich daher auf die Modifizierung der Bedürfnisse oder des Profils, um eine Verhaltensänderung zu erwirken. In Anlehnung an psychologische Modelle, welche das Verhalten von Menschen allgemeingültig zu beschreiben versuchen, wird auch nur ein Verhaltensmodell erarbeitet, welches durch die Modulation der Persönlichkeit (siehe Persönlichkeitsmodell) im Ergebnis variiert. Dem zugrunde liegt der Gedanke, dass jeder Mensch auf die gleiche Weise Entscheidungen trifft und nur die Bewertung des Wissens (Erinnerungen, Wertvorstellungen, Meinungen, Moral etc.) hinsichtlich der Bedeutung (Was ist mir wie wichtig) des Menschen zu unterschiedlichen Verhaltensweisen führt.

4. Lokomotion

Sobald das Verhaltensmodell eine Intention und damit die Aktion des Agenten bestimmt hat, wird die Funktion ausgewertet und in Bewegungsdaten für den Agenten umgesetzt.

2.2 Technische Rahmenbedingungen

Das System ist auf einer Microsoft Windows XP-Plattform zu implementieren. Ein Ausweichen auf ein anderes Betriebssystem muss aber bei der Programmierung berücksichtigt werden und daher dürfen keine exklusiven Technologien, wie Microsoft DirectX, verwendet werden. Das zu entwerfende Programm soll als Modul in die 3D-Animationssoftware Maya integriert werden. Maya ist für 32-Bit Microsoft Windows ®, Linux ®, und Max OS ® X sowie für 64-Bit Windows und Linux verfügbar. Als Hardware-Voraussetzungen werden mindestens 2GB Arbeitsspeicher, 2GB Festplattenspeicher und eine Grafikkarte mit OpenGL®[2]-Unterstützung empfohlen. Somit ist Maya auf heute handelsüblichen Desktop-Systemen lauffähig. Das Rendering wird im professionellen Einsatz in der Regel über eine sogenannte Renderfarm bewerkstelligt. Eine Renderfarm ist ein Cluster von mehreren Rechnern meist durchschnittlicher Ausstattung, welche als Verbund an der Berechnung eines Bildes oder einer Animation arbeiten. Dadurch wird nicht nur im Gegensatz zu einem leistungsstarken Einzelrechner die Rechenleistung erhöht (durch parallele Bearbeitung), sondern

[2] API für hardwareunterstützte 3D Grafik - http://www.opengl.org

auch Ausfallsicherheit gewährleistet (Ausfall einzelner Knoten des Clusters) und eine am Arbeitsaufwand gemessene Skalierung ermöglicht (Hinzufügen und Entfernen von Knoten). Auf einer leistungsfähigen Arbeitsstation wird mit Maya gearbeitet und die fertigen Daten zum Berechnen der Bildsequenz werden an den Cluster gesendet. Somit bleiben der Arbeitsstation die lokalen Ressourcen für die Darstellung der 3D-Daten, welche in der Regel auch voll ausgeschöpft werden. Diese Ressourcenverteilung muss im Entwurf und der anschließenden Implementierung berücksichtigt werden. Insbesondere ist darauf zu achten, dass das System die normale Arbeit an der Arbeitsstation nicht stört.

Entscheidend für eine praktikable Nutzung ist neben der Funktionalität die Laufzeit der Anwendung. Grundsätzlich ist jede Berechnungsdauer, welche kleiner als die manuelle Erstellung der Animationen ist, ausreichend. Damit wäre das Ziel der Zeit- und damit Kostenersparnis bedient. Aufgrund der möglichen hohen Zahl von Agenten sollte darauf geachtet werden, dass eine Komplexität unter $O(n^2)$ angestrebt wird, um zeitnah, optimalerweise in Echtzeit, mit Ergebnissen rechnen zu können. Aufgrund der hohen Ansprüche an die visuelle Qualität der grafischen Elemente ist die Option einer Berechnung der Simulation mit verminderter Qualität notwendig.

Als Schnittstelle bietet Maya die proprietäre Scriptsprache MEL und eine C++ API an. Es muss erprobt werden, welche Lösung sich als effizienter, sowohl in Hinblick auf Entwurf und Konfiguration als auch auf die Laufzeit, erweist (siehe Kapitel 4.3.1). Die API ist auf ein prozedurales Vorgehen hin optimiert. Ihre Funktionen und Freiheiten sind auf Modifikationen von Objekten ausgelegt. Die Eingabedaten werden durch eine Funktion berechnet und eine Ausgabe generiert. Es gibt keine direkte Möglichkeit, berechnete Daten im Speicher zu halten bzw. globale Objekte zu verwalten. Es ist vom Entwickler empfohlen, dass jede Node in sich abgeschlossen (self-contained node) ist. Für die angestrebte Funktionalität muss der Agent aber auf alle bisher gesammelten Daten Zugriff erhalten.

Eine völlige Integration in Maya erscheint daher als nicht optimal, da die Beschränkungen der API sehr restriktiv ausfallen. Es bietet sich an, das KI-System als Client-Server-Architektur zu entwerfen und somit die Berechnung außerhalb von Maya durchzuführen. Dadurch erreicht man nicht nur eine Entkoppelung des Systems für andere Einsatzzwecke, sondern kann zugleich die Ressourcen der Arbeitsstation schonen, indem das System auf einem externen Rechner oder Cluster läuft.

Jedoch muss die Simulation des KI-Systems selbst innerhalb von Maya erfolgen, um die Ergebnisse weiter zu verarbeiten. Maya selbst bietet dazu keine Unterstützung. Ein Objekt besteht aus seinen räumlichen Daten und Parametern, wie Position und Rotation. Die Veränderung des Zustandes und damit der Parameter eines Agenten ist über zwei Wege möglich.

Ereignis System:

Maya ist eine nodebasierte Animationssoftware (siehe Kapitel 4.3.1). Jeder Parameter einer Node lässt sich als Auslöser für Prozesse bestimmen. Sobald sich der Parameterwert ändert, werden ein Ereignis ausgelöst und ein Vorgang angestoßen. Ein alternierender Parameter kann somit zu jedem beliebigen Zeitpunkt t eine Neuberechnung des jeweiligen Agenten auslösen.

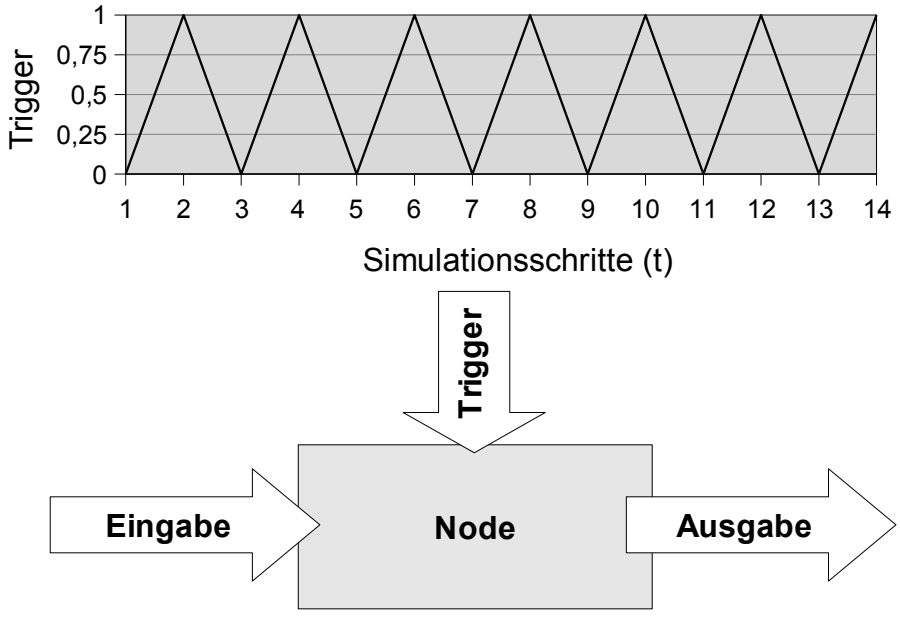


Abbildung 4: nodebasiertes Ereignis System

Schlüsselbildanimation:

Maya bietet ein Animationsmodul, welches die Parameter ***p*** eines Objekts für einen bestimmten Zeitpunkt ***t*** sichern kann. Die Animationsgeschwindigkeit wird in Schritten ***t*** gemessen und definiert sich über die Anzahl der Schritte pro Sekunde[3]. Die Animation ergibt sich folglich aus den Veränderungen der Werte ***p*** über ***t*** (siehe dazu Abb. 5). Dieses Verfahren wird Schlüsselbildanimation genannt. Die fixen Zeitpunkte ***t*** mit Änderungen bilden die sogenannten Schlüsselbilder. Die Werte für die Zeitpunkte t_i zwischen zwei Schlüsselbildern k_1 und k_2 werden interpoliert und somit wird eine flüssige Animation erzielt.

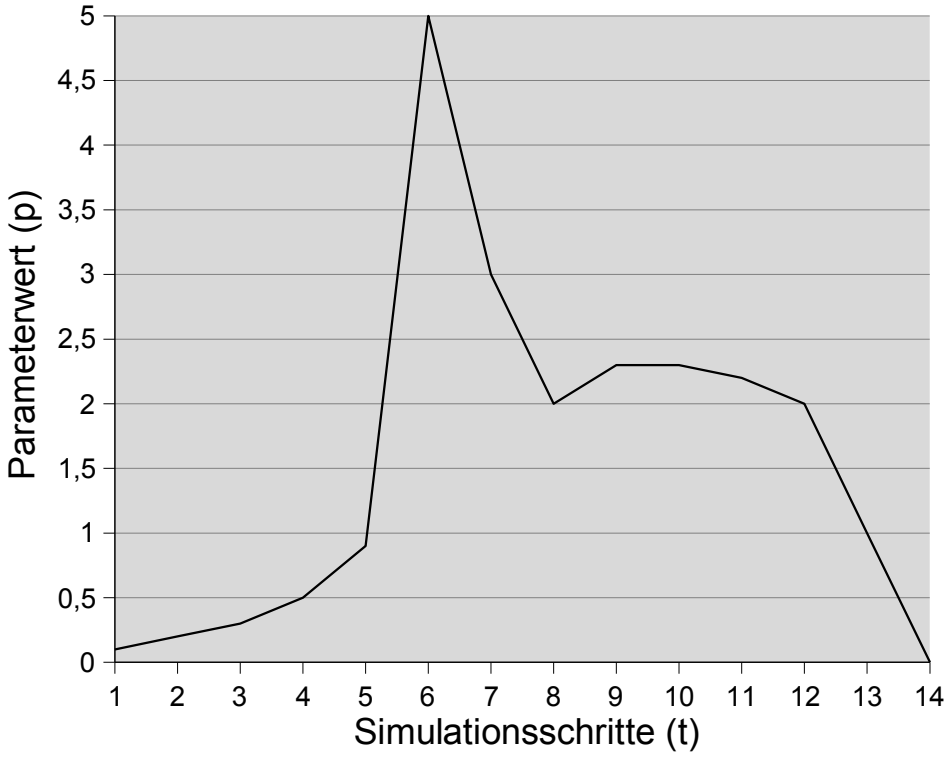


Abbildung 5: Veränderung von ***p*** über ***t***

Folglich lässt sich für jeden Zeitpunkt ***t*** eine Abfrage für eine Erneuerung des Zustands jedes Agenten erwirken. Der neue Zustand $Z_{(t+1)}$ ergibt sich aus der Berechnungsvorschrift ***F*** des KI Systems und dem alten Zustand $Z_{(t)}$:

$$Z_{(t+1)} = F(Z_{(t)})$$

[3] Für eine Verwendung in einem Spielfilm werden in der Regel 24 Bilder/Sekunde benötigt

3 Stand der Technik

Es besteht eine Vielzahl von Werkzeugen zur Simulation von natürlichen Entscheidungen. Diese „natural decision making“ genannten Systeme (natürliche Entscheidungen treffende Systeme – NDM) sind jedoch oft nur im beschränkten Maße für den Einsatz in der Tricktechnik einsetzbar, da das Verhalten nur schwer vorhersagbar ist oder die Daten nur schwer in eine Animationssoftware importiert werden können. Gerade die Integration der NDM-Systeme in eine andere Animationssoftware ist nicht vorgesehen, sodass die gewonnenen Daten nur schwer für die Nutzung in Maya aufbereitet werden können. Des Weiteren müsste man die Gestaltung der Szene, welche in der Regel in Maya erfolgt, in das NDM-System integrieren, sodass der Agent darauf reagieren kann.

Das heutzutage einzige System, welches alle Aspekte betrachtet, ist MASSIVE[4] von Massive Software.

3.1 Massive Software

MASSIVE ist eine KI Middleware. Dies ist eine eigenständige Entwicklungsumgebung zur Erstellung, Simulation und Visualisierung von KI-Systemen. Durch die Verwendung, Konfiguration und Verknüpfung bestehender Komponenten können KI-Netzwerke aufgebaut werden, welche das Verhalten des Agenten determinieren. Dieses Verhaltensmodell muss anschließend auf eine beliebige Gruppe von Agenten übertragen werden.

Das Hauptaugenmerk von MASSIVE ist aber die Simulation von Physik und Animation für die Agenten, welche prozedural erzeugt wird und so auf die jeweilige Situation angepasst ist. Das sogenannte Brain Modul bestimmt schließlich, welche der Entscheidungen aus dem Animationspool gewählt wird. Dies wird in der Regel über if-then-Befehle realisiert, indem der Nutzer dem Agenten vorgibt, welche Aktion er in welcher Situation verwenden soll; der Agent hat jedoch zu keiner Zeit die Möglichkeit, selbst zu bestimmen, welche Aktion für ihn die beste wäre.

[4] MASSIVE Software – Artificial Life Solutions http://www.massivesoftware.com

MASSIVE verfolgt das Ziel, in Zukunft völlig autonome Agenten zu generieren, welche sogar Neben- oder Hauptrollen in Spielfilmen übernehmen können. Inwieweit sich dies als realisierbar erweist, bleibt abzuwarten, jedoch wird die jetzige Evolution der Software schon in vielen weiteren Bereichen genutzt: In der Architektur als Personenflussmodell, in der Robotik zur Steuerung der Effektoren, in Computerspielen, Transportwesen und zur Studie menschlichen Verhaltens.

Während das statisch determinierte Verhalten in einer Filmszene ausreichend und unter Umständen auch gewünscht ist, wird mit dieser Arbeit ein anderer Ansatz verfolgt. Anders als in MASSIVE entscheidet der Agent selbst, welche seiner Aktionen er, in Abhängigkeit seiner Bedürfnisse, einsetzt Dadurch wird nicht zwangsläufig ein bestimmtes Verhalten erzielt, jedoch ist der Agent völlig autonom in seinem Verhalten. Der Agent simuliert eine eigene Motivation und wird diese verfolgen.

Die Simulation und Visualisierung (Rendering) der Agenten erfolgt ausschließlich in MASSIVE. Daher müssen Objekteigenschaften wie die Simulation von Textilien, Haaren und ähnlichem ebenfalls in MASSIVE implementiert sein. Die Integrierung solcher Eigenschaften hängt i.d.R. der von spezialisierter Animationssoftware wie Maya hinterher und ist so redundant vorhanden. In der Produktion von Trickeffekten werden solche „Brüche“ in der Produktionslinie nur ungern vorgenommen. Dies hat eine Verteilung der Daten und langwierige Import- und Export-Vorgänge zur Folge, um den Effekt durch verschiedene Software zu „schleusen“. Es wird bevorzugt, den Effekt möglichst in einer Software zu generieren.

3.2 Andere Implementierungen

Neben MASSIVE ist im Bereich der Tricktechnik nur noch die Software AI Implant[5] weiter bekannt, welche aber seit der Übernahme durch Persagis den Fokus auf die Verwendung in Computerspielen legt. Auch hier liegt das Problem nicht in der Simulation der Agenten, sondern in der Integration der gewonnenen Daten in die restliche Produktion.

Bei größeren Post-Produktionsunternehmen wird auch im Unternehmen Software zur Simulation von Agenten entwickelt, welche mehr oder weniger für einen Film oder Effekt spezialisiert ist. Ein Beispiel hierfür ist ALICE (Artificial Life Crowd Engine) der

[5] Presagis AI Implant – http://www.presagis.com

Moving Picture Company[6]. Hier wurde die Simulation über ein PlugIn direkt in Maya integriert, jedoch basiert die Entscheidung der Agenten auf einem simplen Regelwerk (ähnlich dem Flocking-Verhalten von Craig Reynolds [7]). Des Weiteren wird die Simulation vollständig in Maya vorgenommen, was die Simulationsgeschwindigkeit negativ beeinflusst. Solche intern entwickelte Software ist i.d.R. anderen Unternehmen nicht zugänglich und so sind Details der Implementierung und der tatsächlichen Leistungsfähigkeit weitestgehend unbekannt.

[6] The Moving Picture Company - http://www.moving-picture.com

[7] Für weitere Informationen - http://www.red3d.com/cwr/

4 Grundlagen

4.1 Psychologie

Es gibt eine Vielzahl von verschiedenen Schulen und Modellen zur Beschreibung menschlichen Verhaltens und Typisierung der Persönlichkeit. Aufgrund fehlender Kenntnisse der Struktur und Funktionsweise des zugrundeliegenden Systems, dem menschlichen Gehirn, kann eine Beschreibung nur durch Observation des Verhaltens untersucht werden. Wegen der oft nicht oder schwierig zu formalisierenden Modelle der Psychologie muss man sich auf die (Modelle) beschränken, welche ein Reiz-Reaktions-Schema beschreiben. Diese können ohne weitere Anpassungen als Eingabe-Verarbeitung-Ausgabe-System in einer Software modelliert werden. Daher bedient man sich noch immer dem behavioristischen Paradigma. Das menschliche Gehirn wird hierbei als sogenannte „Black Box“ **[Wat19]** aufgefasst und eben ein solches „Stimulus-Response“ (S-R)-Schema definiert. Externe oder interne Stimuli lösen (triggern) dabei psychologische bzw. kognitive Prozesse aus, welche sich (noch) nicht durch naturwissenschaftliche Methoden beschreiben lassen und somit auch nicht in Betracht gezogen werden können. Messtechnisch erfassbar sind nur der Reiz und die Reaktion.

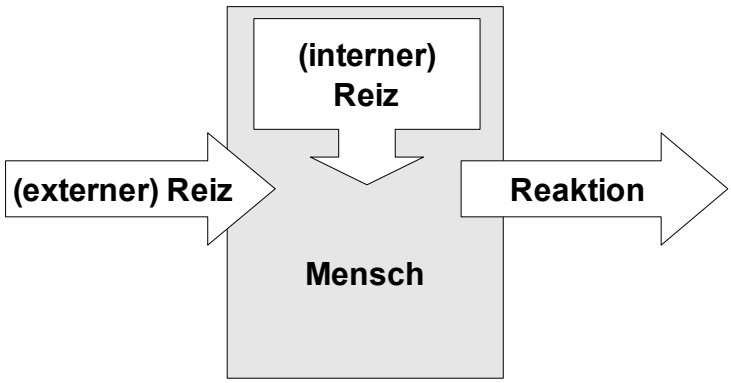


Abbildung 6: Stimulus-Reaktion

Obwohl der Behaviorismus mittlerweile durch den Kognitivismus als vorherrschendes Paradigma der Psychologie abgelöst wurde, bietet das Modell der „Black Box“ eine ausbaufähige Grundlage zur Definition eines Verhaltensmodells.

Im Nachfolgenden werden zwei Modelle vorgestellt: der Meyer Briggs Type Indicator, welcher die Kategorisierung menschlicher Persönlichkeit umfasst und das Belief-Desire-Intention-Modell, welches die Entscheidungsfindung beim Menschen beschreibt.

4.1.1 Persönlichkeit – Meyer Briggs Type Indicator

Die Persönlichkeit eines Menschen ist maßgeblich an dessen Entscheidungen beteiligt. Sie bestimmt nicht nur die Art der Handlung, sondern auch deren Intensität und Ausprägung. Wie ein Filter wird aus einem Pool von möglichen Verhaltensweisen eine zur Situation und zum Charakter passende Aktion ausgewählt.

Um dem virtuellen Agenten seine Wesenszüge mitzugeben, anhand derer der Filter operieren kann, wird eine Kategorisierung der unterschiedlichen Typen von Charakteren benötigt. Zu diesem Zweck wurden seit der Studie der menschlichen Persönlichkeit verschiedene Modelle entwickelt. Eines der bekanntesten ist, neben dem von Sigmund Freud, jenes von Carl Gustav Jung. Jung ging davon aus, dass jeder Mensch angeborene Tendenzen und Präferenzen besitzt, welche sein Verhalten beeinflussen.

Jungs Arbeit „Psychologische Typen“**[Jun21]**, auch bekannt unter „Jung'sches Persönlichkeitsprofil“ (Jungian Personality Profile), kategorisierte den Menschen anhand von einem Einstellungstypus und zwei Bewusstseinsfunktionen:

Introversion (I):

Ausrichtung auf die innere, subjektive Welt. Die Ziele werden durch innere Motive festgelegt. Es wird versucht, die Umwelt nach der eigenen Vorstellung zu formen. Introvertierte Menschen neigen dazu, Situationen nüchtern zu betrachten und lassen sich selten von der Umgebung leiten.

Extraversion (E):

Ausrichtung auf die äußere, objektive Welt. Ausgezeichnet durch einen hohen Grad an Beeinflussung durch und Anpassung an die Umwelt.

Jeder dieser Einstellungstypen besitzt vier Funktionstypen – **Denken (T), Fühlen (F), Empfinden (S) und Intuition (N)**. Durch Kombination ergeben sich acht verschiedene Typen von Persönlichkeiten.

Einstellung	Funktion	Persönlichkeitstyp
I	**T**	introvertiertes Denken Verfolgen ihre eigenen Ideen und lassen sich durch nichts, erst recht nicht von anderen Menschen, dabei stören.
I	**F**	introvertiertes Fühlen Meist Still und schwer zugänglich. Lassen sich überwiegend von ihrem Gefühl leiten und wirken planlos und mysteriös.
I	**S**	introvertiertes Empfinden Orientiert sich an der subjektiven Wirkung objektiver Reize, wirken daher oft leichtgläubig und passiv.
I	**N**	introvertierte Inutition Mystischer Träumer und Seher, Phantast und Künstler. Lebt in seiner eigenen Welt und verschließt sich fast vollkommen vor der Wirklichkeit.
E	**T**	extravertiertes Denken Pragmatisch und gefühlsarm, aber auch fair. Sehr von der Realität beeinflusst und auf Tatsachen hin ausgerichtet.
E	**F**	extravertiertes Fühlen Beruht auf festen Werten in der Welt und in Verfolgung dieser. Tugendhaft und Anständig.
E	**S**	extravertiertes Empfinden Extremer Realist mit ausgeprägtem Tatsachensinn. Legt wenig Wert auf Erfahrungen und mehr auf die akute Wahrnehmung. Im Gegensatz zum introvertiertem Typus nur auf den objektiven Reiz hin ausgerichtet.
E	**N**	extravertierte Inutition Ähnlich Umgebungsabhängig wie alle extravertierten Typen, jedoch mit Sicht auf Zukünftiges und Möglichkeiten. Unter Umständen sehr Sprunghaft.

Tabelle 1: Jung'sche Persönlichkeitsprofile

Aufbauend auf dem Modell von Jung entwickelten Katherine Briggs und Isabel Meyers den Meyer-Briggs-Typindikator (MBTI) und erweiterten das bestehende Konzept auf vier Dimensionen. Die vier Dichotomien sind:

1. Bewusstseinsfunktion: Introversion (I)/Extraversion (E)

Diese Funktion deckt sich weitestgehend mit den Definitionen von Jung.

2. Wahrnehmungsfunktion: Intuition (N)/Sensorik (S)

Beschreibt die Perzeption der Umwelt. Der sensorische Typus ist objektiv und nimmt die aktuelle Situation als solche wahr, während der intuitive Typus eine subjektive Wahrnehmung präferiert. Objekten werden Bedeutungen zugewiesen und diese in Zusammenhang zu bisherigen Erlebnissen gestellt.

3. Beurteilungsfunktion: Fühlen (F)/Denken (T)

Der Denker geht bei einer Aktionsselektion analytisch vor und wägt alle Faktoren gegeneinander ab. Die zu wählenden Aktionen werden auf das gewünschte Resultat hin ausgewählt. Der fühlende Typ geht nicht nur auf die für das Resultat entscheidenden Faktoren ein, sondern auch auf ihre Nebeneffekte, insbesondere auf andere Individuen.

4. Orientierungsfunktion: Wahrnehmung (P)/Entschlossenheit (J)

Die Orientierungsfunktion beschreibt die Beeinflussbarkeit eines Individuums durch die Umwelt. Entschlossene Menschen versuchen die Umwelt an ihre Wünsche und Vorstellungen anzupassen, während der wahrnehmende Typus eher auf Adaption und Flexibilität Wert legt.

Um einem Menschen die entsprechenden Funktionen zuordnen zu können, wird ein Fragebogen erstellt, den der Proband ausfüllen muss [8].

[8] Ein Beispiel für einen solchen Fragebogen findet sich unter: http://www.humanmetrics.com/cgi-win/JTypes2.asp

Auch hier ergeben sich aus der Kombination der Funktionen die resultierenden 16 Persönlichkeitstypen:

Einstellung	**Wahrnehmung**	**Beurteilung**	**Orientierung**	**Typ**
introvertiert	sinnlich	analytisch	entschlossen	ISTJ
introvertiert	sinnlich	analytisch	wahrnehmend	ISTP
introvertiert	sinnlich	gefühlsmäßig	entschlossen	ISFJ
introvertiert	sinnlich	gefühlsmäßig	wahrnehmend	ISFP
introvertiert	intuitiv	analytisch	entschlossen	INTJ
introvertiert	intuitiv	analytisch	wahrnehmend	INTP
introvertiert	intuitiv	gefühlsmäßig	entschlossen	INFJ
introvertiert	intuitiv	gefühlsmäßig	wahrnehmend	INFP
extravertiert	sinnlich	analytisch	entschlossen	ESTJ
extravertiert	sinnlich	analytisch	wahrnehmend	ESTP
extravertiert	sinnlich	gefühlsmäßig	entschlossen	ESFJ
extravertiert	sinnlich	gefühlsmäßig	wahrnehmend	ESFP
extravertiert	intuitiv	analytisch	entschlossen	ENTJ
extravertiert	intuitiv	analytisch	wahrnehmend	ENTP
extravertiert	intuitiv	gefühlsmäßig	entschlossen	ENFJ
extravertiert	intuitiv	gefühlsmäßig	wahrnehmend	ENFP

Tabelle 2: MBTI Persönlichkeitsprofile

Dieses System wird häufig im Bereich des Personalwesens verwendet, da eine Korrelation zwischen bestimmten Persönlichkeitstypen und Eignungen für einen Arbeitsplatz nachweisbar ist. Empirische Untersuchungen nach **[Ham93]** ergaben folgende Eignungstabelle.

ISTJ Management Administration Gesetzeshüter Bilanzierung	**INTJ** wissenschaftliche und technische Felder Computer Gesetz	**ESTJ** Management Administration Gesetzeshüter	**ENTJ** Management Führung
ISTP Handwerker Technische Felder Landwirtschaft Gesetzeshüter Militär	**INTP** wissenschaftliche und technische Felder	**ESTP** Marketing Handwerker Unternehmer Gesetzeshüter Angewandte Technologie	**ENTP** Wissenschaft Management Technologie Kunst
ISFJ Ausbildung Gesundheitswesen Religion	**INFJ** Religion Beratung Lehren Kunst	**ESFJ** Ausbildung Gesundheitswesen Religion	**ENFJ** Religion Kunst Lehren
ISFP Gesundheitswesen Unternehmer Gesetzeshüter	**INFP** Beratung Lehre Religion Kunst	**ESFP** Gesundheitswesen Lehre Nachhilfe Kinderbetreuung Handwerker	**ENFP** Beratung Lehre Religion Kunst

Tabelle 3: Anstellungsverteilung nach Typ

Anhand dieser groben Einteilung lassen sich Handlungen zu bestimmten Personentypen zuweisen und umgekehrt.

4.1.2 Kognitives Entscheidungsmodell – Belief Desire Intention

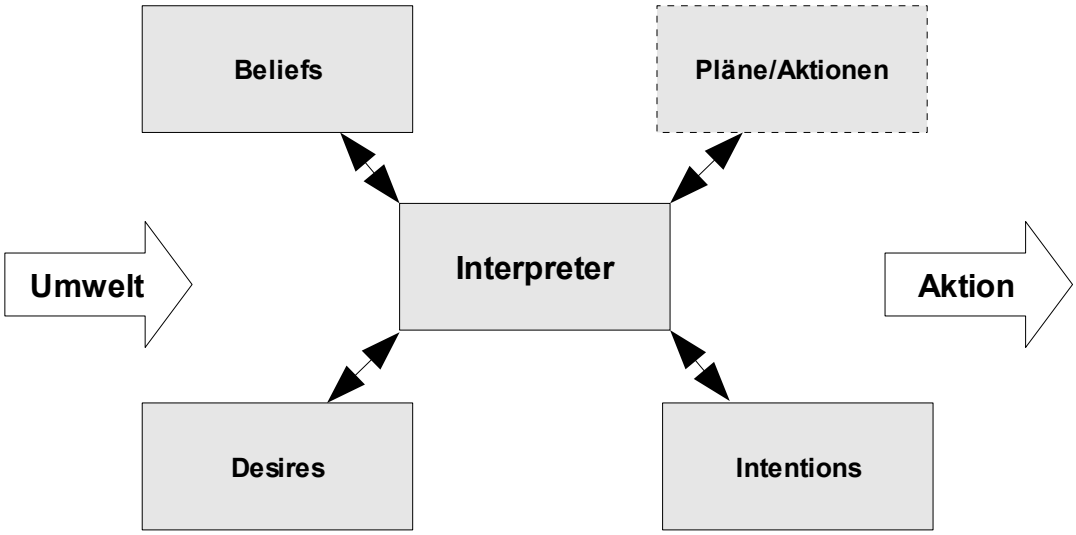


Abbildung 7: Belief-Desire-Intention Modell

Ziel eines kognitiven Entscheidungsmodells ist die Implementierung eines deliberativen Verhaltens. Im Gegensatz zu rein reaktiven Modellen, bei denen ein Zustand eine Handlung auslöst, versuchen deliberative Modelle Handlungen zu wählen, die einen gewünschten Zustand herbeiführen – einem Istwert einen Sollwert angleichen. Dieses Verhalten ist auch unter dem Begriff „practical reasoning“ bekannt und umfasst das Finden von Zielen und das Ausführen von Handlungen, die zu diesem Ziel führen. Das von Michael E. Bratman, Professor für Philosophie an der Universität von Stanford, postulierte BDI-Modell soll genau diesen Zweck erfüllen.

Das BDI-Modell besteht im Wesentlichen aus drei Komponenten – Wissen über die Welt (*beliefs*), Wünschen, Bedürfnissen und Zielen (*desires*) und Absichten (i*ntentions*). Damit der Agent seinen Wünschen näherkommen kann, wird ihm durch Pläne und Aktionen eine Möglichkeit gegeben, dies zu erreichen. Im Folgenden werden die drei Komponenten näher erklärt:

Motivation (Desire)

Die Motivation eines Agenten ist maßgeblich für die Formung eines Ziels. Wie der Mensch kann jeder Agent Sollwerte für Eigenschaften besitzen, welche bei Abweichung ein Bedürfnis auslösen. Beim Staubsauger-Roboter aus **[Rus95]** ist dieses Bedürfnis ein sauberes Zimmer. Sollte das Zimmer nicht sauber sein, weicht dieses Bedürfnis von seinem Sollwert ab und erzeugt eine Motivation für eine Aktion beim Agenten. Der Agent wird zu jedem Zeitpunkt mehrere Bedürfnisse haben, welche um eine Befriedigung konkurrieren. Es ist die Aufgabe des Interpreters, diesen Konflikt aufzulösen und eine Entscheidung zu treffen. Beim Menschen ist dieser Prozess unter dem Begriff „contention scheduling“ bekannt **[Fun06]**.

Wissen (Belief)

Um Entscheidungen treffen oder um Bedürfnisse aktualisieren zu können, benötigt der Agent ein Wissen von der Welt. Der Zustand der Welt ändert sich aufgrund des eigenen Verhaltens, des Verhaltens der anderen Agenten oder externer Gründe von Zeitpunkt zu Zeitpunkt. Dieses Wissen wird i.d.R. durch Fakten gespeichert, die eine atomare Information über eine Gegebenheit in der Welt, beispielsweise die Position eines anderen Agenten, beinhalten. Wichtig sind dabei nicht nur die aktuelle Situation, sondern durchaus alle bisher gesammelten Fakten der Welt.

Entscheidungsfindung (Intention)

Die durch die Motivation vorgegebenen möglichen Ziele müssen bewertet werden und es muss sich schließlich auf eines festgelegt werden. Dieses Festlegen auf ein Ziel und das Wählen einer Aktion formen die Absicht, die Intention des Agenten. Wie sehr der Agent eine einmal getroffene Entscheidung beibehält, wird durch das commitment bestimmt.

Obwohl das BDI-Modell einen psychologischen Ursprung hat, bietet sich aufgrund seiner Struktur eine einfache Implementierung als vom Computer auswertbares Modell an, was sich auch in der Vielzahl der am Markt verfügbaren Lösungen zeigt – IRMA (das ursprüngliche Modell von M. Bratman), dMars[9], Jason[10], JACK[11], um nur einige zu nennen. Insbesondere die Planungskomponente bzw. der Weg von einer

[9] dMars – http://garfix.blogspot.com/2005/05/bdi-agent-dmars-implementation.html

[10] Jason – http://jason.sourceforge.net/JasonWebSite/Jason Home.php

[11] JACK – http://www.agent-software.com.au/

Intention zu einer Aktion und die Repräsentation des Wissens ist unterschiedlich gelöst.

Der Vorteil des BDI-Modells liegt im intuitiven Verständnis. Die Prinzipien der Entscheidungsfindung und der zugrundeliegenden Komponenten sind im Wesentlichen dem Menschen ähnlich. Daher erscheint es als sinnvoll, gerade diese Architektur für die Modellierung eines NDM-Systems zu verwenden.

Ein großer Nachteil des Modells ist hingegen die fehlende Berücksichtigung eines Lernverfahrens **[Geo99]**. Das Verhalten des Agenten ist durch die Festlegung der Bedürfnisse und der möglichen Aktionen determiniert und i.d.R. nicht variabel. Wie in Kapitel 5 dargestellt, lässt sich das Problem der statischen Entscheidungsfindung durch einige Erweiterungen leicht umgehen.

Trotz der großen Vielfalt an Implementierungen wurde sich gegen eine vorhandene Lösung entschieden. Durch die benötigte enge Verbindung zu Maya müsste ein direkter Austausch von Daten ermöglicht werden, was nicht von allen Implementierungen unterstützt wird – i.d.R. läuft die gesamte Simulation innerhalb der Software und lässt keine schrittbasierte Abfrage der Daten zu (wie es für Maya benötigt wird). Einige Implementierungen beschränken sich trotz des BDI-onzepts auf ein reaktives System, welches durch eine Scriptsprache (AgentSpeak **[Rao96]**) beschrieben wird. Trotz der dadurch entstehenden einfachen Beschreibung der Welt wäre die Transformation der 3D-Daten aus Maya in eine durch AgentSpeak akzeptierte Form eine unnötige Wandlung, welche darüber hinaus das Risiko von Informationsverlust birgt (falls nicht alle Daten wandelbar sind).

Das bestehende Risiko durch die Verwendung einer Software, deren internen Vorgänge nicht bekannt sind, und dem zeitlichen Aufwand der mit einer tiefergehenden Analyse verbunden wäre, wurde sich entschieden, eine eigene BDI-Lösung zu implementieren, über die volle Kontrolle sowohl in Funktion als auch Geschwindigkeit vorhanden ist.

4.2 Physiologie

4.2.1 Sensorik

Der Agent benötigt Sensoren zur Wahrnehmung seiner Umwelt. Obwohl der Mensch fünf Sinne zur Orientierung besitzt, basieren seine Entscheidungen zu 90% auf den Daten des visuellen Systems. Da in Maya selbst nur visuelle Daten modelliert werden, bietet sich die Implementierung nur dieses Sinnes an. Dabei kann der Sensor auch über die Qualität der gesammelten Daten Einfluss haben, sodass der Agent davon ausgehen muss, nur einen Teil der realen Umwelt wahrzunehmen (siehe Kapitel 4.2.1 Belief-Desire-Intention-Modell).

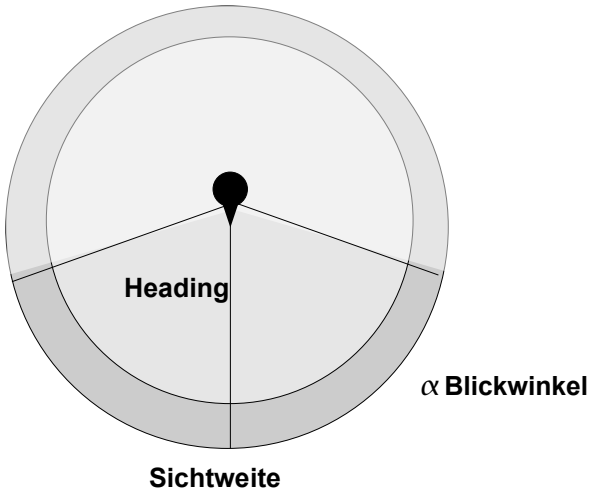


Abbildung 8: Sichtfeld

Der visuelle Sinn wird durch drei Parameter definiert. Das *heading* bestimmt die Blickrichtung und die Fokussierung des Sensors, das *field of view* den Blickwinkel. In der Regel befindet sich dieser bei Säugetieren bei ca. 170°. Aber auch atypische Konfigurationen sind durch Einstellen der Parameter möglich. Des Weiteren lassen sich mehrere Sichtweiten definieren, welche bestimmen können, ab welchem Radius

Objekte mit welcher Qualität wahrgenommen werden. Diese Sichtweite könnte auch eine Grenze für weitere Sensoren definieren.

4.2.2 Perzeption

Die wahrgenommenen Rohdaten müssen vom Agenten in verwertbare Informationen interpretiert werden. Dazu müssen Objekte und Eigenschaften erkannt und ihnen semantische Informationen zugewiesen werden. Diese Informationseinheit wird Perzept genannt. Dabei ist der Prozess der Perzeption nicht objektiv und absolut zu bestimmen. In Abhängigkeit der Qualität der Wahrnehmung und der Hintergrundinformationen eines Agenten ist aufgrund der gleichen Rohdaten eine unterschiedliche Interpretation möglich.

Der Agent benötigt eine Funktion zur Transformation der visuellen Daten in einer Form, welche der KI als Fakten zur Grundlage von Entscheidungsprozessen dienen kann.

4.3 Technik

4.3.1 Maya

Maya ist eine 3D-Modellierungs- und -Animationssoftware des Herstellers Autodesk. Sie ermöglicht es dem Benutzer, durch Erzeugen von Objekten beliebige Szenerien zu gestalten. Alle Objekte einer Szene bestehen aus Punkten im dreidimensionalen Raum, welche über Kanten miteinander verbunden sind und über die ein sogenanntes Mesh oder Polygonnetz „gespannt“ wird, um letztlich einen Körper darzustellen. Ein Polygon (Vieleck) beschreibt eine Fläche, welche durch die Grenzen (Kanten) des Vielecks definiert wird. Der durch die Kanten umschlossene Bereich bildet eine geschlossene Oberfläche, welche mit Eigenschaften (Farbe, Material, Lichtbrechung etc.) versehen werden kann. Durch Zusammensetzen dieser Flächen können Körper gebildet werden. In der Regel sind die heutigen Grafikkarten zur Darstellung und Verarbeitung von Dreiecken optimiert. Daher wird bei der Tesselierung (der Unterteilung einer Oberfläche in sogenannte primitive Flächen) darauf geachtet, die durch die Punkte beschriebene Hülle vollständig in Dreiecke aufzulösen. Darüber hinaus lässt sich aus Kombination von mehreren Dreiecken jedes andere Polygon bilden (siehe Abbildung 9 – Punkte sind violett, Kanten türkis und Polygonflächen grau gekennzeichnet).

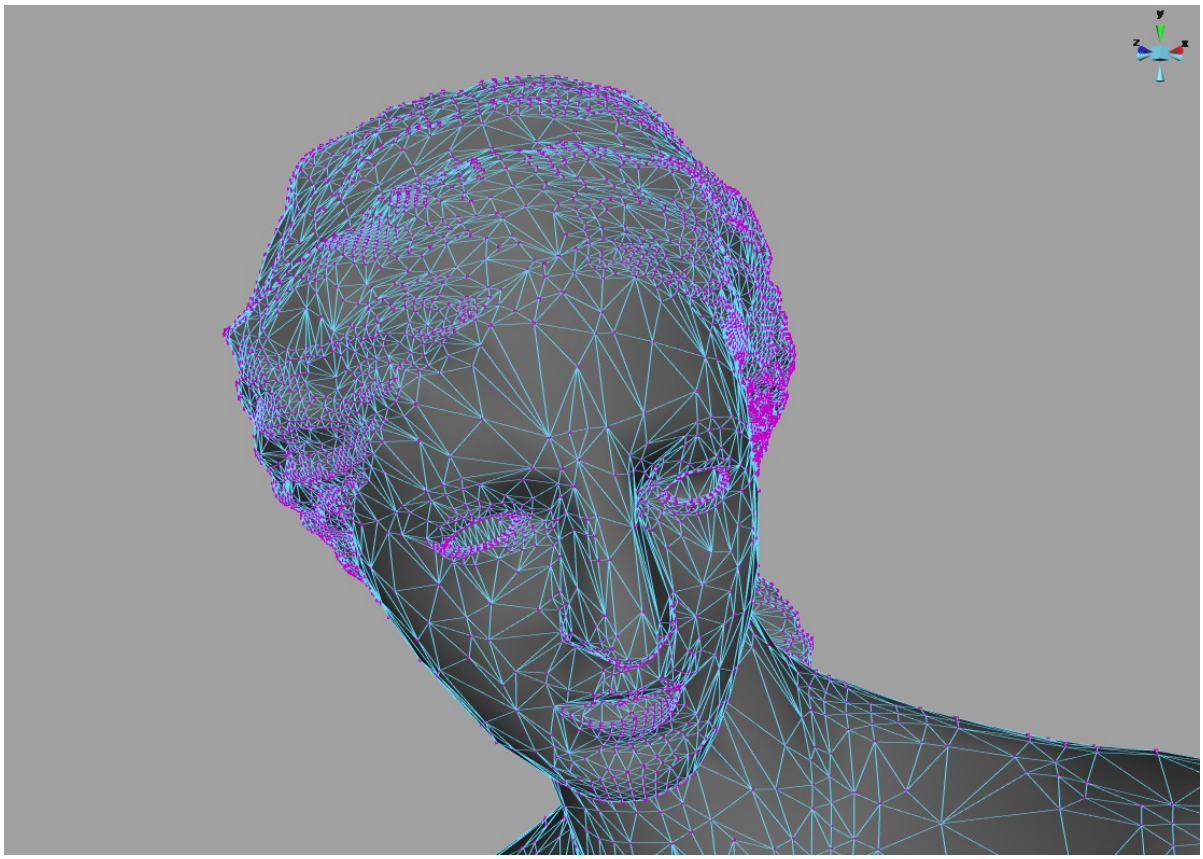


Abbildung 9: Maya Modell

Diese Modelle lassen sich über Schlüsselbilder animieren. Dazu werden die Zeit in diskrete Einheiten unterteilt und Attribute des Objekts (z.B. die Position) für einzelne Zeitpunkte (Schlüsselbilder) festgelegt. Durch Interpolation der Zeitpunkte zwischen den Schlüsselbildern lässt sich für jeden diskreten Zeitpunkt eine Position aller Punkte erwirken und damit eine Bewegung darstellen. In Abbildung 10 wird die Animation vom Attribut *TranslateX* dargestellt. Dieses Attribut beschreibt die Bewegung des Objektes entlang eines Vektors (einer Achse) des Raums, die Werte die Punkte entlang dieser Achse. Die Animation dauert 15 Zeiteinheiten (Frames) und es wurden zwei Schlüsselbilder gesetzt (bei Frame 0 und Frame 15). Die rote Linie zeigt die stattfindende lineare Interpolation der Werte. Auch wenn es unterschiedliche Ansätze gibt, so folgen doch alle Animationen diesem Prinzip.

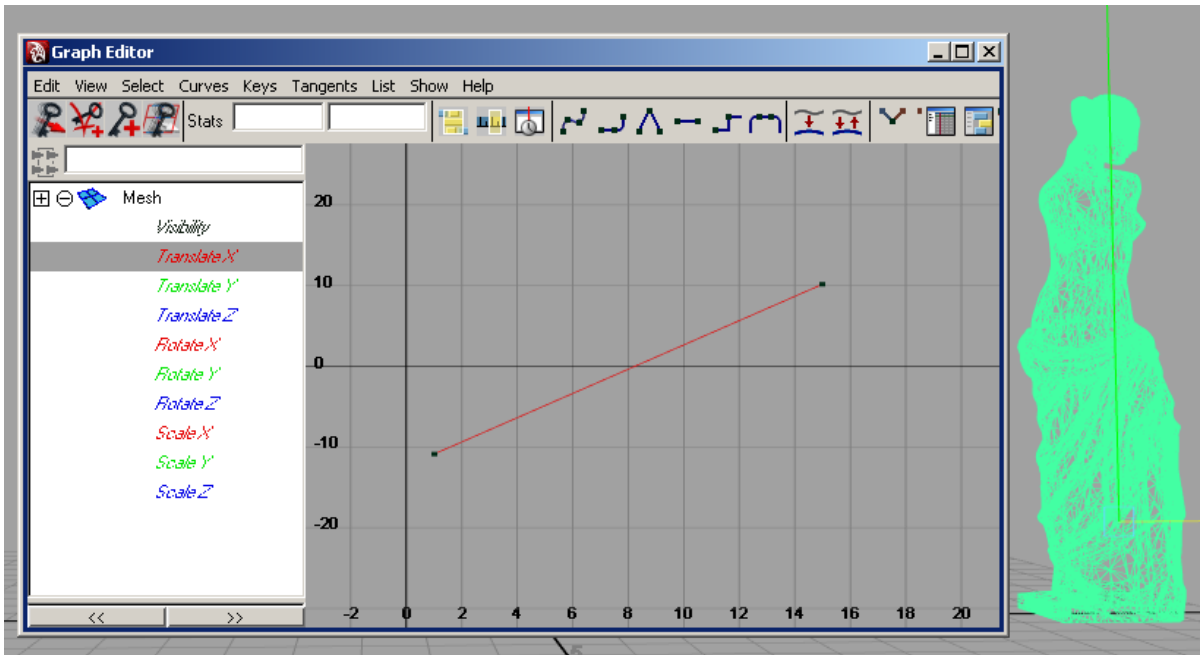


Abbildung 10: Maya Animation

Abschließend wird die Szene gerendert. Der Renderer projiziert dabei die dreidimensionale Darstellung auf eine Ebene, sodass am Ende ein Bild erzeugt wird. Je nach Konfiguration werden unterschiedliche Konzepte zur Bildgenerierung eingesetzt (Ray Tracing, Radiosity, Scanline[12]), welche sich hauptsächlich in Geschwindigkeit und dem verwendeten Beleuchtungsmodell unterscheiden.

[12] Dies ist nur eine Auswahl der gängigen Renderverfahren, für weitere Information wird auf einschlägige Literatur verwiesen **[Bir01]**

Abbildung 11: Maya Rendering

Nodesystem

Die Szene bildet die Basis für jede Gestaltung in Maya. Es ist ein leerer 3D-Raum, in dem Objekte erstellt, manipuliert, positioniert und animiert werden können. Sie wird durch einen *Dependency Graph* (DG) strukturiert. Der DG bestimmt, welche Daten sich in der Szene befinden und wie sie miteinander verknüpft sind. Jegliche Daten werden in sogenannten *Dependency Graph Nodes* (DG Node) gespeichert, sie sind die Knoten des DG (siehe Abb. 9).

Eine DG Node hat einen Dateneingang, Datenausgang und eine Berechnungsvorschrift (nach **[Gou03]**) sowie eine bestimmte Anzahl von Attributen.

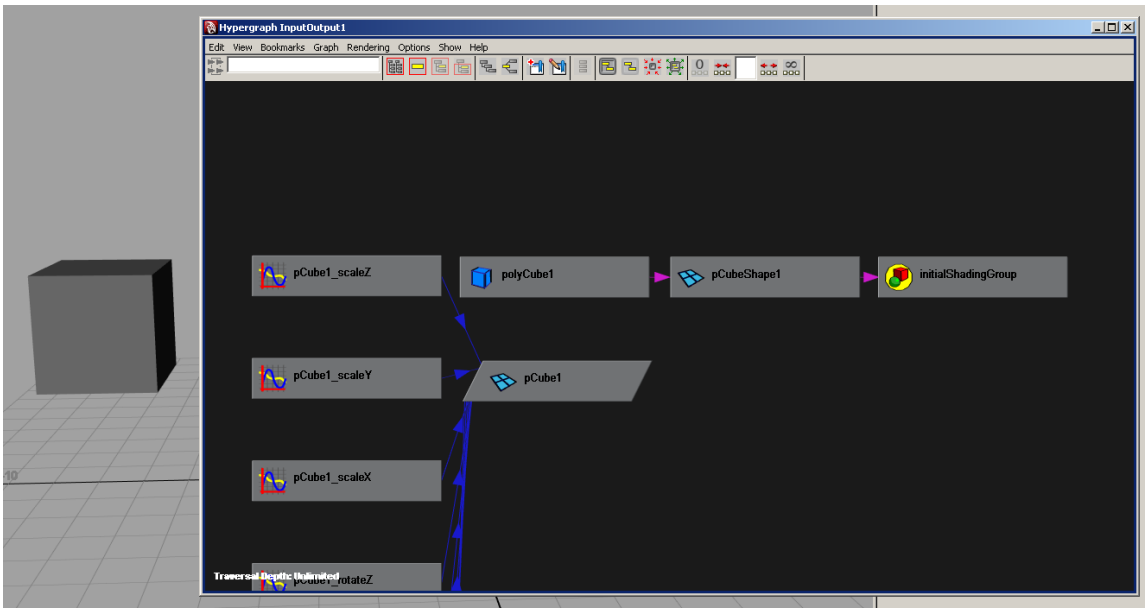


Abbildung 12: Beispiel - Dependency Graph

Ein Attribut bildet die Datenquelle der Node. Neben den häufig verwendeten simplen Datentypen *boolean, byte, char, short, long, float,* double können auch komplexe Datentypen wie parametrisierte Oberflächen gespeichert werden. Es kann innerhalb der Berechnungsvorschrift verwendet, aber auch extern über das GUI modifiziert werden, so dass der Benutzer Einfluss auf die Funktion nimmt.

Die Dateneingänge und -ausgänge bilden die Schnittstelle zu der Funktion eines Knotens. Attribute können mit einem *data plug* verbunden und als Eingang oder Ausgang deklariert werden. Der Eingang steht der Node als Parameter der Funktion zur Verfügung, während der Ausgang als Schnittstelle für andere Nodes verwendet wird. Die Neuberechnung der Ausgänge erfolgt nur, wenn ein über die Funktion *attributeAffects (attribute1, attribute2)* bestimmtes Attribut den Wert ändert. In Maya können eigene Nodes mit der API programmiert werden. Sie lassen sich wie die internen Nodes mit dem DG verbinden und in die Szene integrieren.

Der DG und die Funktionsweise der Nodes werden im Folgenden an einem Beispiel erläutert.

Beispiel:

Für die Auswertung einer Szene wird das Volumen jeder Kugel innerhalb der Szene benötigt. Eine Kugel besteht im Standardfall aus vier Nodes. *polySphere* gibt die Punkte vor, welche die Form und räumliche Ausdehnung des Objekts bilden. Die Gruppe von Punkten P (x, y, z) sind in einem Raum, der durch drei orthogonale Vektoren X, Y, Z aufgespannt ist, angeordnet (in diesem Fall in Form einer Kugel). Durch die Attribute der polyShere Node lassen sich unter anderem die Unterteilung (Subdivisions) der Kugel und damit der Genauigkeitsgrad und der Radius der Kugel anpassen. *pSphereShape* erzeugt aus der Punktwolke durch Tessellierung eine geschlossene Oberfläche, dass sogenannte Polygonnetz oder Mesh. *initialShadingGroup* beschreibt das Material- und Farbverhalten der Oberfläche[13]. *pSphere* ist die Transform Node, welche für die Positionierung und Lage im Raum verantwortlich ist (DG siehe Abb. 11). Die Formel für das Volumen einer Kugel ist:

$$\boldsymbol{V} = \frac{4}{3} * \pi * \boldsymbol{r}^3$$

Die Node polySphere hat das Attribut Radius, welches wir benötigen, um das Volumen zu berechnen. Somit sind alle benötigten Daten für die Volume Node vorhanden. Über die API wird eine leere Node erstellt. Sie erhält zwei Attribute: *radius* und *volume*, welche die Eingabe (Radius) und Ausgabe (Volumen) darstellen. Mittels *attributeAffects (radius, volume)* muss eine Verbindung hergestellt werden, sodass, wann immer sich der Radius der Kugel in der polySphere Node ändert, das Volumen neu berechnet wird. Abschließend wird der Ausgang *radius* der polySphere Node mit dem Eingang *radius* der Volume Node verknüpft.

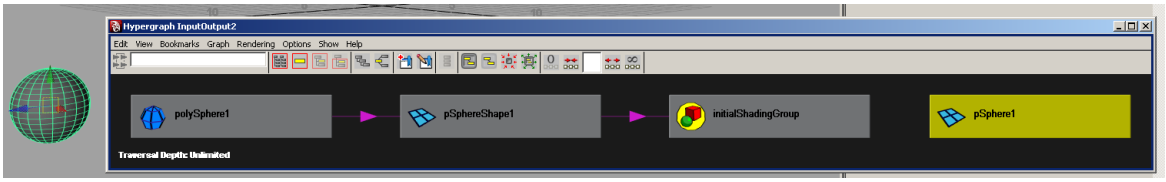


Abbildung 13: Beispiel - DG für eine Kugel

[13] Für eine umfassendere Erläuterung sei auf einschlägige Literatur verwiesen, wie [CG95], [OGL07]

MEL

Die Software unterstützt die proprietäre Scriptsprache MEL. Über das Script Interface erhält man Zugriff auf die Attribute der Objekte und kann diese manipulieren. Syntax und Semantik sind anderen Scriptsprachen, wie PHP[14] oder Perl[15], sehr ähnlich. Seit Version 8 wird ebenfalls Python[16] als alternative Scriptsprache unterstützt. Neben den klassischen Modulen für mathematische Funktionen gibt es eine große Anzahl Maya-spezifischer Befehle **[Aut08]**. Da jeder Vorgang innerhalb der Anwendung durch einen MEL-Befehl ausgelöst wird, ist es möglich, Maya nur über das Scriptinterface zu steuern. Es können Objekte erzeugt, abgefragt und verändert werden. Nachteile der Scriptsprache sind ihre unzulängliche Geschwindigkeit und die fehlende Möglichkeit, zur Laufzeit ausgewertet zu werden. Jeder Befehl führt einmalig eine Aktion durch und ist danach beendet. Zwar können Variablen über mehrere Ausführungen hinweg global gesichert werden, jedoch sind komplexere Strukturen nicht effizient verwaltbar.

Maya C++ API

Für Anwendungen, welche benutzerspezifische Nodes oder eine schnellere Ausführungszeit benötigen stellt Maya eine C++ API zur Verfügung.

Die API lässt dem Entwickler die Möglichkeit diverse Erweiterungen für Maya zu programmieren. Die für diese Arbeit relevanten Objekte sind Kommandos und DG Nodes.

Kommandos sind, ähnlich wir Scripte, eine Möglichkeit, Befehle zu definieren, welche innerhalb von Maya wie ein herkömmliches MEL-Script aufgerufen werden können. Neben der schnelleren Ausführungszeit bieten sie die Möglichkeit, weitere Bibliotheken einzubinden und so die Komplexität der Berechnung zu erhöhen. Jedoch ist ein Kommando ebenso wie ein MEL-Script auf die bestehenden Objekte innerhalb von Maya beschränkt. Nach Ablauf des Kommandos sind alle erstellten Objekte und Strukturen verloren. Um diese Daten zu sichern, müssten sie in bestehenden Objekten gespeichert werden oder als externe Datei vorliegen und zu jedem Aufruf geladen werden.

---

[14] http://www.php.net

[15] http://www.perl.org

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

Benutzerspezifische DG Nodes bieten die Möglichkeit, eigene Funktionen zu definieren, welche persistent in der Szene gespeichert sind. Maya bietet eine Vielzahl von spezifischen Nodes und Funktionen zur Manipulation von 3D-Daten, welche sich erweitern lassen, aber auch die Möglichkeit bieten, vollständig eigene Nodes zu erschaffen. Ein großer Vorteil bei der Verwendung von DG Nodes ist das Vorhalten von Daten, sodass diese nicht zur Laufzeit neu geladen oder berechnet werden müssen. Jedoch ist das vorrangige Problem bei der Verwendung von Nodes für die Simulation das Auslösen der Neuberechnung der Ausgangsdaten.

Die Maya API entspricht nicht dem klassischen objektorientierten Programmierparadigma. Jedes Objekt in Maya wird durch ein Mobject Objekt repräsentiert. Möchte man Zugriff auf eine bestimmte Node innerhalb einer Maya Szene, so deklariert man ein Objekt und weist ihm mit einer *getDependNode* Funktion eine Node zu. Das *Mobject* enthält nur die Daten der Node und bietet keine Funktion zur Manipulation. Für jede Gruppe von Manipulationen gibt es ein gesondertes sogenanntes *Function Set,* welches mit dem gewünschten Objekt verbunden wird.

Nachfolgend wird die Funktion der Maya API an einem Bespiel verdeutlicht.

Beispiel:

```
// Definition der Node
     MObject node;
// füllen des Mobjects node mit den Daten von Node i aus der
// Liste pnList (enthält alle Nodes der Szene)
     pnList.getDependNode(i, node);
// verbinden des Function Sets MfnDependencyNode für DG Nodes
// mit dem Mobject node
     MFnDependencyNode fNode(node);
```

Das Objekt node enthält nun alle Daten der Node *i* aus der Liste von Nodes *pnList* und hat einen direkten Verweis auf diese Node innerhalb der Szene.
Mit dem Objekt *fNode* können alle Funktionen des *MFnDependencyNode* auf das Objekt *node* angewendet werden und damit Node *i* manipulieren.

5 Architektur

Die Architektur muss die in Kapitel 2 aufgestellten Anforderungen befriedigen. Dazu wurden im vorangegangenen Kapitel die Grundlagen einiger Modelle für Faktoren menschlichen Handelns vorgestellt. Diese Bausteine müssen nun verbunden und in eine zusammenhängende Struktur für die Implementierung überführt werden.

Zu den Anforderungen gehören das Umsetzen des MBTI-Modells für persönlichkeitsgesteuertes Verhalten, eines BDI-Motivationssystems für die Bildung interner und eines Regelsystems für die Bildung externer Intentionen. Anders als bei einem reinen Regelsystem, bei dem eine Situation eine definierte Reaktion zur Folge hat, sollen mehrere Motivationen um eine Umsetzung konkurrieren. Erfolgsaussichten, Wichtigkeit der Motivation und Kosten determinieren dabei den Nutzwert (Utility) einer Handlung und bestimmen ihre Wahl.

Aufgrund der Maya-bezogenen Eigenheiten der Integration von Anwendungen ist eine Externalisierung des Verhaltensmodells vorzuziehen. Durch die weiteren Vorteile, die durch Verwendung einer separaten Arbeitsstation für die Berechnung entstehen (siehe Kapitel 2), wurde sich für die Umsetzung eines Client-Server-Systems entschieden. Neben dem zusätzlichen Aufwand für die Verwaltung der Verbindung beider Module muss ein Datenaustauschformat entwickelt werden.

Eine weitere essentielle Bedingung ist die intuitive Konfigurierbarkeit des Systems. Der Benutzer muss aus seiner eigenen Erfahrung und subjektiven Vorstellung über das zu resultierende Verhalten des Agenten die richtigen Einstellungen vornehmen können.

Das Verhaltensmodell kann auf zwei Arten umgesetzt werden. Ein Ansatz sieht vor, für jeden Agenten ein „Gehirn“ zu instanzieren und durch dessen Manipulation Einfluss auf den Agenten zu nehmen. Da die Auswertung des Modells jedoch i.d.R. den gleichen Gesetzen folgt, wurde entschieden, nur ein Modell zu erzeugen und dies durch die Parameter des Agenten zu konfigurieren. Die Autonomie der Agenten bleibt in beiden Fällen erhalten.

Zum besseren Verständnis wird analog zu den Komponenten der Architektur der Aufbau eines Agenten beispielhaft erläutert.

5.1 Client Server Modell

Die Trennung der Berechnung in zwei Module erweist sich nicht nur in technologischer Hinsicht als vorteilhaft. Indem man die Wahrnehmung und die (virtuelle) Welt des Agenten von seinem Gehirn trennt, erhält man nicht nur ein genaueres Abbild der menschlichen Struktur, bei der die Sensoren prinzipiell auch von der Berechnungseinheit (dem Gehirn) getrennt sind, sondern auch die Möglichkeit einer Schnittstelle für andere Anwendungen. Maya ist nur für die Aufnahme der Welt und die Weitergabe zuständig, während das KI-System diese Daten auswertet und ein Verhalten generiert. Ebenso sind die Effektoren sauber von der KI getrennt. In der vorliegenden Implementierung sind dies Rotation und Translation eines Objektes in einer dreidimensionalen virtuellen Welt, könnten aber auch die Motoren eines Roboters sein.

Im Allgemeinen wird unter einem Client-Server-System eine 1:n-Beziehung von mehreren Clients zu einem Server verstanden. In diesem Kontext stellen die Agenten die Clients und der Server das Verhaltensmodell dar, das für jeden Agenten (Client) eine Antwort erzeugt.

5.1.1 Client

Wie einleitend erwähnt, stellt der Client in Form von Maya die Sensoren und Effektoren des Agenten. Jeder Agent muss in jedem Simulationsschritt Objekte wahrnehmen und ihnen für die Auswertungen Eigenschaften zuweisen können, um anschließend im KI-System ein Perzept daraus zu bilden. Die auf dem Server berechnete Reaktion muss vom Client empfangen und umgesetzt werden.

5.1.2 Server

Auf dem Server ist das KI-System implementiert, welches die Konfiguration des Verhaltensmodells beinhaltet. Es erhält vom Client die rohen Sensordaten und weist diesen eine Bedeutung zu. Ein Expertensystem erzeugt nach Auswertung eine Reaktion, welche dem Client als Antwort zugesandt wird.

5.2 Agent

Der Agent ist der Akteur in der Welt. Er nimmt die Umwelt wahr, hat innere Motivationen und reagiert und, im Unterschied zu anderen Architekturen, agiert in und mit seiner Welt. Die Parameter, die einen Agenten ausmachen, seine Persönlichkeit, sein Wissen, sein Verhalten (determiniert durch seine Motivationen) müssen in einer Struktur gespeichert werden. Sie müssen variabel sein und Manipulation durch die Umwelt und andere Agenten zulassen, um eine völlig integrierte Umgebung zu schaffen. Der Nutzer muss dabei Zugriff auf die determinierenden Parameter des Agenten haben, um ein gewünschtes Ergebnis zu erzielen. Die entscheidenden Parameter, welche Einfluss auf das Verhalten des Agenten haben, sind seine Sensorik, seine Persönlichkeit und das Expertensystem (siehe Kapitel 5.4).

Beispiel:

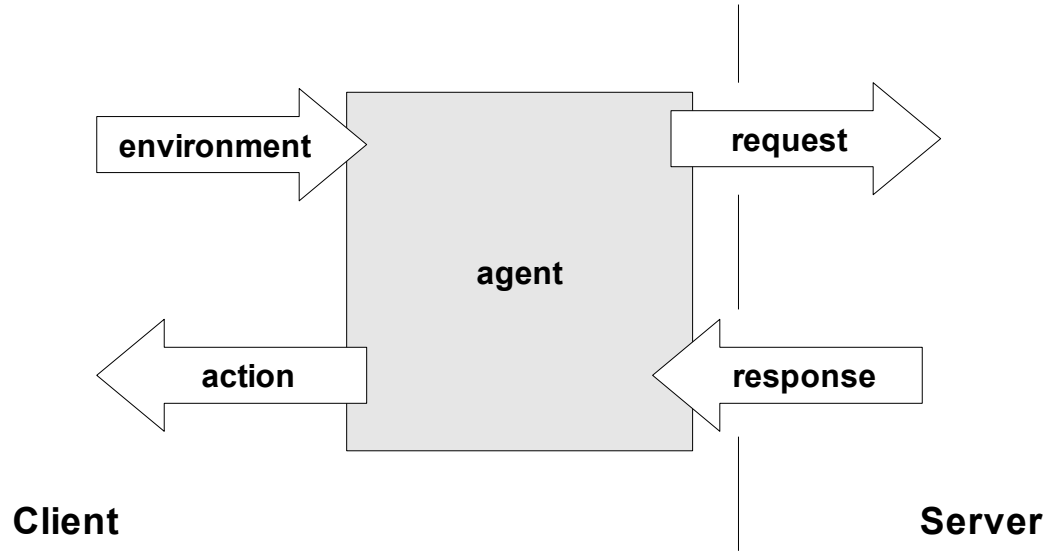


Abbildung 14: Agent - Sensorik

Der Agent besitzt bisher nur die Kommunikationsschnittstellen. Er kann Anfragen an den Server schicken und die Antworten empfangen. Darüber hinaus besitzt er Zugriff auf seine virtuelle Welt.

5.2.1 Sensorik

Der Agent muss Objekte wahrnehmen können. Da Wahrnehmung in einer bestimmten Art begrenzt ist, werden für jeden Agenten eine Sichtweite und ein Blickwinkel bestimmt, in dem er Objekte erkennen kann. Diesen Objekten muss anschließend eine semantische Information zugewiesen werden. Da bisher kein effizienter Ansatz für eine generelle Objekterkennung bekannt ist, musste hierfür eine andere Lösung gefunden werden. Die Entscheidungen eines Agenten beruhen auf den wahrgenommenen Eigenschaften eines Objektes, beispielsweise blau, gefährlich, weiblich, der Zuordnung einer Funktion, beispielsweise Waffe, Fahrzeug, aber auch einfacher Objekterkennung, beispielsweise Stuhl, Mensch, Gegner. Um diese komplizierte Zuordnung semantischer Informationen zu vereinfachen, wird jede dieser Eigenschaften mit einem Integer-Wert identifiziert. Jeder Agent erhält somit durch eine Gruppe von Zahlen Eigenschaften zugewiesen und ist für einen anderen Agenten auswertbar (Abbildung 15 – Komponente *K1*).

Anhand dieser Sensordaten wird eine Situation beschrieben und erkannt. Alle Bedingungen, unter denen Handlungen erfolgen sollen, sind von diesen Eigenschaften, oder *features*, abhängig. Da der Agent modelliert werden soll, spielen diese Informationen die wesentliche Rolle in der Entscheidungsfindung (siehe Kapitel 5.3) und in der Konfiguration des Expertensystems (siehe Kapitel 5.4), die das Verhalten bestimmen.

Da das Verhalten nicht nur von der aktuell wahrgenommenen Situation bestimmt werden soll und im schlimmsten Fall ein rein reaktives Verhalten erfolgt, müssen die Perzepte gespeichert und mit der Dimension Zeit versehen werden. Der Agent muss über eine Gedächtnisfunktion auf Informationen aus vorangegangenen Simulationsschritten zugreifen können und diese ggf. für die aktuelle Situation berücksichtigen. Der Zeitfaktor kann (muss aber nicht) dabei eine Rolle spielen und die Wichtigkeit oder Korrektheit einer Information beeinflussen (Abbildung 15 – Komponente *K2*).

Beispiel:

Der Agent aus Abbildung 15 erhält eine Sensorik, um die Umwelt wahrzunehmen. Wie bereits erläutert, erhält jedes an der Simulation teilnehmende Objekt Eigenschaften zugewiesen, welche durch einen anderen Agenten erkannt werden. Sollen zwei Agenten ihre Gruppenzugehörigkeit erkennen, um beispielsweise Freund und Feind in einer Schlacht zu erkennen, kann dies durch die Zuweisung einer solchen Eigenschaft erreicht werden. Agent A aus Gruppe 1 erhält in seine Liste von Eigenschaften somit eine 1 eingetragen, um die Zugehörigkeit zu Gruppe 1 und Agent B eine 2, um die Zugehörigkeit zu Gruppe 2 zu beschreiben.

Durch Vergabe weiterer Eigenschaften (durch Einfügen von weiteren Identifikatoren zur Eigenschaftenliste) kann der Benutzer eine vollständige sensorische Beschreibung eines Agenten erzeugen.

5.2.2 Persönlichkeit

Ein weiterer Faktor, der für jeden Agenten festgelegt werden muss, ist die Persönlichkeit, welche entscheidenden Einfluss auf die Aktionswahl hat. Das in Kapitel 4 vorgestellte MBTI-Modell ist sehr einfach durch einen vier Bit-Breiten-Vektor realisierbar, dessen Elemente die SNTF-Eigenschaften des Modells bedienen. Um eine Verbindung zwischen einer Aktion und der Persönlichkeit eines Agenten zu schaffen, muss nicht nur jedem Agenten, sondern auch jeder Aktion ein solches Persönlichkeitsprofil mitgegeben werden. Durch eine Vergleichsfunktion lässt sich ein Ranking erzeugen, wie gut eine Aktion zu einem Persönlichkeitsprofil passt. Die Persönlichkeit wirkt somit wie ein zusätzlicher Filter bei der Antwortselektion. Auch hier liegt die Information in der Konfiguration des vom Benutzer festgelegten Expertensystems, welche Aktion zu welcher Persönlichkeit passt (Abbildung 15 – Komponente *K3*).

Beispiel:

Die beiden Gruppen der Agenten A und B aus dem vorangegangenen Beispiel bestehen aus mehr als nur einem Agenten. Gruppe 1 hat beispielsweise zehn Mitglieder. Damit nicht alle Agenten aus dieser Gruppe identisch handeln, sondern eine natürliche Varianz erreicht wird, beschreiben wir diese Agenten durch unterschiedliche Profile. Dadurch haben wir Einfluss darauf, welcher Agent mit einer höheren Wahrscheinlichkeit eine bestimmte Aktion bevorzugt. Wollen wir beispielsweise, dass die Agenten aus Gruppe 1 vermehrt vor Mitgliedern der Gruppe 2 fliehen, so müssen,

abhängig vom restlichen Expertensystem, nur die Profile der Agenten derart abgeändert werden.

5.2.3 Motivation

Ein Handlungswunsch kann eine von zwei Ursachen haben. Entweder entsprechen die wahrgenommene externe Situation oder die internen Parameter des Agenten nicht dessen Vorstellungen. Diese Abweichung von Sollwert und Istwert ist einer der Faktoren für eine Intention. Der Agent hat dabei so viele Handlungswünsche wie Bedürfnisse, jedoch bestimmt die Dringlichkeit der Deckung eines Bedürfnisses und dessen Wichtigkeit die Wahrscheinlichkeit einer Wahl. Um eines dieser Bedürfnisse zu decken, muss der Agent eine Aktion wählen, welche für diesen Zweck am geeignetsten ist. Das BDI-Modell (siehe Kapitel 4.1.2) beschreibt dieses Modell ideal. In Abhängigkeit des Wissens über die aktuelle (und vergangene) Situation (*beliefs*) und der aktuellen Bedürfnisse (*desires*) muss eine Aktion gewählt werden, um das notwendigste aller Bedürfnisse zu befriedigen (*intention*). (Abbildung 15 – Komponente *K4*)

Beispiel:

Der Agent kann nun durch seine Sensoren die Umwelt wahrnehmen und durch seine Persönlichkeit von anderen Agenten unterschieden werden, um ihm andere Aktionen zuweisen und dadurch sein Verhalten beeinflussen zu können. Durch die Einführung der Motivation erhält der Agent eine Möglichkeit, seinen Bedarf und letztendlich eine Aktion zu bestimmen. Zunächst wird bestimmt, was Agent A und B für ein Verhalten an den Tag legen sollen – hier: miteinander kämpfen oder fliehen. Dazu müssen hierfür entsprechende Bedürfnisse und Aktionen formuliert werden.

Bedürfnis(Kampf) -> Aktion(Kämpfen)

Bedürfnis(Flucht) -> Aktion(Flucht)

Je nach Ausprägung der Bedürfnisse wird der Agent nun eine der beiden Aktionen wählen.

In Abbildung 15 werden alle Komponenten des Agenten zusammengefasst. Insbesondere sollen hier die Parallelen zum BDI-Modell (siehe Kapitel 4.2.1) aufgezeigt werden.

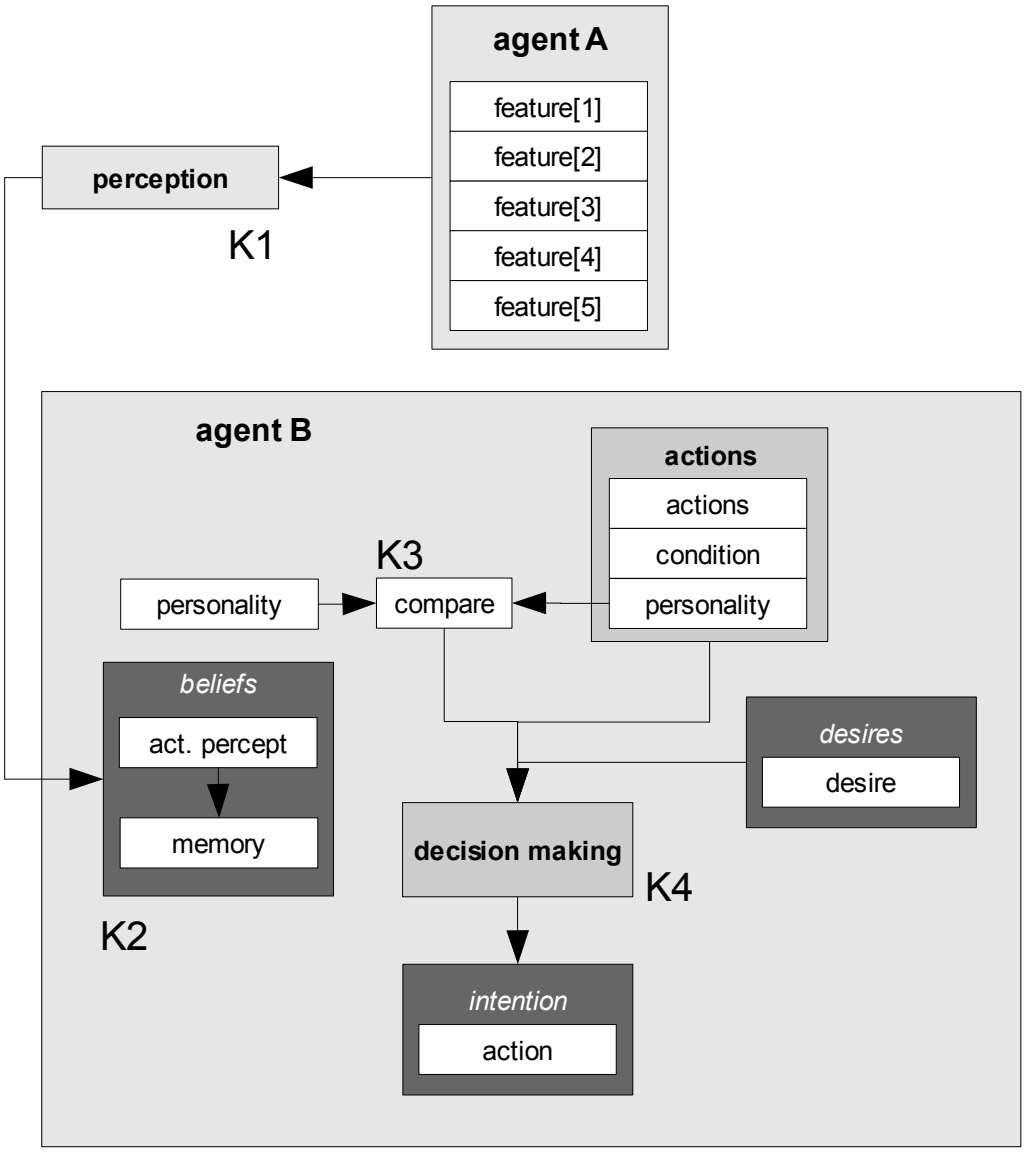


Abbildung 15: Architektur – Agent

5.2.4 Aktion

Um dem Agenten eine Handlungsoption zu geben, müssen Aktionen definiert werden. Eine Aktion hat dabei Vor- und Nachbedingungen, welche bestimmen, unter welchen Prämissen die Aktion ausführbar ist und wie sich die aktuelle Situation dadurch ändert. Die Nachbedingungen sind dabei nicht explizit formuliert. Stattdessen zeigen sie sich in der Manipulation der Umwelt, sodass sie bei einer späteren Wahrnehmung des Agenten die implizit formulierten Nachbedingungen widerspiegeln werden.

Um für eine bestimmte Absicht eines Agenten eine passende Aktion auszuwählen, müssen diese miteinander verknüpft werden. Über diese Beziehung kann festgelegt werden, wie gut eine Aktion ein Bedürfnis decken kann, wie hoch die Kosten und der Nutzen sind und somit, wie hoch die Nützlichkeit einer Aktion ist. Da unter Umständen mehrere Aktionen für eine Absicht infrage kommen, muss ein Vergleich dieser validen Aktionen stattfinden und die beste gewählt werden.

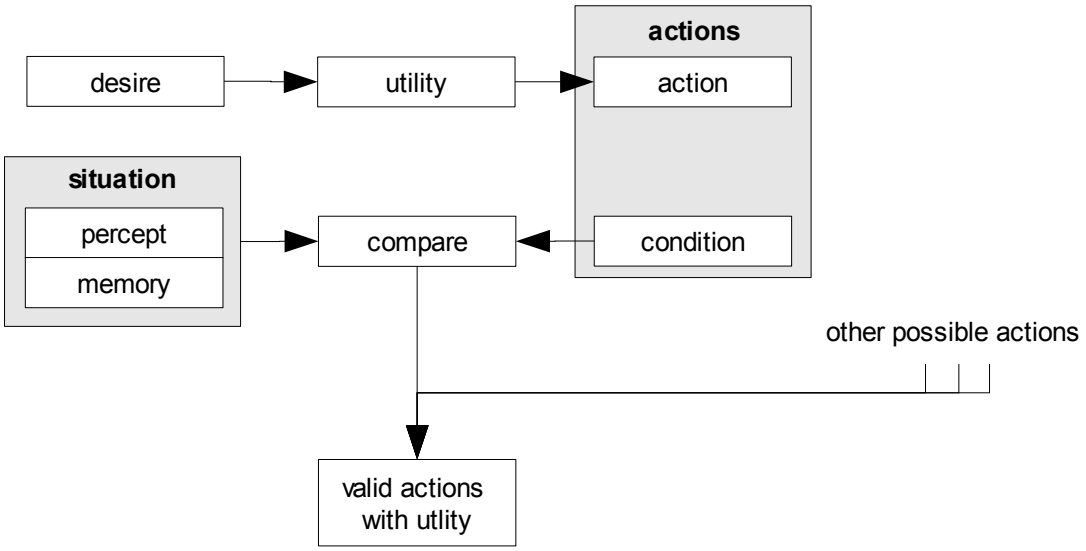


Abbildung 16: Architektur – Aktion

Beispiel:

Die Agenten A und B können über die Bedürfnisse ermitteln, welches Bedürfnis gestillt werden soll, und suchen eine passende Aktion. Dazu ermitteln sie alle Aktionen, die durch das Expertensystem mit dem Bedürfnis verknüpft sind und berechnen für jede dieser Aktionen eine Nützlichkeit (das Bedürfnis zu befriedigen).

In dem hier vorgestellten Beispiel werden zwei Aktionen für die Agenten A und B definiert.

Aktion(Fliehen)	Befriedigt das Bedürfnis(Flucht)
	Reduziert das Bedürfnis(Kampf)
Aktion(Kämpfen)	Erhöht das Bedürfnis(Flucht) des Gegners

Aufgrund dieses Systems werden Agenten A und B so lange gegeneinander kämpfen, bis bei einem vom beiden die Motivation, das Bedürfnis „Flucht“ zu befriedigen, größer wird als das Bedürfnis „Kampf“. Die Aktion „kämpfen“ hingegen hat dynamische Komponenten in Abhängigkeit der Persönlichkeit und der Eigenschaften des Agenten. So erreicht man ein Ungleichgewicht, was eine Pattsituation unterbindet.

Damit wird der Agent vollständig beschrieben. Er hat die Möglichkeit, Umweltdaten zu sammeln, kann auf Basis dieser mögliche Aktionen bestimmen, durch eine Motivation eine Intention bilden und daraufhin eine für ihn passende Aktion wählen.

5.3 Entscheidungsfindung

Das Herzstück des Systems ist die Entscheidungsfindung. Es bestimmt, wie die Parameter des Agenten und die Vorgaben des Expertensystems miteinander kombiniert werden, um eine Antwort zu erzeugen. Diese Antwort besteht im Wesentlichen aus einer Aktion, welche eine Geschwindigkeit und eine Richtung für den Agenten vorgibt, in die er sich im folgenden Simulationsschritt bewegen soll.

Wie bereits in Kapitel 5.2 dargelegt, bestimmen die Bedürfnisse eines Agenten seine Aktionswahl. Für jedes dieser Bedürfnisse wird eine Wichtigkeit ermittelt und diese mit einer möglichen Aktion verbunden. Die Kombination der Wichtigkeit des Bedürf-

nisses mit der Ausführbarkeit und Nützlichkeit (für das gewählte Bedürfnis) ergibt einen endgültigen Vergleichswert für alle Aktionen und Bedürfnisse des Agenten. Dieser bestimmt am Ende die Wahrscheinlichkeit einer Wahl.

So wird sichergestellt, dass der Agent in jeder Situation immer das aktuell für ihn notwendigste Bedürfnis mit der dafür besten Aktion deckt.

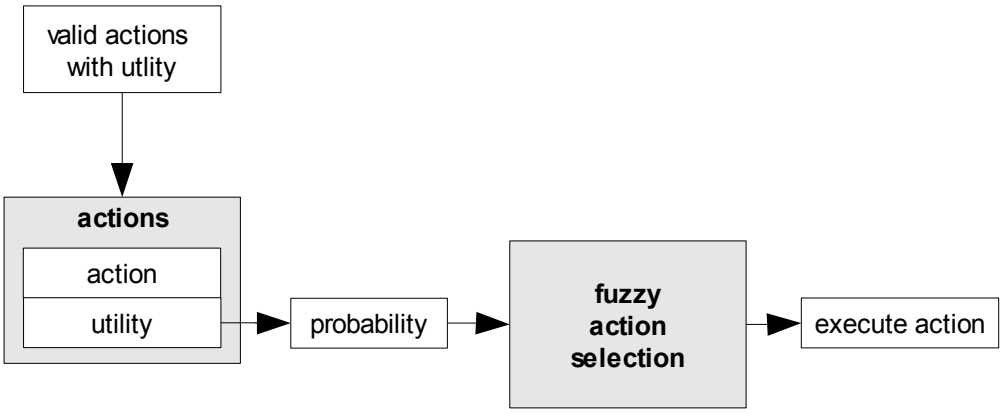


Abbildung 17: Architektur – Entscheidung

5.4 Expertensystem

Der Agent benötigt eine Grundlage, auf deren Basis er eine Entscheidung für eine Aktion treffen kann. Das beschriebene Verhaltensmodell benötigt vier Komponenten, um berechnet werden zu können. Ein Satz von Regeln und deren Bedingungen bestimmen das reaktive Verhalten des Agenten in der Form, „wenn Situation A gegeben ist, führe Aktion B aus“. Dieses reaktive Verhalten ist nicht nur für eine Aktion des Agenten verwendbar, sondern auch, um interne Parameter eines Agenten zu setzen, sodass ebenfalls Regeln der Form „wenn Situation A gegeben, dann setze Parameter B von Agent C“ möglich sind (beispielsweise, um bei der Wahrnehmung einer Gefahr den Parameter Angst zu erhöhen). Da nicht nur externe Faktoren einen Einfluss auf interne Parameter haben können, müssen auch unbedingte Situationen

berücksichtigt werden und für diesen Fall muss ebenfalls eine Struktur geschaffen werden.

Um die so verwalteten Bedürfnisse des Agenten mit einer Aktion zu verknüpfen, muss eine weitere Komponente definiert werden. Diese muss nicht nur eine Verbindung schaffen, sondern diese mit einem Gewicht belegen, um die Nützlichkeit einer Handlung für ein Bedürfnis zu modellieren.

Letztlich bestimmen die Bedingungen der Regeln und Aktionen deren Auslösung und Ausführbarkeit. Durch sie wird ein Zusammenhang zwischen den Eigenschaften der Agenten (und dadurch der Situation) und Bedürfnissen und schließlich zu Aktionen geschaffen (siehe Abbildung 17 – Die Komponenten des Expertensystems sind grau unterlegt).

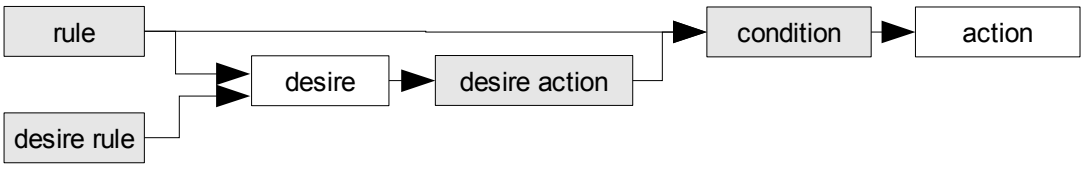


Abbildung 18: Architektur – Expertensystem

5.5 Planung

Für den beabsichtigten Anwendungszweck ist eine Planungskomponente nicht nur nicht erforderlich, sondern sogar störend, daher wurde bewusst auf einen expliziten Entwurf und eine Implementierung verzichtet. Für den Einsatz im Film wird eine ausreichende Kontrolle des Agenten und seines Verhaltens benötigt. Der Ausgang der Simulation ist durch die Gestaltung der Szene bereits vorgegeben. Würde der Agent selbständig eine Aktionssequenz ermitteln, um zu einem festgelegten Ziel zu gelangen, würde unter Umständen eine Handlungsfolge gewählt werden, welche nicht dem erwarteten und gewollten Verhalten des Benutzers entspricht. Beispielsweise würde der Drang, das Bedürfnis „Durst“ zu stillen, auch mit dem Wasser aus einem Springbrunnen möglich, wäre jedoch nicht gewollt. Ein weiteres Problem ist die Auswertung des semantischen Netzwerks, welches durch die Definition des Experten-

systems erzeugt wird. Die Auswahl einer Handlung H wäre nicht nur aufgrund des gewollten Ergebnisses zu bestimmen (Befriedigung von Bedürfnis A), sondern ebenfalls in der Berücksichtigung, dass kein Bedürfnis B durch Handlung H ausgelöst wird, was negativer wäre als Bedürfnis A. Da das System ohne Nachbedingungen auskommt, müsste eine Bewertungsfunktion eingeführt werden, welche die Auswirkungen von Handlung H auf alle Faktoren bestimmt und diese bewertet und anschließend bestimmt, ob H ausgeführt wird. Des Weiteren würde die Auswertung eines Planers unter Berücksichtigung aller weiteren Agenten innerhalb der Szene einen Großteil der zur Verfügung stehenden Ressourcen verbrauchen, ohne ähnlich hohen Nutzen zu generieren.

Um dennoch rudimentäre Handlungssequenzen zusammenstellen zu können, kann man mehrere Aktionen mit einem Bedürfnis verknüpfen über die Utility dieser Verknüpfung. Eine solche Handlungssequenz würde beispielsweise aus den Aktionen A1, A2 und A3 bestehen, wobei A3 die Aktion ist, welche das Bedürfnis befriedigt, und A1 und A2 Unterziele, um Aktion A3 auszuführen. A3 erhält die höchste Utility, A2 eine mittlere und der Anfang einer Sequenz A1 die niedrigste. Sollte A3 direkt ausführbar sein, so wird der Agent diese Aktion wählen, wenn nicht A2 (die nächstbeste), und falls diese ebenfalls nicht ausführbar ist, dann Aktion A1. Der Benutzer behält in jedem Fall die Kontrolle über den Agenten, ohne sich aber direkt um seine Handlungen kümmern zu müssen.

5.6 Lernen

Aus einem ähnlichen Grund wie beim Planen wird auch auf ein Lernverfahren verzichtet. Die gelernten Abläufe bzw. Handlungen der Agenten würden nicht zwangsläufig zum gewollten Verhalten tendieren. Durch eine statistische Zuweisung ist nicht nur das Verhalten der Agenten explizit bestimmbar, sondern das Ergebnis sofort berechenbar. Durch eine im anderen Fall vorangehende Lernphase würde dieser zeitliche Vorteil einer automatisierten Steuerung verringert werden. Jedoch ist die Verwendung eines Lernverfahrens für andere Anwendungszwecke durchaus sinnvoll, daher wurde beim Entwurf der Architektur auf passende Schnittstellen geachtet.

Erlernbare Parameter wären die Gewichte der Verknüpfung zwischen einem Bedürfnis und einer Handlung und die Parameter der Regeln der Bedürfnisse (siehe Abbildung 17 – desireRule und desireAction). Die Utility einer Aktion für ein Bedürfnis

wäre leicht über eine „reinforcement learning“-Strategie realisierbar, bei der vor und nach Durchführung einer Aktion der Wert des Bedürfnisses gemessen und verglichen wird. Je besser die Befriedigung des Bedürfnisses oder die der Gesamtsituation (Summe aller Bedürfnisse) – je nach Konfiguration – ausfällt, desto höher ist die Wahrscheinlichkeit einer erneuten Wahl.

Ein ähnliches Verfahren wird in den Arbeiten **[Spr05]** und **[Lad08]** verwendet, deren Nützlichkeit und Anwendbarkeit durch Implementierungen für die Computerspiele „Jagged Alliance 2“ und „Neverwinter Nights“ nachgewiesen wurden. Durch eine sogenannte Fitnessfunktion würde durch mehrmalige Simulation einer Szene die Utility einer Aktion oder einer Aktionsfolge für ein bestimmtes Bedürfnis ermittelt und somit ein passendes Gewicht bestimmt. Für den Einsatzzweck für Filmszenen ist dies nur bedingt nutzbar, da keine Interaktivität und somit die Anpassung der Gewichte aufgrund eines Nutzers gefordert ist und die Vorhersagbarkeit der Simulation verletzt wäre. Um dennoch einen gültigen Simulationsausgang zu erreichen, müsste man den Umweg gehen und die Aktionen der Agenten so modellieren, dass immer die gewollte Aktion auch die im System tatsächlich beste Aktion für eine bestimmte Situation ist. Wenn man aber dies bereits ermittelt hat, können die Gewichte auch statistisch direkt bestimmt werden.

Geht man jedoch den anderen Weg und implementiert das in dieser Arbeit entwickelte Entscheidungssystem in eine interaktive Anwendung, so lässt sich aus variablen Gewichten ein Nutzen ziehen.

6 Implementierung

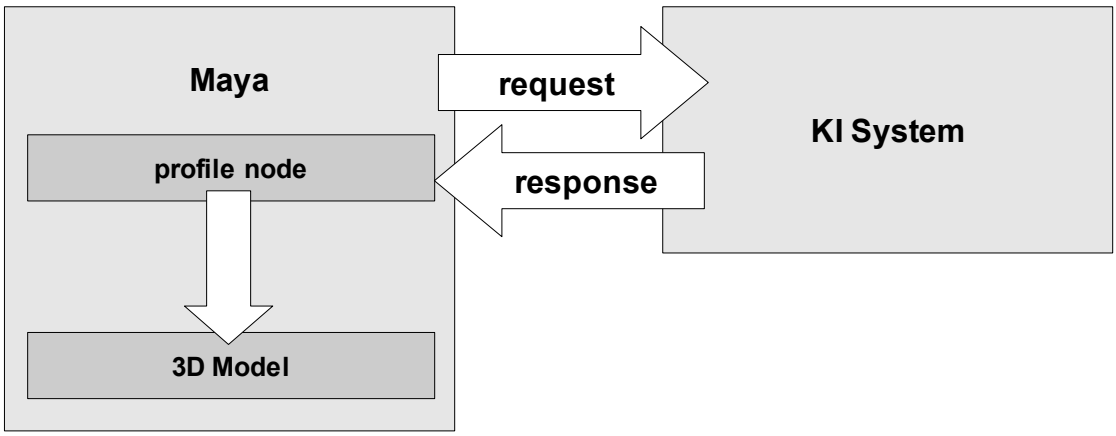


Abbildung 19: Client Server Architektur

Die Implementierung der in Kapitel 5 erläuterten Architektur wird durch einen Top-Down-Ansatz beschrieben:

Die Implementierung umfasst zwei Hauptkomponenten, das Maya-Modul und das Server-Modul. Maya bildet den Client und die Schnittstelle zum Benutzer, welcher über die Attributfelder der *profile node* den Agenten konfigurieren und die Szene gestalten kann. Über eine weitere Schnittstelle kann eine Anfrage (*request)* für ein Update an den Server gestellt werden. Der Server – das KI-System – bestimmt auf Basis des Verhaltensmodells und des Expertensystems im KI-System eine Antwort (*response*) für das Verhalten des Agenten im nächsten Simulationsschritt. Es ergeben sich somit drei Phasen:

1. Anfrage für jeden Agenten (Client)
2 Berechnung der neuen Aktion (Server)
3. Umsetzen der Aktion (Client)

Der simulierte Agent ist redundant in der Client und der Serveranwendung vorhanden und wird in jedem Schritt abgeglichen (von Client zu Server). Dadurch wird ein kohärentes und konsistentes Abbild der Agenten auf beiden Seiten erreicht. Dies ist zwingend notwendig, da der Server aus Performancegründen nur auf die Daten dieser Abbilder zugreift. Zusätzlich zu den Attributen des Agenten verwaltet der Server die Wahrnehmung und die Knowledge Database des Agenten.

Die Kommunikation zwischen Client und Server wird über Netzwerk-Sockets und das TCP/IP-Protokoll realisiert. Erst, wenn eine durch die sogenannte Handshake-Phase sichere Verbindung hergestellt wurde, wird mit dem Datenaustausch begonnen. Um eine einfache Synchronisation mit Maya zu gewährleisten, wurde von der Verwendung von Non Blocking States abgesehen. Für diesen Prototypen des KI-Systems wurde wegen der einfacheren Änderung des Systems der sequentiellen Verarbeitung der Anfragen gegenüber einer Multi-Threading-Anwendung der Vorzug gegeben.

6.1 Client Plug In

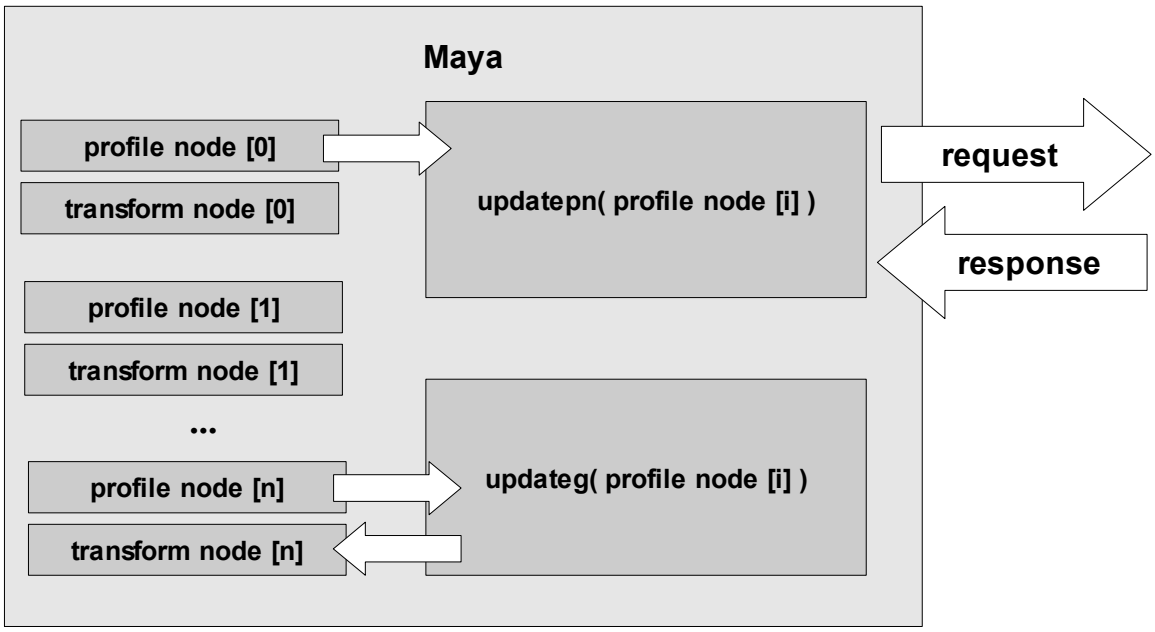


Abbildung 20: Client Plug In für Maya

Das Client Plug In besteht aus einer Maya DG Node (*profile node*) und zwei Kommandos (*updatepn, updateg*). Zu den Anforderungen des Maya Plus Ins gehören das Versenden der Daten des Agenten an den Server und der Empfang der Nachricht, sowie deren Umsetzung.

Jedes Objekt, welches an das KI-System angeschlossen werden soll und als Agent fungiert, wird mit einer *profile node* verknüpft. Jedes transformierbare Objekt in Maya ist darüber hinaus mit einer *transform node* verknüpft, welche u.a. die Parameter für Rotation und Translation des 3D Meshes speichert (siehe Kapitel 4.3.1). Dieses Paar von DG Nodes bildet den Agenten auf der Seite des Clients. Es enthält alle Daten über das Profil des Agenten sowie seine räumliche Position und Orientierung. Eine Verknüpfung zwischen den Knoten ermöglicht einen Datenaustausch (siehe Kapitel 6.2.1). Sollte das Objekt durch den Benutzer oder eine maya-interne Funktion transformiert werden, so werden zeitgleich die Daten in der *profile node* aktualisiert.

Jeglicher Austausch zwischen Client und Server geschieht über das *updatepn*- Kommando. Dabei werden, abhängig vom auszuführenden Befehl, die Daten aus der *profile node* ausgelesen und an den Server gesendet. Der Client befindet sich nach dem Senden der Daten im Blocking State der Client Server-Kommunikation. Die Anwendung hält so lange alle aktiven Prozesse an, bis eine Antwort vom Server erhalten wurde.

Das *updateg*-Kommando erzeugt aus der Antwort des Servers einen Maya-Befehl, welcher den Agenten steuert, d.h. eine neue Position, Orientierung und Aktion zuweist.

6.1.1 Profile Node

Die *profile node* ist eine durch die Maya API benutzerspezifische DG Node zum Speichern und Verwalten der Parameter des Agenten in der Szene. Sie beinhaltet drei Gruppen von Parametern (siehe Abbildung 14[17]). *agent* umfasst die Eigenschaften (*feature*), die Bedürfnisse (*desire, desireAuto*) und das Persönlichkeitsprofil (*profile*) des Agenten, die Gruppe *transform* die Daten zur Orientierung und Positionierung in der dreidimensionalen Szene. Zur Bestimmung der Position und Orientierung sind diese Attribute mit der zugehörigen *transform node* verknüpft. Über das *transla-*

[17] Die schematische Darstellung beinhaltet nicht alle Elemente der profile node

tion-Attribut der *transform node* wird eine Verknüpfung hergestellt, um Maya die Möglichkeit zu geben, zu jeder *profile node* die korrekte *transform node* zu identifizieren. *simulation* enthält alle notwendigen Attribute zur Verwaltung der Animation und der Simulation (*action, actionI*). Darüber hinaus wird bei jeder Veränderung der Rotationsdaten des Agenten das Heading (Blickrichtung) neu berechnet.

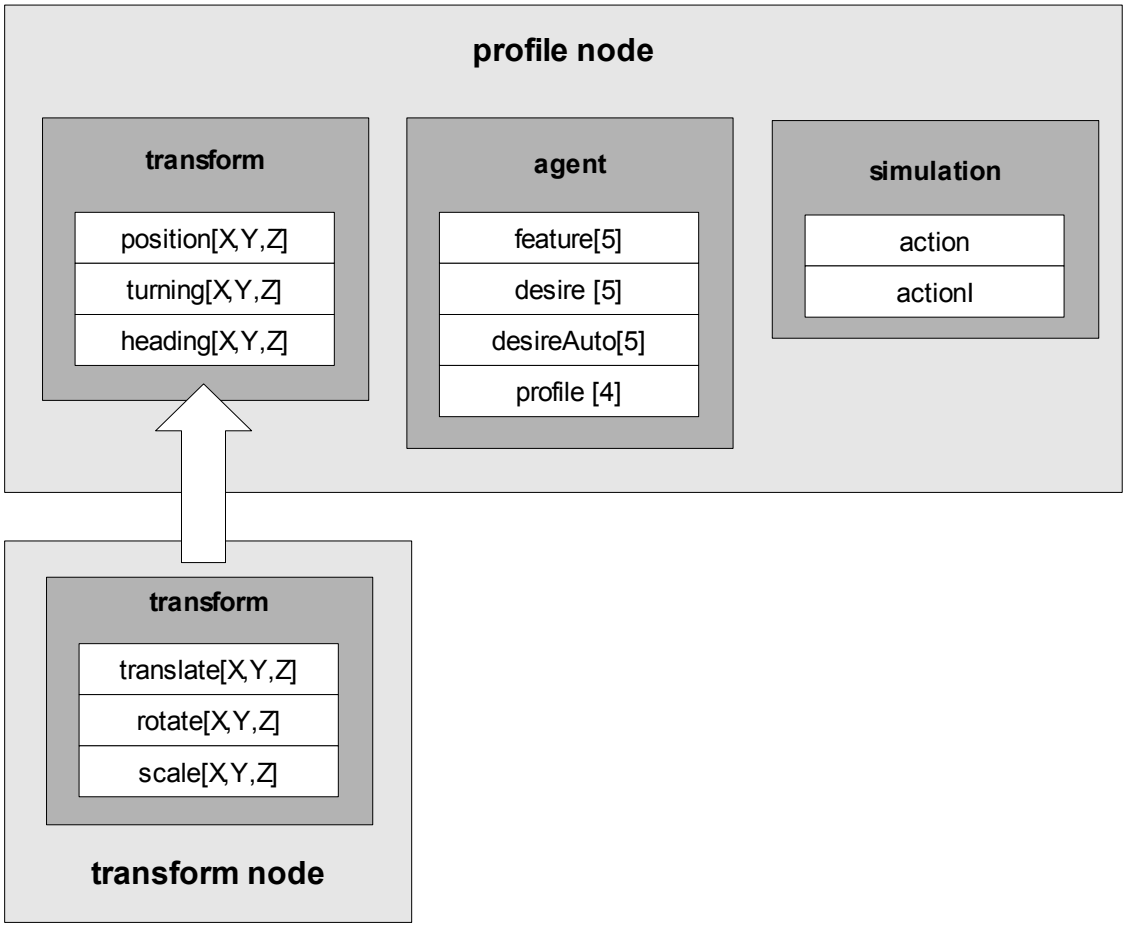


Abbildung 21: profile node

6.1.2 Update Profile Node

updatepn ist ein Kommando zum Versenden und Empfangen von Serveranfragen und -antworten. Beim Aufruf durch einen MEL-Scriptbefehl wird eine TCP/IP-Verbindung zum KI-Server aufgebaut. Anschließend wird eine List *pnList* aller in der Szene

befindlichen Agenten (identifiziert über eine Verknüpfung zu einer *profile node*) generiert und für jeden Eintrag wird eine Anfrage an den Server für eine Aktualisierung des Agenten gesendet.

```
// MSelectionList ist eine Maya Datenstruktur zur Verwaltung
// selektierter Objekte.
     MSelectionList pnList;
// Um eine Liste aller Agenten zu erhalten, werden die Namen
// aller DG Nodes mit einer regulären Ausdruck nach dem
// Schlüsselwort profile durchsucht und anschließend in die
// MselectionList eingefügt.
     MString match("profile*");
     MGlobal::getSelectionListByName(match, pnList);
     for(int i = 0; i < (int)pnList.length(); i++)
     {
        [...]
     }
```

Mit der Anfrage erhält der Server noch die Positions-, Heading-, Radius- und Desire-Daten des Agenten sowie eine Liste von sogenannten *Neighbours* für Agenten, welche sich im Sichtradius und Blickfeld des aktuell betrachteten Agenten befinden. Damit ein Agent einen anderen wahrnehmen kann, müssen zwei Bedingungen erfüllt sein. Der wahrzunehmende Agent (oder auch Nachbar) muss innerhalb des Sichtradius und im Blickwinkel des Betrachters sein. Die Entfernung lässt sich einfach durch eine Vektor-Subtraktion, der Winkel durch ein Vektor-Punktprodukt und den arccosinus berechnen. Da jeder Agent mit jedem verglichen werden muss, hat diese Funktion eine quadratische Laufzeit. Da die Wahrnehmung des Agenten in Maya selbst geschieht, ist die Verwaltung einer räumlichen Datenstruktur zur Verbesserung der Laufzeit (siehe Kapitel 7.2) nur schwer realisierbar. Die Datenstruktur selbst müsste in einer separaten Node gespeichert und bei jeder Bewegung eines Agenten balanciert werden, was einen enormen Overhead erzeugt, da Maya für die Organisation von solch großen Datenmengen nicht ausgelegt ist.

Nach dem Senden der Daten wartet der Befehl auf eine Antwort, welche nach Erhalt dir *profile node* aktualisiert.

6.1.3 Update Geometry

updateg ist eine Maya API-Kommando zur Aktualisierung der *transform node* eines Agenten. Die Daten für das Update werden aus den Attributen *velocity* und *turning* der *profile node*, welche zuvor von *updatepn* berechnet wurden, entnommen. Mittels des *MFnTransform* Function Sets auf der DG Node *nodeG,* wird der Agent rotiert und entlang seiner X-Achse transliert.

```
// erzeugen einer List aller Agenten (suchen nach allen Agenten
// mit einer "profile" node
     MSelectionList pnList;
     MString match("profile*");
     MGlobal::getSelectionListByName(match, pnList);
     [...]
// für jede Node:
     for(int i = 0; i < (int)pnList.length(); i++)
// erzeugen eines Mobjects nodeG zum verwalten der Node Daten
          MObject nodeG;
          [...]
// erzeugen eines Functions Sets zur Transfomration einer Node
          MFnTransform fTransformNode(nodeG);
// rotiere nodeG um turning Grad
          fTransformNode.rotateBy(turning,
            MTransformationMatrix::RotationOrder::kXYZ,
            Mspace::kTransform);
// transliere nodeG um velcoity entlang der X-Achse
          fTransformNode.translateBy(velocity,
            Mspace::kObject);
```

6.2 Server

Bevor der Client Anfragen an den Server stellen kann, muss dieser gestartet und konfiguriert werden. Für zukünftige Erweiterungen wurde bereits jetzt ein Windowsystem[18] verwendet, um bei Bedarf ein graphical user interface (GUI) für die Konfiguration und Analyse des Systems zu entwickeln.

[18] Über die Microsoft API WinAPI

Die Serveranwendung stellt das KI-System dar, welches das Verhalten der Agenten bestimmt. Im Folgenden werden zuerst die Komponenten beschrieben und anschließend ihre Funktion und ihre Abhängigkeiten erläutert.

profiles:

profiles ist ein STL[19] Vektor vom Typ *profile,* welcher die Agenten verwaltet. Sollte Maya einen neun Agenten erzeugen, wird in diesem Vektor abgelegt.

profile:

Der Agent und seine Parameter sind in der Klasse *profile* spezifiziert.

actionList:

actionList ist ein STL-Vektor vom Typ action (siehe Kapitel 6.4). Es ist ein Pool von Handlungen, die dem Agenten zur Verfügung stehen. In Abhängigkeit des Szenarios kann hier jede Art von Funktionen programmiert und der Liste hinzugefügt werden. Zukünftig ist geplant, dass beliebige Bibliotheken geladen werden können, welche dem Pool hinzugefügt werden können .

action:

action enthält die Metadaten, um eine Aktion mit einem Bedürfnis zu verknüpfen. In dieser Klasse werden alle Informationen über die Aktion (Dauer, Kosten, Beschreibung, Profil) sowie der Funktions Pointer zur eigentlichen Funktion gespeichert. Die Klasse hat die Methode *checkProfile(),* um ein Ranking für die Gleichheit des Profils des Agenten und der Aktion zu erstellen. Dadurch wird festgestellt, wie sehr diese Aktion zum Agenten „passt“. Der Funktions-Pointer muss der Deklaration

[19] STL – Standard Template Library

```
bool (*actionp) (Profile &profile,
                 std::vector<float> parameter,
                 std::vector<Profile> &profiles,
                 std::vector<FeatureCounter> featureCouter,
                 std::vector<FactCounter> k,
                 float response[2]
            );
```

entsprechen. Jede Aktion hat somit Zugriff auf alle Agenten, die aktuelle Wahrnehmung und die Knowledge Database. Jede Aktion liefert über *response* die Werte *velocity* und *turning* zurück, um das Verhalten des Agenten zu beschreiben. Wenn gefordert, kann die Aktion auch über den Rückgabetyp bool die Nicht-Ausführbarkeit einer Aktion mitteilen. Der Agent würde demnach in einer Situation eine Aktion ausführen wollen, deren Ausführbarkeit unbekannt ist, und stellt dies zur Laufzeit fest.

desireRules:

desireRules ist ein STL-Vektor vom Typ *desireRule* zum Verwalten der Updatefunktion der autonomen Bedürfnisse eines Agenten. Diese Struktur wird durch die Funktion r*eadDesires* der *configReader*-Klasse geladen und enthält die Methode *UpdateDesireAuto* zur Aktualisierung der Bedürfnisse eines Agenten. Zu diesem Zweck wird überprüft, ob der Agent ein Bedürfnis besitzt, für das eine Funktion definiert ist. Gegebenenfalls wird diese ausgeführt und der neue Wert des Bedürfnisses berechnet. Dieser Updateprozess wird in jedem Simulationsschritt wiederholt. Der Index des Vektors identifiziert das zu betrachtende Bedürfnis.

desireRule:

Eine *desireRule* beinhaltet die notwendigen Informationen zur Definition einer Updatefunktion. Der Parameter *function type* bestimmt die arithmetische Operation, die durchgeführt werden soll, die Parameterliste *parameter* den Parameter für die Operation.

desireActions:

desireActions ist ein STL-Vektor vom Typ *desireAction* zum Verwalten der Verknüpfung einer Aktion mit einem Bedürfnis. Jedes Bedürfnis erhält einen eigenen Vektor, sodass für jedes Bedürfnis mehrere Aktionen definiert werden können.

desireAction:

Die Metadaten für die Verknüpfung einer Aktion mit einem Bedürfnis sind in der *desireAction*-Klasse definiert. Die Verbindung wird über die ID des Bedürfnisses und der Aktion bewerkstelligt. Der Parameter *weight* beschreibt ein Gewicht für die Nützlichkeit einer Aktion für ein Bedürfnis, der Vektor *parameter* beinhaltet die Übergabewerte für die durch den Funktions-Pointer festgelegte Funktion in einem *action*-Objekt.

condition:

Eine *condition* beschreibt eine Bedingung. Sie definiert sich durch eine Zahl, eine Eigenschaft und einen Operationscode (*number*, *feature, opcode*). Es wird verglichen, ob die wahrgenommene Situation (beschrieben durch den Wahrnehmungsvektor *featureCounter*) die Bedingung erfüllt. Durch den Operationscode kann ein relationaler Operator (größer, kleiner, gleich) definiert werden. Die Anzahl der wahrgenommenen Eigenschaften wird anschließen über diesen Operator mit der in der Bedingung vorgegebenen Anzahl verglichen. Zusätzlich lässt sich noch eine minimale, maximale oder identische Entfernung für die Eigenschaft bestimmen (*distance, opcodeD*). Für diese Auswertung stellt *condition* die Funktion *evaluate* und *evaluateFact* bereit. Sie unterscheiden sich dadurch, dass *evaluate* auf der aktuell wahrgenommenen Situation und *evaluateFact* auf Basis der Knowledge Database arbeitet.

desire:

desire beschreibt die durch den Nutzer festgelegten Bedürfnisse des Agenten. Sie bilden die Motivatoren für den Agenten, eine bestimmte Aktion zur Befriedigung des Bedürfnisses auszuwählen. Die Priorität bestimmt die Wichtigkeit einer Befriedigung und wird in Maya festgelegt und bei jedem Update neu ausgelesen.

desireAuto:

Im Gegensatz zu den in Maya festgelegten Bedürfnissen ist ein autonomes Bedürfnis zwar vom Nutzer initialisiert, wird aber durch das KI-System gesteuert. Der Wert lässt sich nur durch eine *desireRule* oder durch eine *action* beeinflussen. Die eigentliche Priorität dieser Gruppe von Bedürfnissen bestimmt sich durch die Differenz zwischen einem Soll- und Istwert. Diese Differenz wird durch die Funktion *calculatePriority* durch eine weitere Funktion (logarithmisch, exponentiell, Absolut-Betrag, Sigmoid) geführt, um die finale Priorität zu berechnen.

featureCounter:

Der *featureCounter* ist ein STL-Vektor vom Typ *feature*. Sein Index entspricht der ID der Eigenschaft. Die Eigenschaften werden aus den *features* der wahrgenommenen Agenten ermittelt. Bei jedem Simulationsschritt wird der *featureCounter* mit diesen Werten gefüllt. Für jeden Index des Vektors wird die Anzahl der Einträge geprüft und so eine *condition* ausgewertet.

feature:

feature erweitert den *featureCounter* um die Parameter *profile* und *distance*, sodass ein Bezug zum Agenten hergestellt werden kann, der die gesuchte Eigenschaft besitzt und in welcher Entfernung sie wahrgenommen wurde.

factCounter:

Ähnlich wie *featureCounter* ist *factCounter* eine Sammlung aller wahrgenommenen Eigenschaften. Jedoch wird dieser nicht bei jedem Simulationsschritt neu generiert, sondern enthält alle je wahrgenommenen *features*. Diese sind mit einem Zeitstempel versehen, sodass ihre Beständigkeit bewertet werden kann. Je weiter zurück ein *feature* wahrgenommen wurde, desto unwahrscheinlicher, dass es noch immer gültig ist. Jeder Agent besitzt einen eigenen factCounter, der seine Knowledge Database bildet.

fact:

fact erweitert den *factCounter* um den Parameter *time*, um eine Auswertung bezüglich der Gültigkeit zu ermöglichen. Da das Objekt *fact* von *feature* erbt, sind ebenfalls die Parameter *profile* und *distance* verfügbar.

rules:

rules ist ein Vektor zum Verwalten aller Regeln des Systems. Sie sind unabhängig vom Agenten und werden ausgeführt, sobald eine Situation – bedingt durch *conditions* – gegeben ist. Die Regeln werden von der configReader-Klasse gelesen und in einen Vektor geschrieben. *rules* stellt die Funktion *evaluate* zur Auswertung aller Regeln bereit. Es wird über alle Indizes iteriert und die Evaluierungsfunktion des *rule*-Objekts aufgerufen. Sollte die Bedingung der Regel erfüllt sein, so wird die Aktion *action,* die mit dieser Regel verknüpft ist, in den *validActions*-Vektor eingefügt und steht als mögliche Handlungsoption bereit.

rule:

Eine Regel *rule* besteht aus mindestens einer Bedingung *condition* für die Regel, mindestens einer Bedingung für die Aktion, die mit der Regel assoziiert ist, sowie einem vollständigen Profil eines Agenten (*feature, desire, desireAuto, profile*). Die Funktion *evaluate* des *rule*-Objekts wertet die Bedingungen und das Profil aus und gibt das Ergebnis zurück.

validActions:

validActions ist ein STL-Vektor, der alle validen, also ausführbaren Aktionen nach Auswertung der Bedürfnisse und Regeln des Systems enthält. Für jede Aktion werden die Wahrscheinlichkeit einer Ausführung, die Indizes der Aktion und des Bedürfnisses und die Parameter der Aktion gespeichert

Ablauf:

Nach der Initialisierungsphase für die Netzwerk-Socket-Kommunikation und die Fensterklassen lädt der Server über die *configReader*-Klasse das Expertensystem und erzeugt die Aktionsliste. Anschließend befindet sich das Programm in einer Simulationsschleife. Zu Beginn werden über die WinAPI[20] die Nachrichten des Windows-Systems (Knopfdruck, Mouse, Click etc.) abgefragt und bearbeitet. Anschließend geht die Anwendung in den sogenannten Listening State der Netzwerkkommunikation und wartet auf ankommende Verbindungsanfragen des Client. Solange keine Nachricht ankommt, befindet sich die Anwendung in einem Blocking State und lässt keine andere Funktion zu. Sendet der Client eine Anfrage, wird durch die *accept*-Funktion eine TCP/IP-Verbindung aufgebaut. Da nun eine TCP/IP-Verbindug besteht, können Datenpakete gesendet und empfangen werden. Dazu geht die Anwendunge in einer weiteren Receive-Schleife. Durch den *recv*-Befehl werden ein weiterer Blocking State gestartet und die Nachrichten empfangen. Der Header der Nachricht wird ausgewertet und bestimmt die weitere Verarbeitung und Auswertung des Verhaltensmodells. Abschließend wird die Antwort an den Client gesendet und der Server geht zurück in den Receive State für den Empfang einer weiteren Nachricht. Auf diese Weise werden alle Agenten aktualisiert.

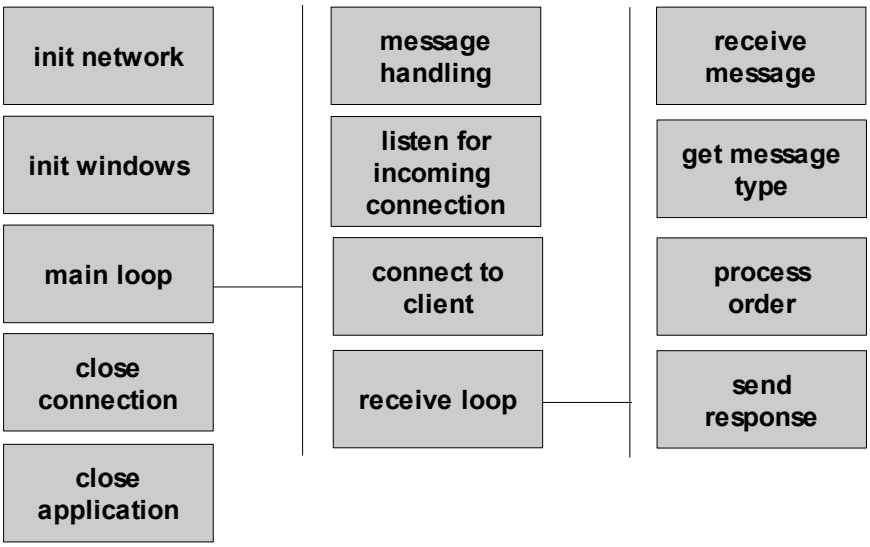


Abbildung 22: Server - Ablauf

[20] Programmierschnittstelle zur Verwendung des Microsoft Windows Systems

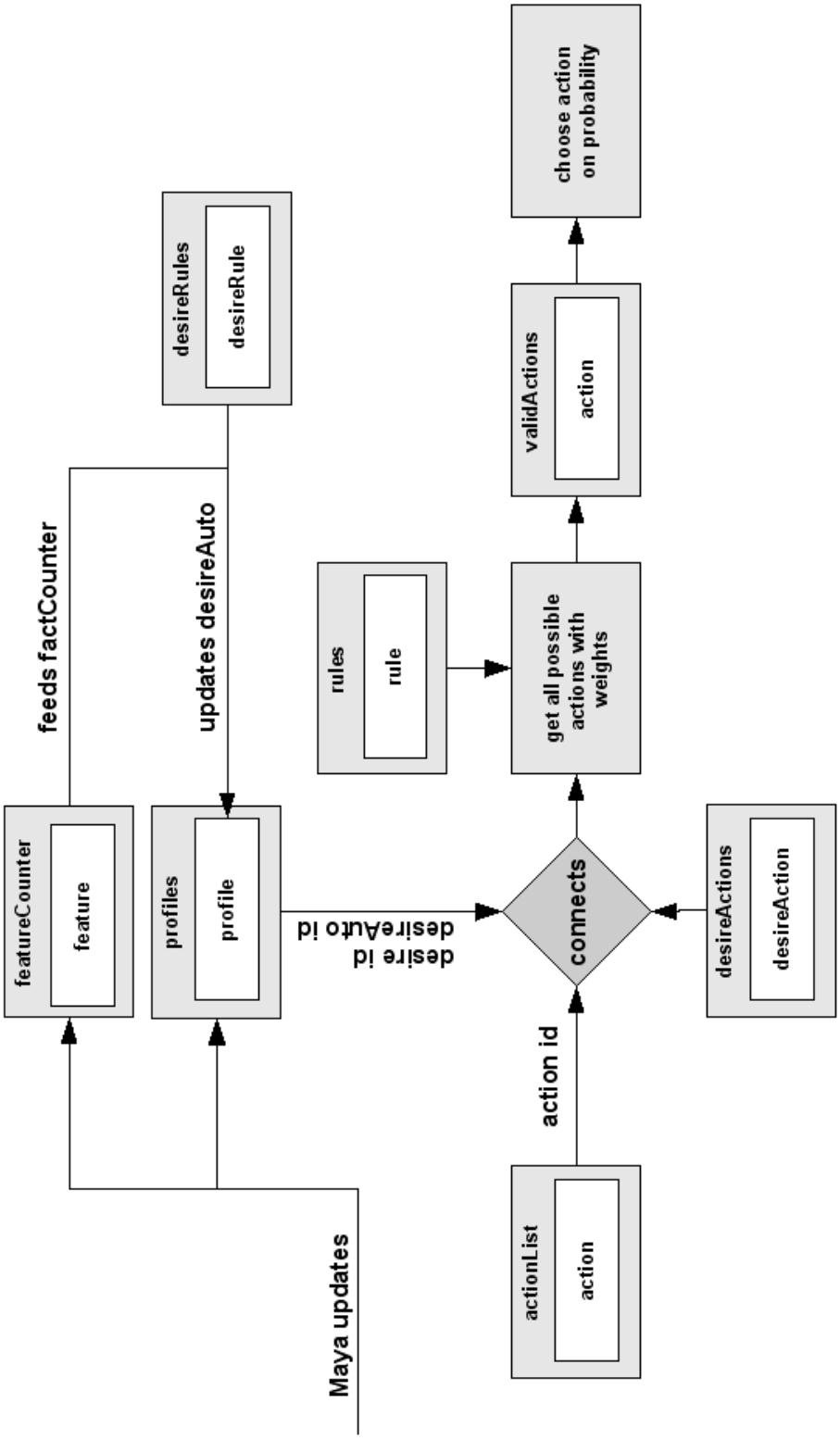


Abbildung 23: Server - Komponenten

6.2.1 Auswertung des Verhaltensmodells

Die Auswertung ist das Herzstück des KI-Systems. Es verwaltet die Auswertung der Bedürfnisse und Regeln und ihre Konkurrenz, um befriedigt bzw. ausgeführt zu werden. Die Auswertung ist in vier Phasen unterteilt.

1. Phase – Anfrage verarbeiten

Für jede Anfrage vom Client mit der *order* 2 wird eine neue Berechnung durchgeführt. Aus dem Datenpaket extrahiert der Tokenizer den Agenten, seine Position, seine Blickrichtung, seine Bedürfnisse (nicht die autonomen Bedürfnisse, diese werden auf Serverseite ausgewertet) und die Nachbarn. Um die gesamte Wahrnehmung zu bestimmen, werden die Eigenschaften aller Nachbarn bestimmt. Diese *features* werden sowohl in den *featureCounter* als auch in den *factCounter* geschrieben. Für jeden *fact* in *factCounter* wird der Zeitstempel um eins inkrementiert.

2. Phase – Regeln

Die im *rules*-Objekt gegebenen Regeln werden durch die *evaluate*-Funktion des *rule*-Objekts ausgewertet und bei positivem Ergebnis wird bestimmt, ob sie zu den *validActions* hinzugefügt wird. Um gültig zu sein, muss die Regel allen Bedingungen (*conditions*) entsprechen. Alle Regeln erhalten die Wahrscheinlichkeit 1.0. Durch Herab- oder Heraufsetzen der Wahrscheinlichkeit lässt sich der Vorzug einer Regel gegenüber einem Bedürfnis beeinflussen. Je höher dieser Wert, desto eher wird eine Regel gewählt. Um als Nächstes die Bedürfnisse zu bestimmen, müssen alle autonomen Bedürfnisse der in *desireRule* festgelegten Aktualisierung durchgeführt und ihre Priorität anhand der in *desireAuto* bestimmten Funktion bestimmt werden.

3. Phase – Bedürfnisse

Jeder Agent besitzt zwei Arten von Bedürfnissen – Bedürfnisse (*desire*) und autonome Bedürfnisse (*autoDesire*). Jedes Bedürfnis besitzt eine Priorität, anhand derer die Wahrscheinlichkeit einer Wahl bestimmt wird. Bei den Bedürfnissen, die durch die Clientanwendung festgesetzt werden, steht die Priorität durch den Benutzer bereits fest. Bei den autonomen Bedürfnissen muss diese erst berechnet werden. Dazu wird die Differenz von Ist- und Sollwert des Bedürfnisses durch die

UpdateDesireAuto- Funktion und die in *desireRules* definierte Regel aktualisiert. Anschließend werden mit der *CalculatePriority* die Differenz zwischen Ist- und Sollwert gebildet und über eine vorher definierte Funktion die Priorität berechnet.

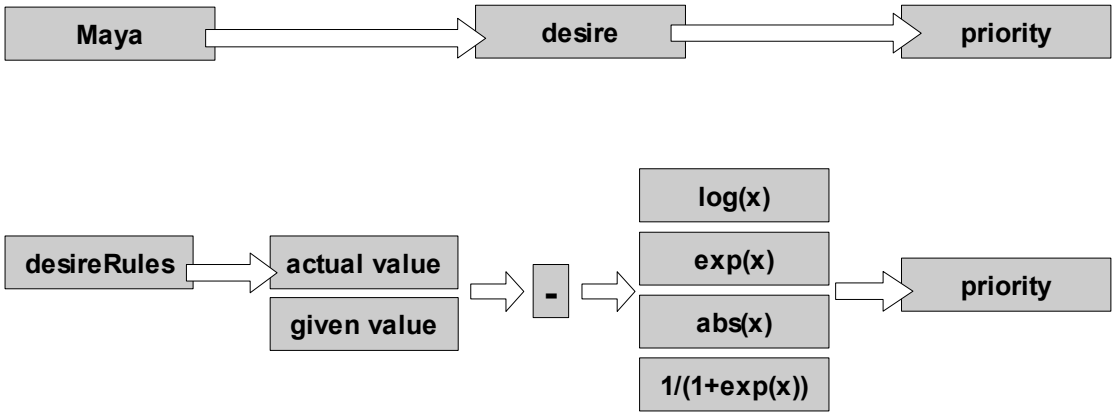


Abbildung 24: Server - Bedürfnisse(1)

Die Priorität bestimmt in beiden Fällen die Wichtigkeit des Bedürfnisses. Je größer die Differenz zwischen Ist- und Sollwert, desto größer der Drang, dieses Bedürfnis zu stillen. Je nach Funktion wirkt sich die Differenz mehr oder weniger stark auf die Priorität aus. Beispielsweise steigt der Drang bei einer logarithmischen Funktion schnell an und bleibt dann annähernd konstant.

Nachdem die Prioritäten feststehen, werden für beide Typen valide Aktionen gesucht, indem für jede Aktion, die über *desireActions* einem Bedürfnis zugeordnet ist, die Bedingungen (*conditions*) ausgewertet werden. Anschließend werden zwei Utilities berechnet. Die *conditionUtility* gibt an, wie gut eine Aktion zum Profil des Agenten passt (über die Anzahl gleicher Flags im Persönlichkeitsarray), die *utilityAction* bestimmt hingegen, wie gut die Aktion das Bedürfnis befriedigt. Dazu wird eine Differenz zwischen Kosten und Nutzen gebildet. Die Kosten sind fix im *action*-Objekt festgelegt, der Nutzen ist hingegen von dem jeweiligen Bedürfnis abhängig, welches mit der Aktion verknüpft ist. Es ist im *desireAction*-Objekt festgelegt. Beide Utilities werden miteinander multipliziert.

Es wird über alle validen Aktion iteriert und es werden die ausgewählt, welche die höchste *utilityAction* aufweisen. Um die finale *utilityMaster* zu bestimmen, wird die *utilityAction* mit der zuvor berechneten Priorität multipliziert. Dieses Produkt ist somit ein Grad für die Wichtigkeit eines Bedürfnisses, kombiniert mit der Möglichkeit einer passenden Aktion. Zum Schluss werden alle Wahrscheinlichkeiten normalisiert und dem *validAction*-Vektor hinzugefügt.

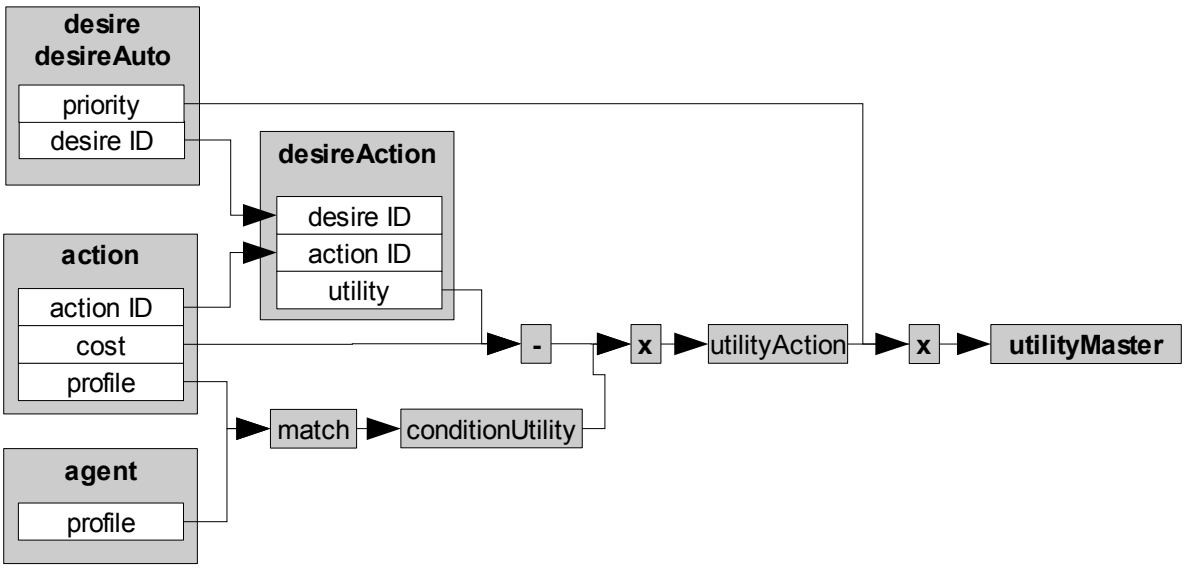


Abbildung 25: Server - Bedürfnisse(2)

Damit der Agent nicht von Schritt zu Schritt seine Aktion wechselt und so keine vollständig ausführt, wird der aktuell ausgeführten Aktion ein Bonus auf seine Wahrscheinlichkeit gewährt. Dieser Bonus wird *commitment* genannt und beschreibt somit die Bereitschaft, eine einmal getroffene Handlung beizubehalten (siehe Kapitel 7.1).

Nun stehent für die nächste Phase für jedes Bedürfnis die beste Aktion zur Befriedigung und die Wahrscheinlichkeit zur Verfügung.

4. Phase – Aktionsselektion und Exekution

Zu Beginn wird der *validActions*-Vektor über eine Zufallsfunktion ungeordnet, sodass sich die Aktionen bei jedem Aufruf in einer anderen Reihenfolge befinden. Die Wahrscheinlichkeit jeder Aktion gibt ihre Wahrscheinlichkeit einer Wahl vor. Da die Werte in Phase 3 normalisiert wurden, kann davon ausgegangen werden, dass die höchste Wahrscheinlichkeit 100% beträgt und alle anderen linear dazu abgestuft sind (die Werte zwischen 0 und 1 werden mit 100 multipliziert, um eine andere Spreizung der Werte zu erhalten). Anschließend wird eine Zufallszahl zwischen 1 und 100 ermittelt. Sollte diese kleiner als die Wahrscheinlichkeit der Aktion sein, so wird die Aktion ausgeführt. D.h., dass bei einer Wahrscheinlichkeit von 100 die Aktion in jedem Fall, bei einer Wahrscheinlichkeit von 0 auf keinen Fall ausgeführt wird. Sollten mehrere 100%-Aktionen vorhanden sein, wird durch die anfangs vorgenommene Zufallssortierung ein Ausgleich geschaffen.

Nachdem eine Aktion selektiert wurde, wird überprüft, ob sie ausführbar ist (dies geschieht in der Funktion der Aktion – nicht jede Aktion muss diese Eigenschaft überprüfen). Im positiven Fall gibt die Aktion ihre Antwort und somit das folgende Verhalten des Agenten durch das Array *response* zurück. Die Aktion wird im Profil des Agenten gespeichert. Im negativen Fall wird die Aktion verworfen und die nächste Aktion im Vektor überprüft. Dieser Vorgang wird so lange wiederholt, bis eine ausführbare Aktion gefunden wird. Um einer Endlosschleife vorzubeugen, wird bei einer leeren Liste oder einer mit zu niedrigen Wahrscheinlichkeiten keine Aktion selektiert, und der Agent wartet auf den nächsten Simulationsschritt.

6.3 Datenaustausch und -formate

Der Datenaustausch zwischen Maya und dem Server wird über das Versenden von ASCII-kodierten Datenpaketen über das TCP/IP-Protokoll erreicht.

Im Folgenden wird die Formatierung der Datenpakete erläutert. Die einzelnen Werte sind in dem übertragenen Paket durch Kommas getrennt und werden über einen Tokenizer dekodiert und anschließend verwertet.

Client Anfragen

Jede Anfrage besitzt ein Zahlenwert (*order*), welcher die Formatierung der Anfrage (*request*) bestimmt.

Abbildung 26: Datenpaket – Order

Momentan sind vier Anfragetypen implementiert. Der *name* Wert des Pakets identifiziert den zu bearbeitenden Agenten.

Typ 1: Creation

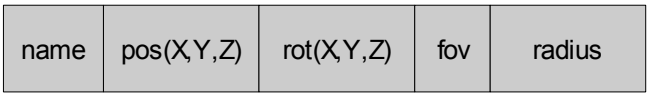


Abbildung 27: Datenpaket – Creation

Die *Creation* Anfrage von Maya erzeugt einen Agenten auf dem Server. Er erhält einen Namen (n*ame*), Positions- (p*os*) und Orientierungsdaten (r*ot*), sowie den Blickwinkel (*fov*) und Sichtweite (*radius*) für den Konstruktor. Anschließend wird der Agent in den *pnNodesVector* eingefügt.

Typ 2: Calculate Behaviour

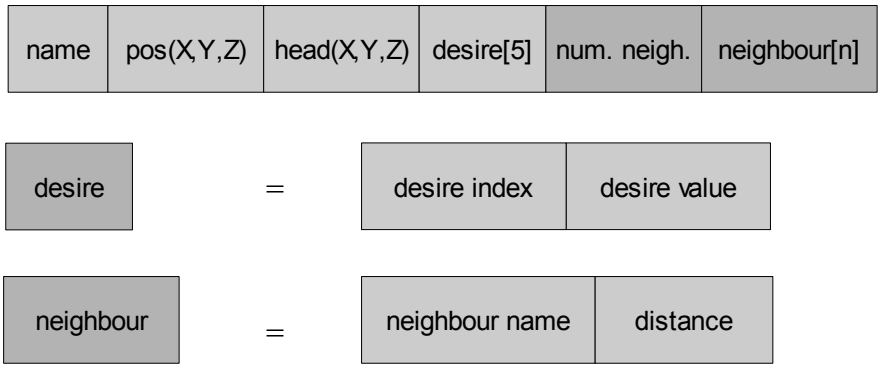


Abbildung 28: Datenpaket – Calculate Behaviour

Neben den Daten zur Wahrung der Konsistenz (*name, pos, head*) werden die Bedürfnisse (*desire*) und die wahrgenommenen Agenten (*number neighbours, neighbour*) übermittelt. Ein Bedürfnis besteht aus einer ID (*desire index*) zur Identifikation und einem Wert (*value*). Der Server kann so die vom Benutzer festgelegten Bedürfnisse aktualisieren. Der Wert bestimmt die Priorität des Bedürfnisses und konkurriert mit den autonomen Bedürfnissen, welche vom Server berechnet werden, um eine Befriedigung. Die wahrgenommenen Nachbarn werden durch einen eindeutigen Namen identifiziert und erhalten als zusätzliche Information die Entfernung für die Generierung des Perzepts.

Typ 3: Update Agent (Feature, Desire, Profile)

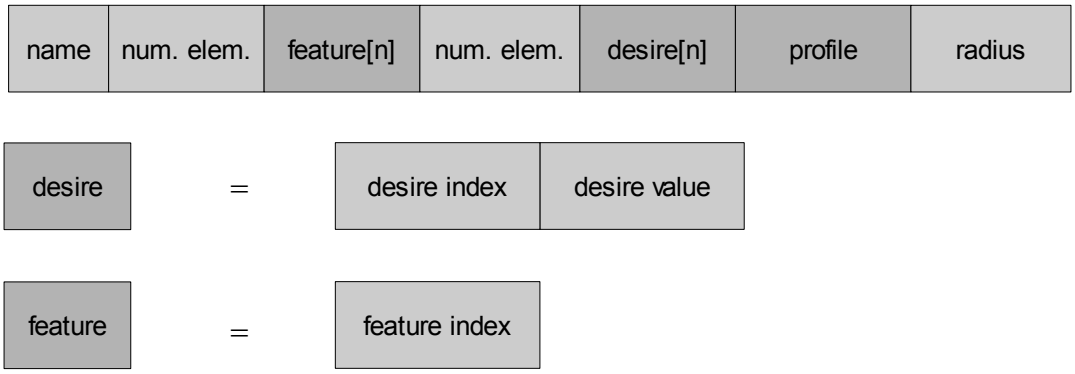


Abbildung 29: Datenpaket – Update Agent (1)

Die Anpassung eines Attributs des Agenten löst eine Aktualisierung des Abbildes auf dem Server aus. Der Einfachheit halber, wurden die die Attribute in zwei Gruppen unterteilt, da bei jeder Aktualisierung Daten aller Attribute einer Gruppe gesendet werden müssen.

Typ 4: Update Agent (DesireAuto)

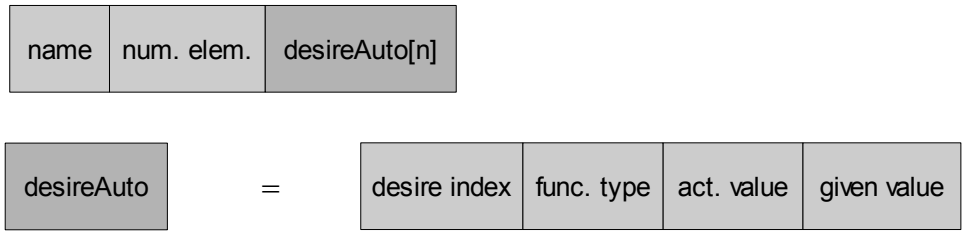


Abbildung 30: Datenpaket – Update Agent (2)

Die autonomen Attribute des Agenten werden vom Server berechnet. Der Benutzer erhält die Möglichkeit auf die Art der Berechung Einfluß zu nehmen und die Update-

funktion zu bestimmen. In der Regel werden diese Attribute nur zu Beginn der Simulation gesetzt und im Laufe der Animation vom Server berechnet.

Server Antworten

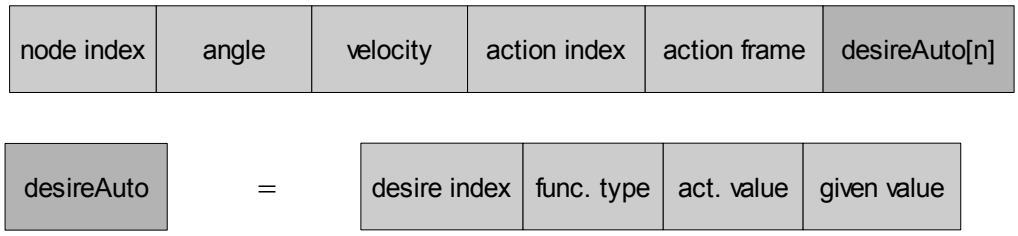


Abbildung 31: Datenpaket – Server Antwort

Der Server besitzt ein Antwortformat (siehe Abbildung 18). Maya befindet sich noch in der Blocking Phase der Client-Server-Kommunikation und führt das Programm erst nach vollständigem Erhalt des Pakets weiter.

Das Antwortpaket enthält alle Daten, die der Agent auf Clientseite benötigt. Der Wert *angle* gibt in Grad die Rotation des Agenten entlang seiner Rotationsachse an, *velocity* die Geschwindigkeit und damit die zurückzulegende Entfernung entlang des Heading-Vektors. *action index* und *action frame* geben dem Benutzer Indikatoren zur Verwendung der korrekten Animation. *action index* bestimmt die zu verwendende Animation und *action frame* die Phase – das Frame – der Animation.

Konfigurationsdateien:

Um das Expertensystem, welches das Verhaltensmodell beschreibt, zu konfigurieren, werden derzeit drei Konfigurationsdateien verwendet. Es sind ASCII-kodierte Textdateien, welche zeilenweise ausgelesen werden und deren Werte durch Kommas getrennt sind. Jeder Index zur Identifikation eines Bedürfnisses, einer Aktion oder einer Funktion wird ebenso wie die Anzahl der folgenden Parameter oder Bedingungen durch einen Integerwert dargestellt. Alle anderen Werte werden als Float ausgewertet.

DesireRules:

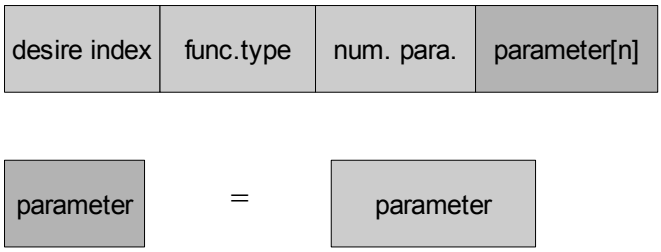


Abbildung 32: DesireRules

Jede Zeile der DesireRules-Konfigurationsdatei beschreibt eine Regel zur Aktualisierung eines Bedürfnisses. Das durch eine ID (*desire index*) gekennzeichnete Bedürfnis wird durch die in *function type* und *parameter* (number parameter, parameter) festgelegten Werte berechnet.

DesireActions:

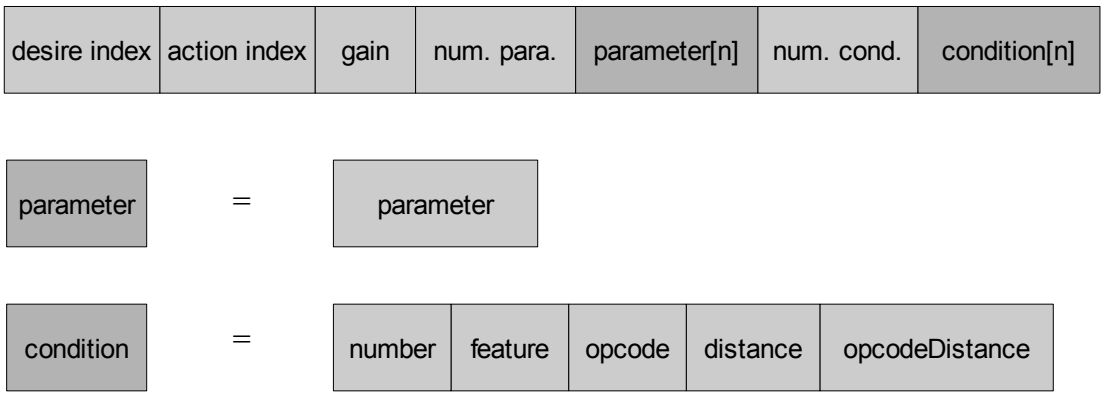


Abbildung 33: DesireActions

DesireActions beschreibt das eigentliche Expertensystem. Es formuliert die Verbindung zwischen einer Aktion (*action index*) und einem Bedürfnisse (*desire index*), dessen Gewicht (gain), Parameter (*number paramter, parameter*) und Bedingungen (*number conditions, conditions*).

Rules:

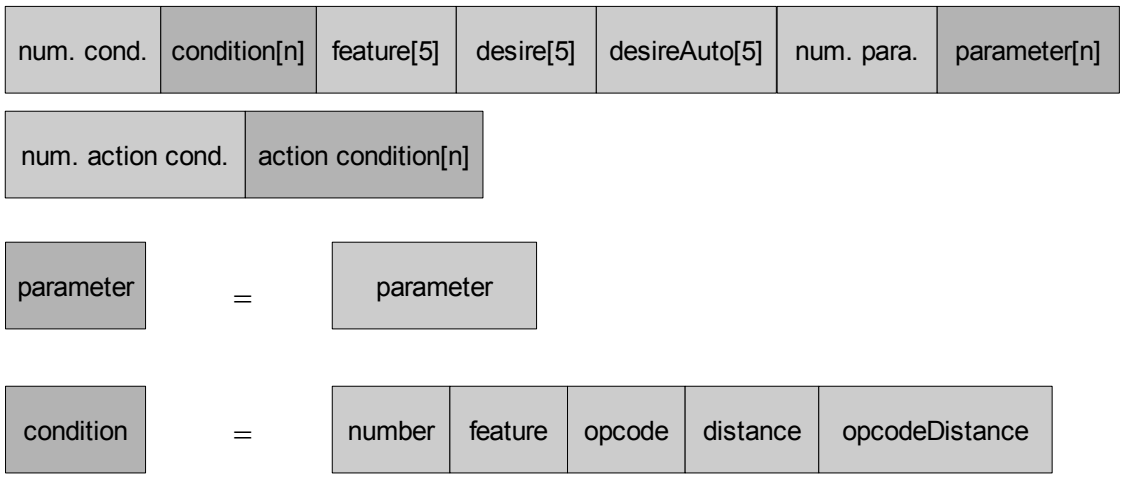


Abbildung 34: Rules

Rules lädt die Bediungen (*number condition, condition*) unter denen die Aktion ausgeführt wird, *feature, desire* und *desireAuto* das Profil das der Agent aufweisen muss. Die Parameter (*number parameter, parameter*) werden zum Auswerten der Aktion verwendet, welche ebenfalls einer bliebigen Anzahl von Bedingungen (*number action condition, condition*) genügen muss.

6.4 Aktionen

Das BDI-Modell des KI-Systems bestimmt nach Abwägen aller Faktoren eine Aktion. Diese Aktion wird durch eine ID eindeutig aus der Liste aller verfügbaren Aktionen (*actionList*) identifiziert und anschließend ausgeführt.

Eine Aktion besteht aus einer ID zur Identifikation, Kosten (*cost*) für die Kosten-Nutzen-Analyse, einer Dauer in Simulationsschritten und einem Profil zum Abgleich mit der Persönlichkeit eines Agenten. Durch diese Daten kann das BDI-Modell einen Utility-Wert für jede Aktion und jedes Bedürfnis berechnen und die Wahscheinlichkeiten einer Ausführung bestimmen. Die Bedingungen, unter denen die Aktion ausgeführt werden kann, werden durch die Konfiguration der DesireActions und Rules definiert, daher war eine redundante Einbindung in die *Action*-Klasse nicht notwendig. Darüber hinaus lassen sich so gleiche Aktionen unter verschiedenen Bedingungen ausführen. Jede Aktion ist über einen Funktionspointer zu einer C++-Funktion verbunden, wel-

che die Manipulation des Agenten vornimmt. Die Funktion hat Zugriff auf alle relevanten Daten und auch die Berechtigung, diese zu verändern: das eigene Profil, eine Liste der Profile aller Agenten, Funktionsparameter, die aktuell wahrgenommenen Eigenschaften (*features*) und die Knowledge Database (alle je wahrgenommenen *features* des Agenten). Als Rückgabe liefert die Funktion zwei Werte: den Winkel, um den sich der Agenten drehen muss, und die Aktion, welche er ausführen soll.

Die Funktion *buildActionList* der Klasse *ActionList* erzeugt einen Vektor mit allen verfügbaren Aktionen und setzt die Parameter. Durch diesen Vektor werden die Aktionen dem Hauptprogramm zugänglich gemacht. Das Hinzufügen neuer Funktionen ist über das Erweitern der *buildActionList*-Funktion möglich.

```
// erzeugen einer neuen Aktion i
     Action *actionI = new Action;
// definition der Parameter der Aktion, functionPointer ist ein
// Funktionspointer zu der verwendenden Funktion für die
// Aktion
     actionI->actionp = functionPointer;
     actionI->id = 1;
     actionI->timesteps = 1;
     actionI->cost = 0;
// Hinzufügen der Aktion zum Vektor aller Aktionen
     this->actions[i] = actionI;
```

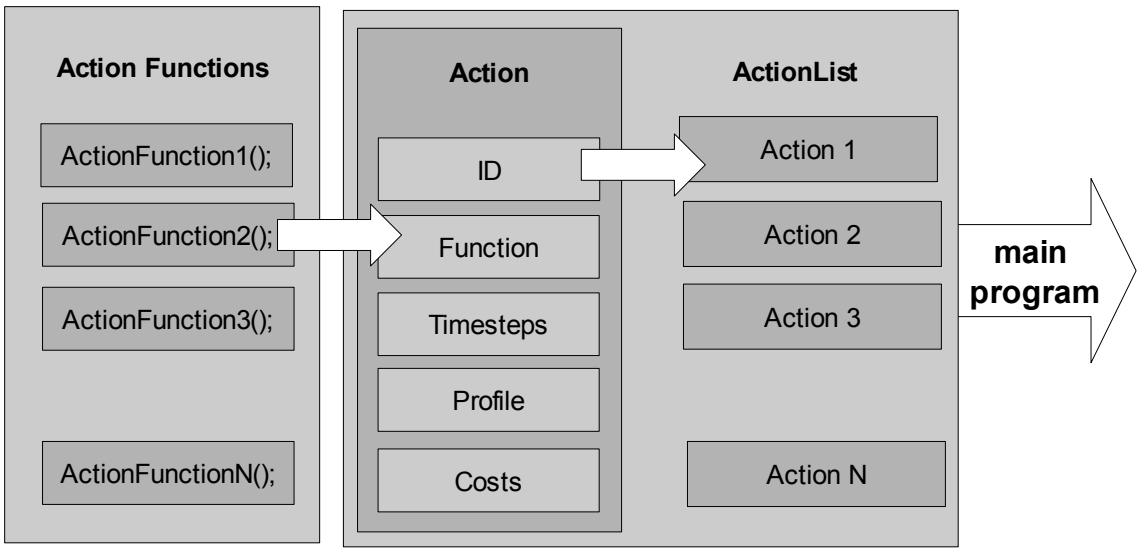


Abbildung 35: Aktionen

7 Ergebnisse

Im Folgenden wird die funktionale Umsetzung der in Kapitel 5 vorgestellten Architektur untersucht. Alle Anforderungen müssen erfüllt und in die Software Maya implementiert sein. Anders als bei einer rein formalen Analyse ist das Ergebnis der Implementierung subjektiv zu betrachten. Das gezeigte Verhalten der Agenten muss natürlich wirken und für einen Menschen intuitiv generierbar und abschätzbar sein. Zur Beweisbarkeit der Kernfunktionalität wird unter 7.1 ein Proof of Concept implementiert.

7.1 Funktionstest

Alle Untersystem des NDM-Modells müssen funktionieren, um aus den Umweltdaten und den internen Parametern des Agenten eine Aktion zu generieren.

Datenaustausch: Maya, Server

Mittels des Kommandos *updatepn* werden die Positions-, Rotations und Desiredaten an den Server geschickt (für die Aufschlüsselung der übertragenen Daten siehe Kapitel 6.4).

```
Send:

2; profile6; -18.4461, 0.734426, -8.15938, 0.934767, 0, 0.355262;
1, 0.500000, 0, 0.000000, 0, 0.000000, 0, 0.000000, 0, 0.000000;
3; profile3, 22.679838,      profile4, 23.681640, profile5,
18.644412
```

Abbildung 36: Output Window Maya

Datenaustausch: Server, Maya

Das Serverprogramm hält diese Daten in der Datenstruktur *Profile* zur weiteren Verwendung vor. Durch das Kommando *updatepn* wird ebenso der Updateprozess ausgelöst, welcher die neuen Positions-, Rotations- und Desire-, sowie die Aktionsdaten berechnet und an Maya zurücksendet.

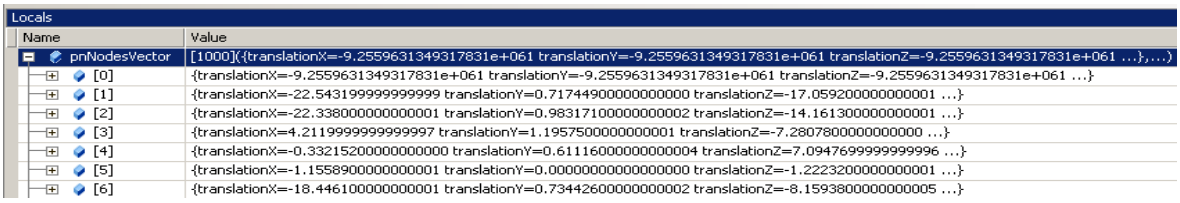


Abbildung 37: MS Visual Studio Debugausgabe

Somit wird gewährleistet, dass zu jeder Zeit auf dem Server ein Spiegel der Parameter aller Agenten verwaltet wird.

Zur Überprüfung der Funktionalität wird ein vollständiger Testfall (der alle Funktionen umfasst) entworfen und zum Testen der Verhaltensanpassung modifiziert.

Der Aspekt der natürlichen Verhaltensweise eines Agenten ist über diese Arbeit nur bedingt überprüfbar. Den größten Einfluss auf die Ausprägung des Verhaltens hat die mit einer Aktion verknüpfte und vom Animator erzeugt Animation. Worauf bei dem in dieser Arbeit entwickelten Entscheider zu achten war, ist das Verfolgen einer Aktion, nachdem der Agent sich zu dieser entschlossen (commitet) hat (siehe Tabelle 5). Bei der Implementierung wurde darauf geachtet, dass die in der Aktionsliste definierte Dauer einer Aktion vollständig abgelaufen sein muss, bevor der Agent sich zu einer neuen Aktion entschließen kann. Das natürliche Entscheiden liegt schließlich in der Logik des vom Nutzer festgelegten Expertensystems. Wenn das Expertensystem ein natürliches Verhalten beschreibt, so folgt der Agent diesem. Dadurch ist dem Nutzer auch ein unnatürliches Verhalten für seine Agenten möglich. Im Gegensatz zu einer unnatürlichen Ausprägung (Animation) des Verhaltens ist eine unnatürliche Entscheidungsfolge weniger schwerwiegend für eine mangelhafte Simulation.

Einen weiteren großen Einfluss auf ein natürliches Verhalten ist die in Kapitel 8 erwähnte Wegfindung. Eine zukünftige Arbeit die das Framework um diesen Faktor erweitert, muss darauf achten, dass der Agent einen natürlichen Weg zurücklegt.

Testfall:

Ein typischer Fall von Crowd Simulation ist die Simulation von Menschen in einem Stadion. Es werden typische Verhaltensweisen, wie Jubeln, Buhen, Essen sowie dazugehörige Bedürfnisse definiert. Als Resultat sollen die Menschen innerhalb des Stadions eine Auswahl für ihre nächste Aktion treffen.

Agenten:

[1] Zuschauer

Der simulierte Zuschauer

[2] Getränkestand

Ein Ort innerhalb der Szene, an dem Agent[1] etwas trinken kann.
Identifiziert durch eine Eigenschaft/feature mit Wert 2.

[3] WC

Ein Ort innerhalb der Szene, an dem Agent[1] sich erleichtern kann.
Identifiziert durch eine Eigenschaft/feature mit Wert 3.

[3] Sitzplatz

Der Sitzplatz des Agenten, an dem er das Spiel verfolgen kann.
Identifiziert durch eine Eigenschaft/feature mit Wert 4.

Rules:

[1] Ausweichen

Sollte der Agent ein Hindernis (ein anderer Agent) vor sich erkennen, wird er versuchen Auszuweichen oder stehen bleiben.

Desires:

[1] Das Spiel verfolgen

Sollte der Agent kein anderes Bedürfnis besitzen, wird er versuchen das Spiel zu verfolgen.

DesireAutos:

[2] Durst

[3] Erleichterung

DesireRules:

[1] Durst

Das Bedürfnis DesireAutos[2] Durst steigt linear mit fortschreitender Zeit. Der Faktor der Steigung hängt mit dem Profil des Agenten zusammen.

DesireActions:

[1] Durst -> Trinken

[2] Durst -> Gehen(2)

[3] Erleichterung -> WC

[4] Erleichterung -> Gehen(3)

Profil 2200 alle Profile mit xxTJ

[4] Erleichterung -> Rennen(3)

Profil 2211 alle Profile mit xxFP

[5] Spiel verfolgen -> Spiel

[5] Spiel verfolgen -> Gehen(4)

Actions:

[0] Nichts tun

Parameter: -

Beschreibung: -

Bedingungen: -

[1] Gehen

Parameter: X (Eigenschaft/feature)

Beschreibung: Richte dich auf die Position eines Objektes mit der Eigenschaft X aus und gehe entlang dieses Vektors darauf zu

Bedingungen: Objekt mit Eigenschaft X bekannt

Profil: 2222

[2] Rennen

Parameter: X (Eigenschaft/feature)

Beschreibung: Richte dich auf die Position eines Objektes mit der Eigenschaft X aus und gehe weit entlang dieses Vektors darauf zu

Bedingungen: Objekt mit Eigenschaft X bekannt

Profil: 2211

[3] Trinken

Parameter: -

Beschreibung: Trinken

Bedingungen: Distanz zu Objekt mit Eigenschaft 2 muss kleiner als 1 sein (Der Agent befindet sich an einem Ort, wo etwas getrunken werden kann)

[4] WC

Parameter: X (Eigenschaft/feature)

Beschreibung: Erleichtern

Bedingungen: Distanz zu Objekt mit Eigenschaft 3 muss kleiner als 1 sein (Der Agent befindet sich an einem Ort, wo er sich erleichtern kann)

[5] Spiel

Parameter:	X
Beschreibung:	Verfolge das Spiel
Bedingungen:	Distanz zu Objekt mit Eigenschaft 4 muss kleiner als 1 sein (Der Agent befindet sich an einem Ort, wo er sich erleichtern kann)

Folgend wird die Verteilung der Aktionswahl eines Agenten untersucht. Aufgrund der hohen Abhängigkeit der determinierenden Faktoren voneinander werden einzelne Fälle nur ausschnittsweise betrachtet. Eine Beobachtung der Aktionswahl unter Berücksichtigung aller Kombinationen von Bedürfnissen ist zur Betrachtung der allgemeinen Verhaltensanpassung nicht notwendig.

Eine Auswertung des Verhaltens ist nur auf Basis von statistischen Verteilungen der gewählten Handlungen des Agenten möglich. Daher werden in den folgenden Tabellen Wahrscheinlichkeiten für eine bestimmte Aktion unter bestimmten Bedingungen angegeben (beschränkt auf Aktionen mit einer Wahrscheinlichkeit $P > 0$).

Variierter Parameter: Profile

Die Veränderung des Profils ruft eine andere Aktionswahl hervor. Der Agent wählt Aktionen, welche seinem Profil und damit seiner Persönlichkeit entsprechen. In diesem Testfall erhält der Agent zwei Handlungsmöglichkeiten – „Gehen“ und „Rennen“ –, um den damit verknüpften Bedürfnissen nachzukommen. Je nach Profil des Agenten wird die eine oder andere Aktion bevorzugt. Ist das Profil des Agenten nicht eindeutig positiv oder negativ zu werten (also alle Profile, die nicht denen der Aktion entsprechen), so ist für den Agenten diese Aktion gleich nützlich und wird über einen Zufallswert ausgewählt.

Variierter Parameter: Desire

Durch Veränderung der Desire verändernt sich auch die Priorität und damit die Wahrscheinlichkeit, als zu befriedigendes Ziel, als Intention, ausgewählt zu werden. Am Testfall wird beispielhaft der Verlauf des Bedürfnisses „Durst“ und die damit ver-

bundene Wahl der Intention deutlich gemacht. Der Commitment-Faktor wurde nicht betrachtet, um ein von der Aktion unabhängiges Ergebnis zu erhalten.

Desire [1]	Desire [2]	Desire [3]	Intention: Spiel verfolgen	Intention: Trinken
0.5	0.0	0	100 %	0 %
0.5	0.1	0	95 %	0.04 %
0.5	0.2	0	91 %	0.09 %
0.5	0.3	0	86 %	14 %
0.5	0.4	0	83 %	17 %
0.5	0.5	0	80 %	20 %
0.5	0.6	0	77 %	23 %
0.5	0.7	0	74 %	26 %
0.5	0.8	0	71 %	29 %
0.5	0.9	0	69 %	31 %

Tabelle 4: Verhaltensanalyse: Parameter Desire

Variierter Parameter: Commitment

Der Commitment-Parameter beeinflusst die Beibehaltung eines gewählten Ziels und die Wahrscheinlichkeit eines Wechsels auf ein lohnenderes Ziel. Die Tabelle zeigt, dass mit steigendem Commitment auch die Differenz der Priorität zwischen zwei Aktionen für einen möglichen Wechsel steigen muss.

Desire [1]	Desire [2]	Desire [3]	Commitment	Intention: Spiel verfolgen	Intention: Trinken
0.5	0.9	0	0.0	69 %	31 %
0.5	0.9	0	0.1	70 %	30 %
0.5	0.9	0	0.2	72 %	28 %
0.5	0.9	0	0.3	73 %	27 %
0.5	0.9	0	0.4	75 %	25 %
0.5	0.9	0	0.5	76 %	24 %
0.5	0.9	0	0.6	77 %	23 %
0.5	0.9	0	0.7	78 %	22 %
0.5	0.9	0	0.8	79 %	21 %
0.5	0.9	0	0.9	80 %	20 %

Tabelle 5: Verhaltensanalyse: Parameter Commitment

Wie in Tabelle 4 dargestellt, liegt die Verteilung der Intentionen „Spiel verfolgen“ und „Trinken“ mit den Werten Desire[1] = 0.5 und Desire[2] = 0.9 bei 69% zu 31%. Nun wird der Commitment-Parameter variiert, um das „Festhalten“ des Agenten an einer Aktion zu simulieren. Tabelle 5 zeigt den Einfluss des Commitment auf die Verteilung. Nach Ablauf der Aktionsdauer (wenn die Aktion abgeschlossen ist) reduziert sich dieser Bonus wieder auf 0 und die Wahl der Intention ist wieder unbeeinflusst von der vorangegangenen Aktion.

7.2 Benchmark

Die Anforderung an eine ausreichende Skalierbarkeit ist erfüllt, aber bietet Raum für Optimierung. Zwar bietet das System die Möglichkeit, eine beliebige Anzahl von Agenten zu berechnen, jedoch erwies sich die Kombination der nativen Scriptsprache MEL und einer Server-Anfrage über TCP/IP als instabil. Häufiger und schneller Aufruf von Befehlen ließ das System abstürzen. Eine Möglichkeit wäre, eine Delay zu verwenden, welche die Aufrufe künstlich verzögert und somit Maya genügend Zeit lässt, das Script korrekt abzuwickeln.

Des Weiteren wird die Nachbarschaftssuche ohne Hilfsstruktur durchgeführt, was in einer Laufzeit von $O(n^2)$ resultiert. Die Verwendung eines kd-Baums oder einer anderen räumlichen Datenstruktur zur Beschleunigung der Nachbarschaftssuche würde einen weiteren Geschwindigkeitsvorteil erzielen und $O(\log n)$ garantieren. Da alle Elemente zu Beginn feststehen, ist die Aufbauzeit zu vernachlässigen, jedoch könnten sich die häufigen Änderungen in der räumlichen Position der Agenten negativ auf die Balancierung des Baums auswirken. Um die Laufzeit auch ohne Hilfsstruktur zu begrenzen, kann ein Maximum für die Anzahl der Nachbarn definiert werden.

Im Folgenden werden die Laufzeiten der Berechnungen für einzelne Agenten und die vollständige Simulation untersucht[21]. Zu diesem Zweck wurde in der *updatepn-Funktion* (siehe Kapitel 6.1) der Zeitraum zwischen Sammeln und Senden der Daten und Empfang einer Antwort in Millisekunden gemessen. Abbildung 38 zeigt die Dauer der Erfassung der Nachbarn eines Agenten. Wie man sieht, steigt die Zeit auf dem Testsystem linear um ca. 1 Millisekunde pro Nachbar.

[21] Testsystem: Intel Core2Duo 6300 @ 1.86GHz, 2GB DDR2 RAM

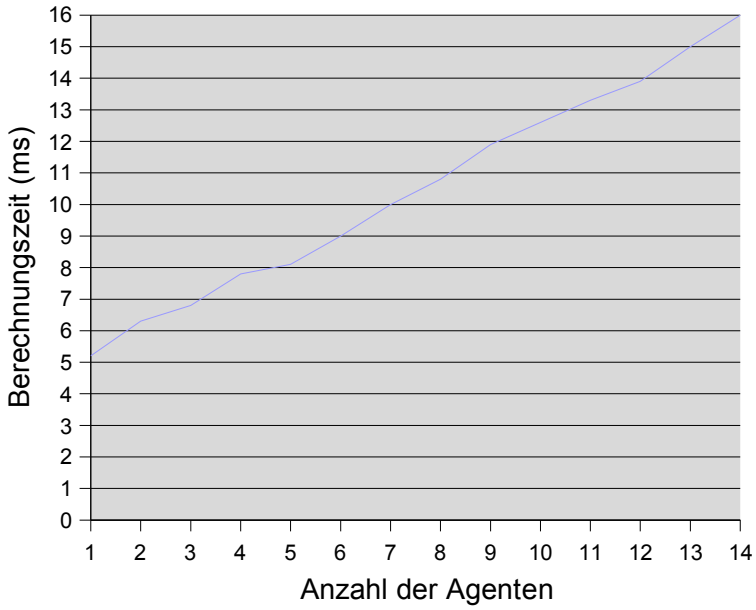


Abbildung 38: Berechnungszeit für einen Agenten

In der Regel erfasst aber jeder Agenten alle seine Nachbarn. Bei n Agenten ergeben sich damit $n \cdot (n-1)$ Abfragen. Abbildung 39 zeigt die Dauer der Nachbarschaftssuche (alle Objekte in meinem Sichtfeld) aller Agenten.

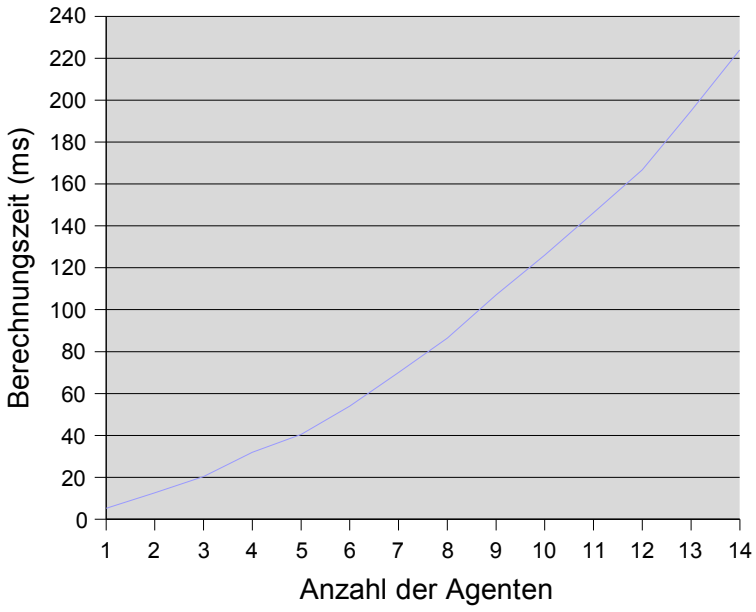


Abbildung 39: Nachbarschaftssuche für alle Agenten

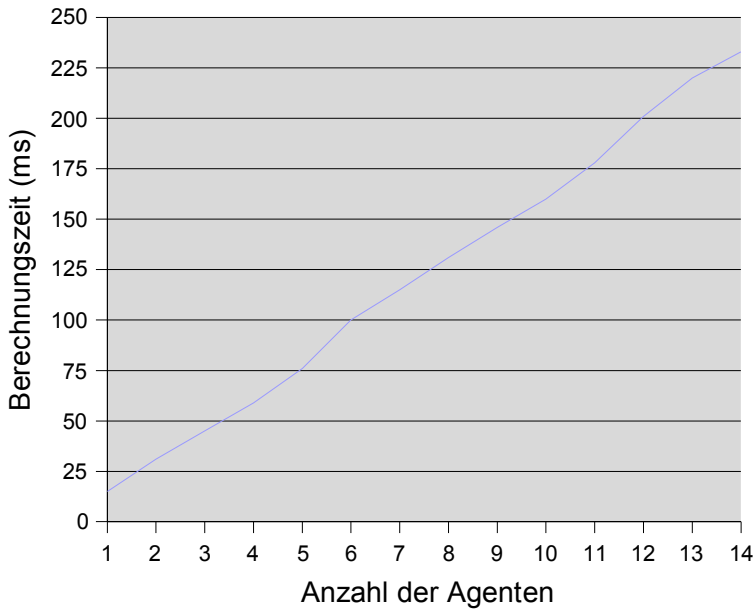


Abbildung 40: Übertragungszeit

Der Prozess der reinen Aktionswahl ist in O(n) für jeden Agenten berechenbar und nur abhängig von der Anzahl der Nachbarn. Er ist unterteilt in die Akquirierung der Daten aller gefundenen Nachbarn, der Überprüfung auf vorhandene Regeln, dem Update der Bedürfnisse, der Berechnung der Priorität jedes Bedürfnisses und der Aktionswahl.

Ein großer Teil der tatsächlichen Laufzeit der Anwendung wird bei der Kommunikation zwischen Client und Server verbraucht (im Testfall: ca. 15 ms/Agent) und ist konstant. Die übertragene Datenmenge steigt zwar mit steigender Agentenzahl, ist jedoch zu vernachlässigen. Auch hier lässt sich durch eine Schranke ein Maximum der Anzahl von Nachbarn einführen.

.

8 Zusammenfassung und Ausblick

Im üblichen Kontext werden MAS zur Beschreibung verteilter KI verwendet, bei der eine Gruppe von Softwareagenten an der Lösung eines gemeinsamen Entscheidungsproblems beteiligt sind, wie beispielsweise in **[Cam00]** als Kooperationsstrategie beschrieben. In dieser Arbeit wird das MAS-Konzept anders formuliert. Die Agenten arbeiten nicht lokal an der Lösung einer globalen Aufgabe, sondern an einer autonom bestimmten lokalen Aufgabe. Durch das intuitiv beschreibbare und leicht anzupassende System können natürliche System zur Simulation von Gruppen von Individuen untersucht werden.

Solche MAS finden, wie einleitend erwähnt, häufig Anwendung in der Architektur (Evakuationsszenarien und Personenfluss **[Pel05], [Mur03]**) und/oder sozialen Studien. Neueste Forschungsberichte legen dar, dass die Einbeziehung von psychologischen und emotionalen Faktoren in die Simulation eine erhöhte Kongruenz mit den experimentell erhaltenen Daten aufweist. Der Schluss liegt nahe, dass sich dies auch in der Simulation für virtuelle Charaktere als hilfreich erweist.

Diese Arbeit zeigt, dass sich BDI-Modelle zur Steuerung von virtuellen Charakteren verwenden lassen. Um von autonomen Agenten zu sprechen, ist hervorzuheben, dass, anders als in bisherigen Implementierungen und Modellen, der Agent modifiziert wird, um das Verhalten anzupassen und keine direkte Modifikation des Verhaltens durchgeführt wird. Die Variation der Parameter für Bedürfnisse reicht aus, um einen gewünschten Effekt zu erzielen, und ist vom Benutzer intuitiv festlegbar. Insbesondere im Hinblick auf mögliche Erweiterungen und Anpassungen des Systems für andere Einsatzzwecke bietet das System ausreichend Schnittstellen.

Der Entwurf der Architektur als Client-Server-System zeigt sich als optimal und wird aufgrund der beschränkten Möglichkeiten der API auch für andere Projekte, welche beliebige Objekte in der Software Maya steuern, verwendet werden. Denkbar ist dies vor allem im Bereich von Partikelsystemen, welche durch Festlegen eines Regelsystems ein beliebiges physikalischen Modell simulieren können, oder bei bestehenden Softwaremodulen, welche mit Maya gekoppelt werden sollen, aber sich nicht als ein Maya-Objekt (siehe Kapitel 4.3.1 - DG Node) implementieren lassen. Durch eine Anpassung der Schnittstelle lässt sich solch eine Integration erwirken.

Während der Entwicklung der Architektur zeigt sich, dass es einiger weiterer Kernkomponenten bedarf, um eine vollständige Nutzbarkeit zu gewährleisten. Die notwendigsten Module wären ein Wegfindesystem, ein Planer und ein Animationsmodul.

Wegfindung ist eins der elementarsten Gebiete der KI zur Orientierung und Bewegung in einem Raum und damit eine bestimmende Komponente eines kompletten KI-Systems. Die Arbeit von Daniel Kastenholz bietet dazu einige interessante Ansätze, wie effiziente Wegfindung, insbesondere in einem dreidimensionalen Raum, realisierbar wäre **[Kas06]**. Die Integration in Maya stellt aber ein noch zu lösendes Problem dar, denn die Wahrnehmung des Agenten basiert in der jetzigen Implementierung auf Eigenschaften und Objekten, welche vom Agenten erkannt werden. Des Weiteren wäre das dynamische Umfeld von Maya, in dem die Szene ihre Konfiguration jederzeit ändern kann, ein zu betrachtender Aspekt und ein Teil weiterer Untersuchungen.

Die prozedurale Generierung von Animationen ist die dritte notwendige Erweiterung. Die Intentionen können zwar in vorhandene Animationen überführt werden (indiziert durch einen Index für die Animation und einen Index für die Phase der Animation), jedoch wäre ein modulares System für die komponentenweise Zusammensetzung der Animation zweckdienlicher **[Bie04]**.

Die Rolle der Emotion bei Entscheidungen wurde bisher nicht betrachtet. Zwar lassen sich Gefühle in Form von Bedürfnissen oder Regeln beschreiben, jedoch ist der Zusammenhang zur Entscheidungsfindung nicht gegeben. Der Einfluss einer Emotion auf die Geschwindigkeit und Aktionswahl selbst muss untersucht und ggf. als zusätzlicher Filter vor die Bedürfnisse geschaltet werden.

All diese Erweiterungen lassen sich über Aktionsmodule implementieren und sind parallel zur jetzigen Implementierung lauffähig. Der Agent erhält durch eine Intention die Aufgabe, einen Plan zu entwickeln oder einen Weg zu finden. Lediglich die Kombination von Animationsfraktalen zur Animationsgewinnung müsste in einer separaten Software eingebunden werden, da hierfür keine Schnittstelle besteht. Das KI-System ist, wie in Kapitel 5 ausgeführt, von der ausführenden Seite abgeschlossen.

Die Praxiserprobung des Systems durch die Verwendung in einem kommerziellen Projekt steht noch aus. Durch mangelnde Nutzerzahl und fehlende Kennzahlen für die derzeitige Arbeitsbelastung in Fällen, bei denen das System eine Minimierung des Arbeitsaufwandes bewirken könnte, lässt sich noch keine Aussage über die kommerzielle Nutzbarkeit machen. Hierzu müssten auch die fehlenden Elemente des Frameworks ergänzt werden, was aber im Rahmen weiterer Arbeiten vorgesehen ist.

Das vorliegende System lässt sich auf eine Vielzahl von Anwendungszwecken hin anpassen. Daher wurden in diesem speziellen Szenario einige Konzepte außer Acht gelassen, insbesondere Lernverfahren und Planer. Beide Module würden den Agenten zu viele Freiheiten einräumen. Das Resultat wäre ein nicht vorhersagbares Verhalten, was, wie in Kapitel 2 ausgeführt, zum Anforderungsprofil gehört.

Sollte kein bestimmtes Verhalten erwünscht sein, so bietet eine reinforcement-learning-Strategie eine Möglichkeit, die Utility zwischen einem Bedürfnis und einer Aktion zu bestimmen. Durch eine anfangs zufällige Aktionswahl würde das Verhalten des Agenten zur optimalen Strategie konvergieren. Der Agent bedarf einer Messmethode, um eine Bewertung für die folgende Situation zu erhalten. Dies kann entweder a priori über Nachbedingungen für jede Aktion oder a posteriori über die Betrachtung des gefolgerten Zustands der Welt erfolgen.

Durch den modularen Aufbau ist das System leicht an jede beliebige Visualisierungssoftware anbindbar. Zur Vereinfachung des Datenaustauschs wäre für zukünftige Entwicklungen ein Austauschformat in XML-Dateien vorteilhaft, was aber zur Datenminimierung, schnelleren Aufschlüsselung und einer einfachen Integration in Maya in dieser Arbeit nicht implementiert wurde.

Vorstellbar wäre insbesondere die Verwendung in Computerspielen, denn in der Regel sind alle Charaktere einer solchen interaktiven Anwendung nichts anderes als autonome Agenten in einer virtuellen Welt. Häufig werden Scripts eingesetzt, um diese Agenten zu steuern, dadurch entsteht ein oft vorhersehbares und musterbehaftetes Verhalten, welches eine Verschlechterung des Spielerlebnisses verursacht. Um dem zu entgehen, könnte der künstliche Gegenspieler durch ein BDI-System modelliert werden und sich tatsächlich situationsabhängig verhalten. Ein weiterer Vorteil wäre, dass dieses einmalig formulierte Expertensystem prinzipiell für alle ähnlich aufgebauten Szenarien ohne Anpassung verwendet werden kann. Ein offensichtlicher Nachteil der Verwendung eines solchen Systems ist die Schwierigkeit einer Erzählstruktur,

welche auf vorab definierten Verhaltensweisen, Geschehnissen und Situationen basiert. Es ist schwierig, eine Geschichte zu erzählen, wenn die Charaktere Autonomie besitzen.

Ein weiterer Einsatzzweck wäre eine Entwicklung einer menschenähnlichen Intelligenz, nicht, was kognitive Fähigkeiten angeht, aber in der Nachahmung der menschlichen Entscheidungsfindung. Ein naiver Ansatz wäre die Steuerung eines elektronischen Haustiers. Der Vorreiter auf diesem Gebiet war Sony mit der Entwicklung des elektronischen Haustiers „Aibo“[22], welches, trotz seines futuristischen Äußeren, durch sein Verhalten ein lebendes Wesen nachahmen sollte. Auch hier wäre der Einsatz eines BDI-Modells zur Verhaltenssteuerung denkbar. Insbesondere dieser Einsatzzweck benötigt weitergehende Entwicklung in der Objekt- und Eigenschaften-Identifikation als auch Wegfindung.

[22] Sony AIBO Europe - http://support.sony-europe.com/aibo

Anhang A: Handhabung

Um ein Objekt an der Simulation teilhaben zu lassen, muss dem Objekt zuerst ein Agentenprofil zugewiesen werden. Dazu muss die „Transform Node“ des 3D-Objektes mit einer „Profile Node“ verknüpft werden. Das folgende MEL-Script erzeugt für ein selektiertes Objekt eine „Profile Node“ und erzeugt die notwendigen Verknüpfungen.

```
string $nodes[];
$nodes = `selectedNodes`;
string $pnode;
$pnode = `createNode "profile"`;

$pnodestring = $pnode+".Translation";
$objectstring = $nodes[0]+".translate";
connectAttr $objectstring $pnodestring;

$pnodestring = $pnode+".Rotation";
$objectstring = $nodes[0]+".rotate";
connectAttr $objectstring $pnodestring;

$pnodestring = $pnode+".Boxmin";
$objectstring = $nodes[0]+".boundingBox.boundingBoxMin";
connectAttr $objectstring  $pnodestring;

$pnodestring = $pnode+".Boxmax";
$objectstring = $nodes[0]+".boundingBox.boundingBoxMax";
connectAttr $objectstring  $pnodestring;

$pnodestring = $pnode+".ShapeConnector";
$objectstring = $nodes[0]+".visibility ";
connectAttr $objectstring  $pnodestring;
```

A.1 MEL Kommandos

updatepn

Durch den Aufruf des *updatepn*-Kommandos wird für alle Objekte mit einer „Profile Node“ ein Update ausgeführt. Das Update umfasst eine Sammlung der Sensordaten (Liste von wahrgenommenen Agenten und ihre Distanzen) und das Senden eines Requests an den Server. Das Kommando geht in den sogenannten „Blocking State“ und wartet auf eine Antwort. Diese wird in der „Profile Node“ in den Attributen „Turning“ und „Velocity“ gesichert.

updateg

Durch den Aufruf des *updateg*-Kommandos wird für alle Objekte mit einer „Profile Node“ ein Update der „Transform Node“ durchgeführt. Das Kommando transliert und rotiert das mit der aktuell betrachteten „Profile Node“ verbundene 3D-Objekt um die in den Attributen „Velocity“ und „Turning“ gesetzten Werte.

updatepn und *updateg* bilden somit ein Kommandopaar, welches einen Simulationsschritt berechnet.

A.2 Parameter

Im folgenden werden alle Parameter eines Agenten aufgelistet. Sie sind über die „Profile Node“ in Maya modifizierbar.

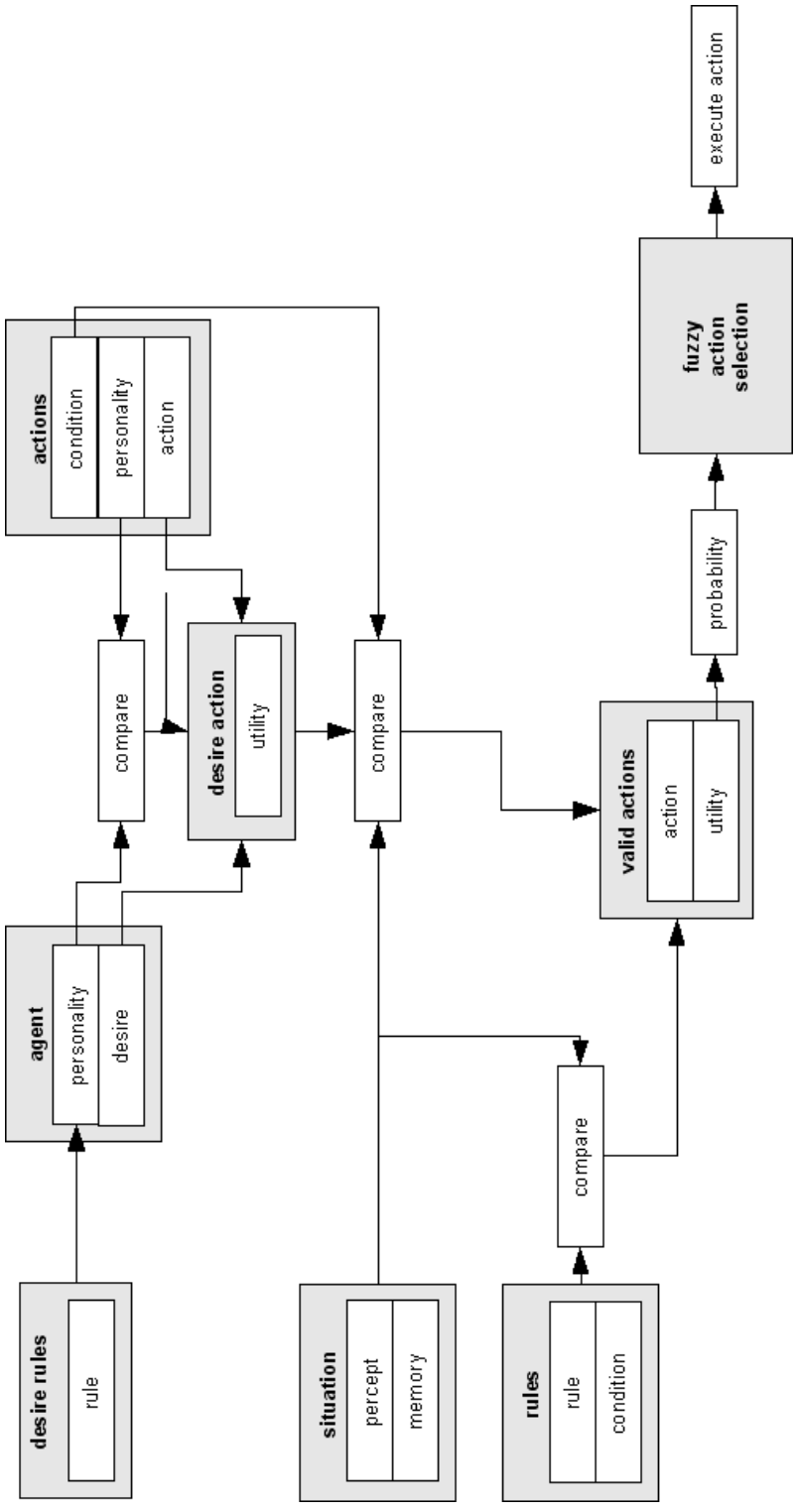
Radius Sight	Sichtweite des Agenten. Bestimmt den Radius in dem der Agent andere Agenten wahrnimmt.
FieldOfView	Sichtbereich des Agenten.
Desire [5]	Die statischen Bedürfnisse eines Agenten. Die Bedürfnisse wer den in Form von Ganzzahlen kodiert. Gegenwärtig werden sie in einem Array mit fünf Zellen gesichert.
DesireValue [5]	Die Priorität der statischen Bedürfnisse. Der Index dieses Arrays ist mit dem des Arrays Desire verbunden, so dass das Bedürfnis Desire[i] der Priorität DesireValue[i] zugeordnet wird.

Profile [4]	Das Persönlichkeitsprofil des Agenten. Bestimmt durch die vier Dimensionen des MBTI Modells.
Di [5]	Der Identifikator der dynamischen Bedürfnisse.
Ft [5]	Die dem dynamischen Bedürfnis zugeordnete Funktion zur Berechnung der Priorität (durch eine Ganzzahl kodiert).
Av [5]	Ist-Wert (actual value) des dynamischen Bedürfnisses.
Gv [5]	Soll-Wert (given value) des dynamischen Bedürfnisses.

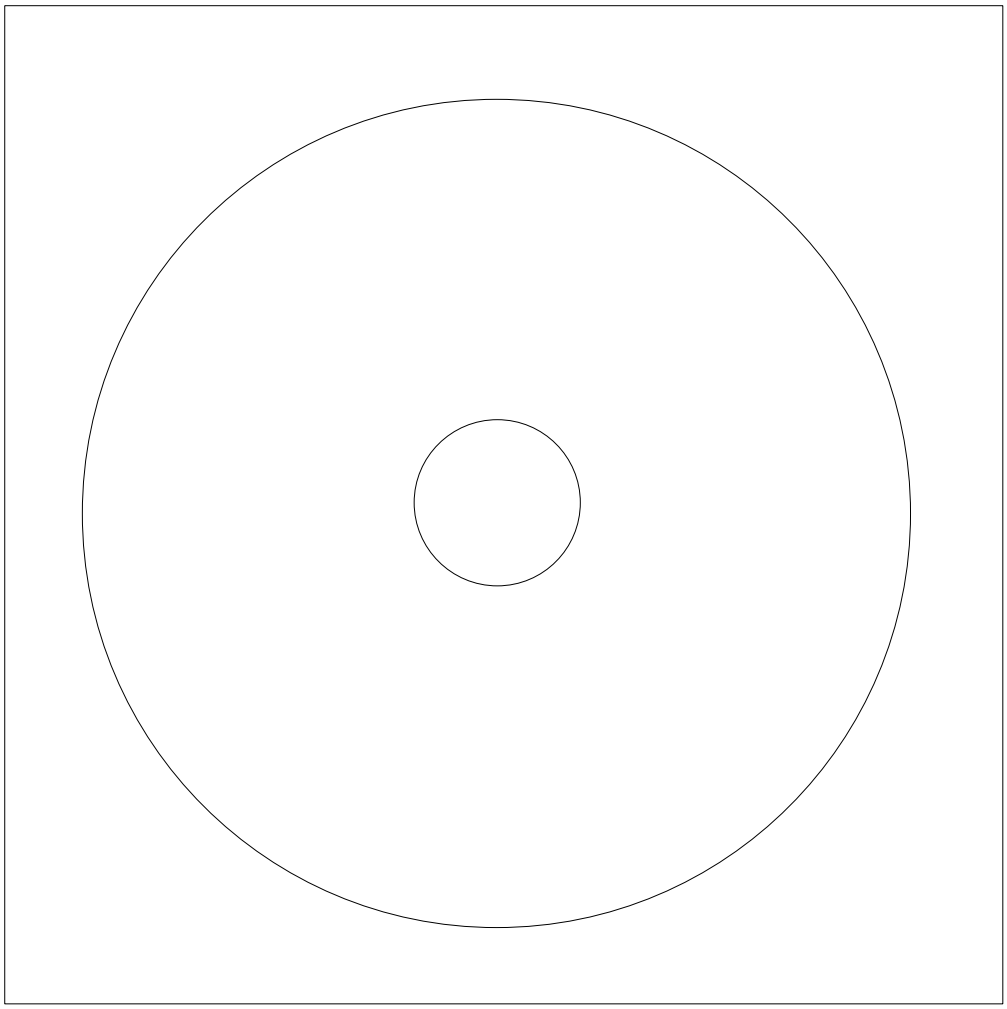
Zusätzlich beinhaltet die „Profile Node“ noch folgende read-only Werte zur Kontrolle.

Server Return	0 = keine Kommunikation mit dem Server bei letztem Request 1 = Request erfolgreich
Translation XYZ	Beinhaltet die von der „Transform Node“ erzeugten Positionswerte
Rotation XYZ	Beinhaltet die von der „Transform Node“ erzeugten Rotationswerte
Heading XYZ	Die Berechnete Blickrichtung des Agenten
Turning XYZ	Die zukünftige Rotation des Agenten
Velocity XYZ	Die zukünftige Translation des Agenten

Anhang B: Architektur



Anhang C: Quellcode



Daten CD (Quellcode)

Literaturverzeichnis

[**Aut08**]: Autodesk, *Maya User Documentation*, Autodesk, San Rafael, USA, 2008

[**Bie04**]: Bielik, Alain, *Troy: Innovative Effects on an Epic Scale*, 2004, http://vfx-world.com/?atype=articles&id=2105&page=1

[**Bir01**]: Birn, J., *Lighting & Rendering*, Markt+Technik, München, Deutschland, 2001

[**Cam00**]: Camponogara, E.; Talukdar, S.N. , *A Cooperation Strategy for Decision-making Agents*, , Carnegie Mellon University, 2000

[**CG95**]: Foley, J.D.; VanDam, A.; Feiner, S.K., *Computer Graphics: Principles and Practice in C*, Addison-Wesley Longman, Amsterdam, 1995

[**Dur07**]: Durupmar, F., *Behavioral Animation for Crowd Simulation*, Bilkent University, 2007

[**Fun06**]: Funke, J; Frensch, P., *Handbuch der Allgemeinen Psychologie: Kognition*, Hogrefe, Göttingen, Deutschland, 2006

[**Geo99**]: Georgeff, M.; Pell, B.; Pollack, M.; Tambe, M.; Wooldridge, M., *The Belief-Desire-Intention Model of Agency*, , Australian AI Institute, 1999

[**Gou03**]: Gould, D.A, *Complete Maya Programming*, Morgan Kaufmann Publishers, San Francisco, USA, 2003

[**Ham93**]: Hammer, A.L., *Intoduction to Type and Careers*, Consulting Psychologists Press, Palo Alto, USA, 1993

[**Jun21**]: Jung, C.G., *Psychologische Typen*, , , 1921

[**Kas06**]: Kastenholz, Daniel, *3D Pathfinding*, , Technische Universität München, 2006

[**Kol00**]: Kolve, C., *Simulation von Gruppenbewegungen durch Verhaltensgesteuerte Animation*, Fachhochschule Dortmund, 2000

[**Lad08**]: Ladebeck, M., *Applying Dynamic Scripting to JaggedAlliance 2*, Technische Universität Darmstadt, 2008

[**Lim04**]: Limsäter, F., *REACT Crowd Simulation System for Visual Effects*, Linköpings Universitet, 2004

[**Mur03**]: Murakami, Y; Minami, K.; Kawasoe, T.; Ishida, T., *Multi-Agent Simulation for Crisis Management*, , Kyoto University, 2003

[**OGL07**]: Schreiner, D.; Woo, M.; Neider, J.; Davis, T., *OpenGL Programming Guide*, Addison-Wesley Longman, Amsterdam, 2007

[**Pat05**]: Patterson, J.A., *Implementing Autonomous Crowds in a Computer Generated Feature Film*, Texas A&M University, 2005

[**Pel05**]: Pelechano,N.; O'Brien, K.; Silverman, B.; Badler, N., *Crowd Simulation Incorporating Agent PsychologicalModels, Roles and Communication*, , University of Pennsylvania, 2005

[**Rao96**]: Rao, A.S., *AgentSpeak(L): BDI Agents speak out in a logicalcomputable language*, Australian Artificial Intelligence Institute, , 1996

[**Rus95**]: Russel, S.J.; Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall Inc., New Jersey, USA, 1995

[**Spr05**]: Spronck, P., *Adaptive Game AI*, , 2005

[**Tor03**]: Torres, J.A.; Nedel, L.P.; Bordini, R.H., *Using the BDI Architecture to Produce Autonomous Characters in Virtual Worlds*, , Federal University of Rio Grande do Sul; University of Liverpool, 2003

[**Wat19**]: Watson, J.B., *Psychology from the standpoint of a behaviorist*, Kessinger Pub Co, , 1919