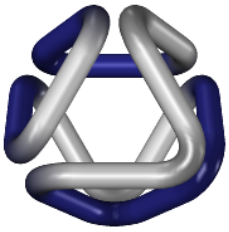

Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme

Kognitionsframeworks II

Ausarbeitung

Fachstudie (SS11)

Betreuer: Michael Raschke, Prof. Thomas Ertl

Stephan Engelhardt, Paul Hummel, Oliver Schmidtmer

Stuttgart, 1. August 2011

Inhaltsverzeichnis

Kurzbeschreibung

Diese Fachstudie beschäftigt sich mit der Untersuchung und dem Vergleich der kognitiven Frameworks ACT-R, Apex und Soar. Dazu wird eine Einführung in die Grundlagen der Kognitionswissenschaft und der KI gegeben. Sowohl der Gehirnaufbau als auch verschiedene Modelle der Wissensrepräsentation und der Informationsverarbeitung werden aufgezeigt. Im Anschluss legen die Autoren Architektur, Einsatzgebiete, Ergebnisvisualisierung und Grenzen der untersuchten Frameworks dar. Abschließend werden die Frameworks bewertet und ein Ausblick in die weitere Entwicklung gegeben.

Abstract

This paper presents the research and comparison of the cognitive architectures ACT-R, Apex and Soar. This includes an introduction in the basics of cognitive science and AI. Authors show the structure of brain as well as several models of knowledge representation and information processing. Afterwards the authors point out the structure, applications, result visualization and the limits of cognitive architectures. Finally this paper evaluates the frameworks and gives an outlook for the development in the future.

1 Einleitung

Bereits die alten Griechen hatten Visionen von Maschinen, welche automatisch mathematische Berechnungen durchführen können. Damals war es das Ziel, eine Maschine zu entwickeln, welche einfache arithmetische Rechnungen löst. Bis heute ist unter den meisten Menschen die Meinung weit verbreitet, dass Personen, die schnell komplizierte Rechnungen im Kopf lösen können, sehr intelligent sind. Ironischerweise ist jedoch genau das die Stärke von heutigen Computern und kaum einer würde diesen deshalb als besonders intelligent bezeichnen.

Nach und nach setzten die Menschen die Ziele, was ein Computer leisten soll, immer höher. Man glaubte, ein Programm, welches einen Schachspieler simuliert, wäre eine intelligente Leistung. Auch dies ist heutzutage kein Problem mehr und gilt deshalb in den Augen vieler Menschen nicht mehr als intelligent.

Wird es jemals ein intelligentes Programm geben, wenn Menschen die Anforderungen an einen Computer immer höher stecken und bisher Erreichtes nicht als intelligent bezeichnen? Damit sich ein Programm menschenähnlich verhält, muss es möglicherweise ähnlich einem Gehirn aufgebaut sein. Nicht nur das Resultat, sondern auch der Lösungsweg müsste wahrscheinlich dem des menschlichen Denkprozesses ähneln. Um also typisch menschliches Verhalten künstlich zu erzeugen, genügen keine binären Entscheidungen - der Weg muss Emotionen und Unschärfen enthalten.

Um den Zielen näher zu kommen, arbeiten Informatiker seit einiger Zeit mit Kognitionswissenschaftlern zusammen und versuchen herauszufinden, wie das menschliche Gehirn arbeitet, um intelligente Programme für verschiedene Einsatzgebiete zu entwerfen.

Diese Fachstudie gibt einen Einblick in die Funktionsweise des Gehirns und beschäftigt sich mit der Frage, was Kognition bedeutet. Mit der Vorstellung einiger Modelle, wird erklärt, wie Wissen im Gehirn abgespeichert und dadurch Entscheidungen getroffen werden. Im Weiteren werden Unterschiede zwischen der klassischen künstlichen und der kognitiven künstlichen Intelligenz erläutert und Grenzen der Simulation solcher Algorithmen aufgezeigt.

Im zweiten Teil des Dokuments gibt es einen Einblick in verschiedene Kognitionsframeworks, welche einige Probleme der kognitiven künstlichen Intelligenz behandeln. Dazu werden sowohl der Aufbau der verschiedenen Architekturen, als auch die Einsatzgebiete der einzelnen Frameworks genauer untersucht und zuletzt miteinander verglichen.

2 Kognition

Wie denkt ein Mensch? Psychologen, Neurologen und Informationswissenschaftler gehen dieser Frage nach und arbeiten gemeinsam, um neue Erkenntnisse über das komplexe System des Denkens zu gewinnen. Man nennt diese interdisziplinäre Wissenschaft „Kognitionswissenschaft“. Sie beschäftigt sich damit, welche neuronalen Vorgänge zwischen der Wahrnehmung von Reizen und den Reaktionen auf diese stattfinden. Diese Vorgänge nennt man „kognitiv“ und „Kognition“ ist der ganze Prozess, der diese Vorgänge beschreibt.

Eine weitere spannende Frage, die sich stellt, ist: Wie weit ist es möglich, die kognitiven Fähigkeiten eines Menschen durch einen Rechner zu simulieren? Auch die Autoren dieser Ausarbeitung setzten sich damit auseinander. Doch bevor man sich damit beschäftigen kann, muss man die grundlegenden Prozesse im Gehirn kennenlernen. Im folgenden Unterkapitel wird der grundlegende Aufbau des menschlichen Gehirns auf Makro- und Mikroebene vorgestellt. Danach beschreibt die Ausarbeitung, wie das Wissen im Gehirn repräsentiert werden kann. Des Weiteren wird auf die Möglichkeiten der Informationsverarbeitung, wie sie im Gehirn stattfinden könnte, eingegangen. Und anschließend werden Grenzen der Kognitionssimulation mit Rechnern aufgezeigt.

2.1 Das Gehirn

Es steht außer Frage, dass das Hauptdenkorgan des Menschen das Gehirn ist. Um seine genaue Funktion besser zu verstehen, muss man sich zunächst mit seinem Aufbau auseinandersetzen. Dieses Unterkapitel stellt einige Gehirnbereiche vor.

Das menschliche Gehirn kann man grob in zwei Teile einteilen: die linke Hirnhälfte und die rechte (Abbildung 2). Sie werden Hemisphären, griechisch für „Halbkugel“, genannt. Die Wahrnehmungen werden, je nach Seite woher der Reiz kam, von der gegenüberliegenden Hirnhälfte verarbeitet. Zum Beispiel werden Signale vom linken Auge an die rechte Hirnhälfte und vom rechten Auge an die linke geleitet (Abbildung 3).

Die zwei Gehirnhälften sind nicht komplett voneinander getrennt, denn es existiert eine Verbindung, die durch den **Balken** (*Corpus callosum*, Abbildung 1) ermöglicht wird. Somit ist die Kommunikation in der Wahrnehmung und der Verarbeitung der Informationen zwischen den Hemisphären sichergestellt.

Auf der Oberfläche der Hirnhälften befindet sich die **Großhirnrinde** (*Zerebraler Kortex*, Abbildung 1, auch Neocortex genannt). Eine hochentwickelte Großhirnrinde ist ein Merkmal der Säugetiere. Sie ist für „intelligente“[1] Handlungsweisen der Menschen zuständig. Dort findet das bewusste Denken statt [39]. Man darf nicht außer Acht lassen, dass einzelne Gehirnteile ohne die anderen nicht funktionieren können, da es immer wichtige Verbindungen zu anderen Teilen des Gehirns gibt. Das heißt, es muss ein Gehirnteil, das für Motorik (z. B. Bogenschießen) zuständig ist, mit einem anderen Gehirnteil, das für die menschlichen Triebe (z. B. Verlangen nach Essen) verantwortlich ist, kommunizieren, um den Menschen dazu zu bringen, jagen zu gehen. Beim Jagen muss er seine Umwelt

[1] Es gibt keine eindeutige und abgegrenzte Beschreibung, was Intelligenz ist und welche Handlungsweisen als intelligent zu bezeichnen sind.

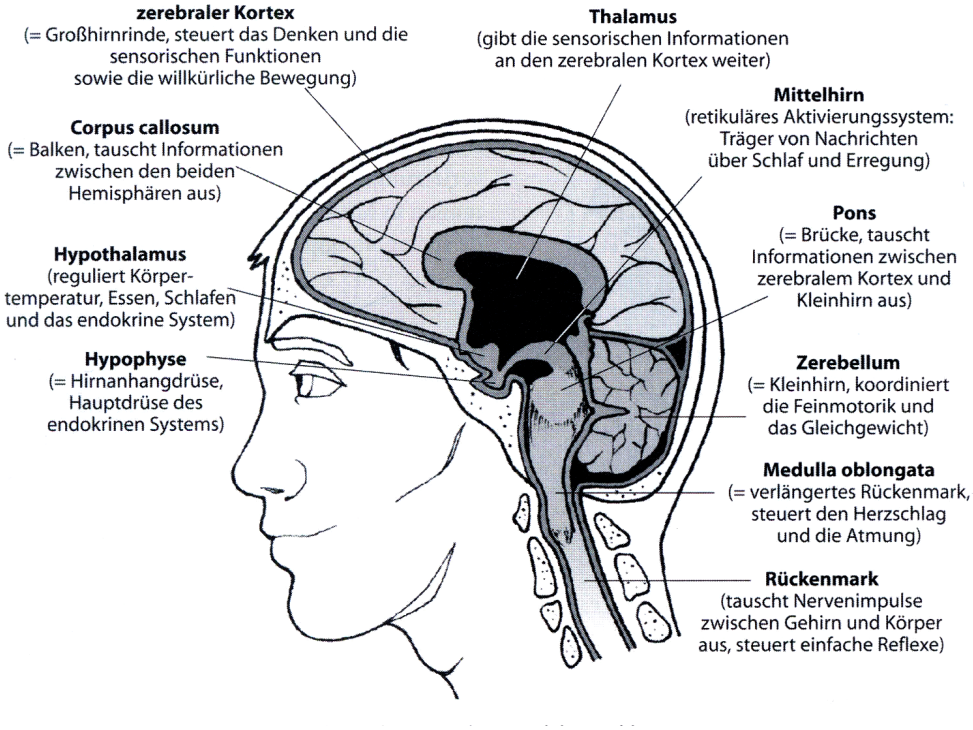

Abbildung 1: Aufbau des menschlichen Gehirns. Man erkennt die grobe Struktur mit verschiedenen Gehirnregionen und ihren Zuständigkeiten [39]

Abbildung 2: Ein 3D-Modell eines menschlichen Gehirns, an dem die Gehirnhälften deutlich sichtbar sind [34]

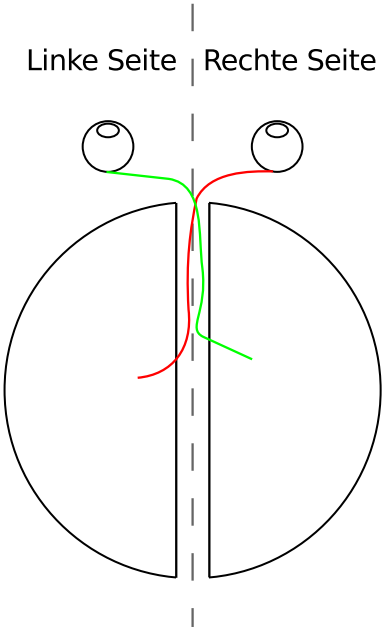

Abbildung 3: Signale werden immer von der gegenüberliegenden Gehirnhälfte verarbeitet. Hier am Beispiel von visuellen Reizen dargestellt.

wahrnehmen, weshalb der Perzeptions-Gehirnteil mitwirken muss. So ergibt sich, dass bei jedem Verarbeitungsvorgang im Gehirn immer mehrere Bereiche beteiligt sind.

Das oben erwähnte Gehirnteil für Motorik ist das **Kleinhirn** (*Zerebellum*, Abbildung 1). Der Begriff „Motorik“ umfasst Bewegungsausführungen, Koordination dieser und das Halten des Gleichgewichts [47].

Im **Thalamus** (siehe Abbildung 1) werden alle sensorischen Signale gesammelt, um von anderen Gehirnteilen weiterverarbeitet zu werden. Dadurch, dass er direkten Einfluss auf den Hormonhaushalt des Menschen nimmt, ist er ein wichtiger Vermittler zwischen dem vegetativen Nervensystem[2] und dem Hormonsystem. Seine weiteren Aufgaben sind Schlaf- und Körpertemperatur-Steuerung, sowie Schmerzempfindung. [47]

Das **verlängerte Rückenmark** (*Medulla oblongata*, Abbildung 1) ist für automatisch ablaufende Vorgänge verantwortlich. Dazu zählt unter anderem: [47]

- Stoffwechselsteuerung
- Herzschlagsteuerung
- Atmungssteuerung
- Schluckreflex

[2]Das vegetative Nervensystem ist ein Teil des gesamten Nervensystems, welches für die Aufrechterhaltung der lebenswichtigen Vorgänge, wie zum Beispiel des Herzschlags, zuständig ist.

- Hustreflex
- Lidschlussreflex

Der Hirnstamm, bestehend aus **Medulla oblongata** (verlängertes Rückenmark), **Pons** (Brücke) und **Mittelhirn** [18] (Abbildung 1), ist der grundlegende Teil des menschlichen Gehirns und wird umgangssprachlich Reptilienhirn genannt, weil es, aus der Sicht der Evolutionstheorie, bereits bei Reptilien vorzufinden ist und im Laufe der Evolution erhalten blieb. Der Hirnstamm steuert reflexartige und instinktive Verhaltensweisen, und verarbeitet teilweise die dafür notwendigen Sinneseindrücke [47]. Teilweise, weil es keine scharfe Trennung der Zuständigkeitsbereiche gibt. Das heißt Gehirnteile haben Hauptaufgaben, was nicht ausschließt, dass nebenliegende Regionen am Verarbeitungsprozess mitbeteiligt sind.

Die oben genannten Gehirnteile sind bei der Strukturbetrachtung auf der Makroebene des Gehirns zu erkennen. Auf der Mikroebene besteht das Gehirn aus neuronalen Netzen, mit denen sich das nächste Unterkapitel auseinandersetzt.

2.1.1 Künstliche Neuronale Netze

Im Gesamten bilden Gehirnteile ein riesiges neuronales Netz - ein Konstrukt aus Milliarden von Neuronen.

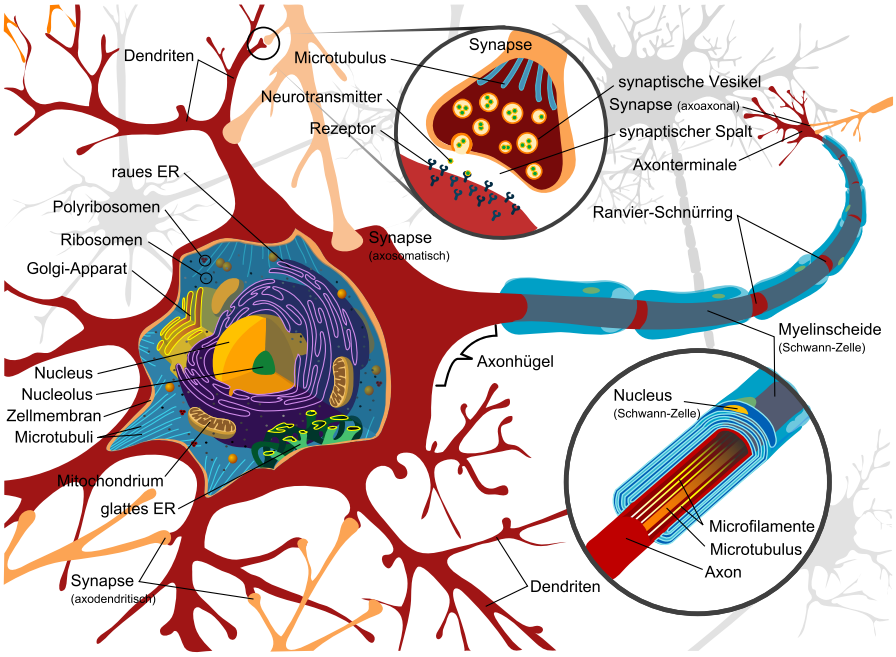

Abbildung 4: Aufbau eines Neurons [19]. Zu näheren Beschreibung siehe Kapitel 2.1.1

Ein Neuron besitzt einen Zellkörper und ist mit bis zu 10.000 anderen Neuronen über Dendriten verbunden. Einige dieser Neuronen sind spezielle Eingangsneuronen, welche

mit Sinnesorganen wie den Augen oder der Haut verbunden sind. Andere Neuronen erhalten von diesen Eingangsneuronen mehrere elektrische Impulse unterschiedlicher Spannungen, welche hauptsächlich jeweils von der Dicke der Dendriten zwischen ihnen abhängen. Gleichzeitig eintreffende Impulse summieren sich in einem Neuron zu einem Gesamtpotenzial auf. Übersteigt dieses einen bestimmten Wert, so „feuert“ das Neuron. „Feuern“ bedeutet, dass das Neuron an all seine Nachbarneuronen mit denen es verbunden ist einen ca. 0,2 ms andauernden elektrischen Impuls von 0,1 V weitergibt [15]. Ob und mit welcher Spannung ein Neuron feuert, ist also völlig unabhängig davon, ob das Eingangspotenzial überstiegen wird - es gilt das Alles-oder-Nichts-Gesetz [37]. Verbindet man drei Neuronen, lassen sich bereits logische UND- und ODER-Gatter simulieren. Koppelt man jedoch viele tausend Neuronen zusammen, können äußerst komplexe, logische Entscheidungen getroffen werden.

Die „Einstellmöglichkeiten“ eines solchen Netzes sind an nur sehr wenigen Stellen möglich. Die größten Faktoren sind die Verbindungen zwischen den einzelnen Neuronen, also welche Neuronen miteinander verbunden sind und wie dick die jeweiligen Dendriten sind. Somit kann man sagen, dass das Wissen in den Verbindungen zwischen den Neuronen steckt.

Inspiriert von dieser Hypothese, wurde der konnektionistische Ansatz entwickelt.

2.1.2 Konnektionistischer Ansatz

Das Gehirn ist bei weitem nicht vollständig erforscht. Verfechter des konnektionistischen Ansatzes, wie zum Beispiel Warren McCulloch und Walter Pitts haben im Jahre 1943 jedoch gezeigt, dass einige Funktionen des Gehirns durch Simulation der neuronalen Aktivitäten nachgebaut werden können [22]. Ein Beispiel dafür ist die optische Mustererkennung.

Diese Forschungen zeigen, dass es eine Möglichkeit gibt, Informationen in Form eines neuronalen Netzes abzuspeichern. Dies bildet eine Grundlage für den konnektionistischen Ansatz, in dem es darum geht, Probleme und Aufgaben mit Hilfe einer computergestützten Simulation des Neuronenverhaltens zu lösen.

Im Gehirn finden unzählige asynchrone Kommunikationen zwischen einzelnen Neuronen statt. Es werden atomare Informationseinheiten in chemischer und elektrischer Form ausgetauscht, die für sich keine Bedeutung haben, jedoch in ihrem Zusammenspiel das Ergebnis für ein Problem liefern können.

Neurone beeinflussen sich anregend oder hemmend [39]. Anregend heißt, dass das ankommende Signal bei einem Neuron verstärkt weitergegeben wird. Hemmend bedeutet, dass das ankommende Signal abgeschwächt durchgeleitet wird oder sogar, dass das Signal das Feuern des Neurons unterbindet.

Auch wenn man meint, dass Menschen regelbasiert handeln (zum Beispiel: „Ich fühle Hunger, also gehe ich zum Kühlschrank“), findet jedes Mal eine komplexe Auswertung der Eindrücke statt, zum Beispiel kann ein Mensch Hunger auf einen Apfel bekommen, der nicht im Kühlschrank liegt. Das Gehirn hält keine Listen, die je nach Reizart und Reizkonfiguration Schritt für Schritt, wie ein Regelbuch, abgearbeitet werden. Das Gehirn ist sehr stark parallelisiert, denn jedes Neuron bildet eine eigene Recheneinheit,

und bei Auswahl der angemessenen Reaktion auf einen Reiz trägt jedes einzelne aktive Neuron zur Entscheidung bei. Es ist vergleichbar mit einer Jazz-Band, bei der jeder einzelne Musiker improvisiert, während viele Instrumente gleichzeitig und nicht nacheinander spielen. Bei jeder der neuen Jamsession mit derselben Hauptmelodie entsteht ein minimal abgewandeltes, neues Musikstück. So ist es auch im Gehirn: Bei ähnlichen Randbedingungen kommen Entscheidungen heraus, die ähnlich, aber nicht zwangsweise gleich sind.

Die neuronalen Netze sind formalisierbar, also können sie im Rechner gespeichert und simuliert werden. Im nächsten Abschnitt wird untersucht, wie Informationen im Gehirn repräsentiert werden können.

2.2 Wissensrepräsentation

Es stellt sich die Frage, wie das Wissen im Gehirn gespeichert wird. Dazu wird die Speicherart des Gehirns zunächst auf Neuronenbasis betrachtet. Davon wird in den nächsten Unterkapiteln abstrahiert und verschiedene Modelle für die Wissensrepräsentation vorgestellt. Zuerst folgt jedoch ein Beispiel, warum diese verschiedenen Ebenen Sinn ergeben.

Die Zahl 3 könnte ebenso als abstraktes mathematisches Modell, als auch als Bild des Zeichens „3“ oder gar als Vorstellung von drei Äpfeln, welche auf einer grünen Wiese liegen, im Gedächtnis verankert sein. Natürlich wäre es auch denkbar, alle drei Repräsentationen abgespeichert zu haben und je nach Kontext die eine oder andere Darstellung zu wählen. Um hervorzuheben, dass es sehr wohl einen Unterschied macht, welche Repräsentation für einen bestimmten Fall gewählt wird, soll hier eine kleine Aufgabe dienen:

> Wir haben ein Quadrat mit der Kantenlänge n. Dieses Quadrat ist in m kleinere Quadrate aufgeteilt, wobei die Anzahl der kleinen Quadrate an einer Kante des großen Quadrats gerade ist. Außerdem bekommen wir beliebig viele Dominosteine, welche jeweils die Fläche von genau zwei kleinen Quadraten abdecken.

Die Aufgabe besteht nun darin, das große Quadrat komplett mit Dominosteinen zu befüllen, sodass keine freie Stelle mehr übrig bleibt. Dies ist eine ziemlich einfache Aufgabe, da man Reihe für Reihe mit Dominosteinen bis zum Ende auffüllen kann (Abbildung 5).

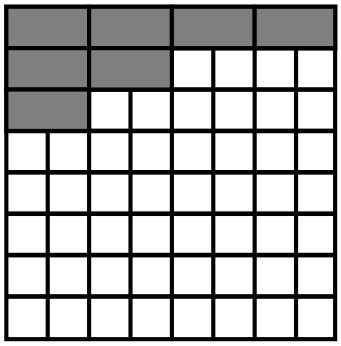

Abbildung 5: Dominosteine lassen sich auf einem quadratischem Gitter mit gerader Zellenanzahl leicht so verteilen, dass das komplette Gitter abgedeckt ist.

Nun wird das große Quadrat etwas abgewandelt: Zwei diagonal gegenüberliegende kleine Quadrate in den Ecken des großen Quadrats werden entfernt. Die Aufgabe bleibt dieselbe. Fänge man jetzt ohne zu überlegen an, die Steine zu verteilen (Abbildung 6), würde man schon bald meinen, dass es nur eine Möglichkeit oder einen speziellen Trick gibt.

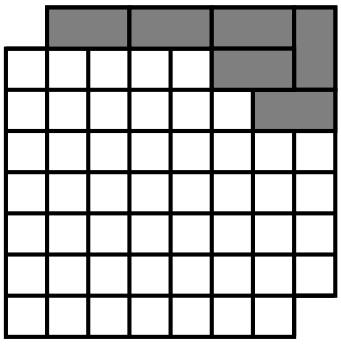

Abbildung 6: Ohne die zwei Diagonalquadrate ist es unmöglich, mit Hilfe der Dominosteine das Gitter komplett abzudecken.

Ersetzt man jedoch die Repräsentation des Quadrats gedanklich durch ein Schachbrett (Abbildung 7), wird der Fall viel leichter lösbar. Die beiden entfernten Quadrate sind beide entweder weiß oder schwarz. Auf dem Schachbrett befinden sich jetzt auf jeden Fall ungleich viele schwarze wie weiße Felder. Ein Dominostein auf einem Schachbrett belegt aber immer genau ein schwarzes und ein weißes Feld. Aus dieser Repräsentation lässt sich nun schließen, dass man die Dominosteine höchstens so weit hinlegen kann, bis letztlich zwei gleichfarbige Felder übrig bleiben. Nur nach sehr langem Herumprobieren wäre irgendwann die Vermutung gekommen, dass das Problem nicht lösbar ist, wobei man erst nach allen Möglichkeiten die sichere Gewissheit darüber gehabt hätte. Wäre die Aufgabe gewesen, den Umfang des Quadrats anzugeben, wäre die Schachbrett-Repräsentation

sicherlich komplizierter gewesen als die mathematische Vorstellung eines Quadrats. Dies verdeutlicht also, dass das Wissen in mehreren Formen (in diesem Beispiel ein einfaches Quadrat und ein Schachbrett) redundant vorhanden zu sein scheint und entsprechend der Fragestellung die Form ins Bewusstsein gerufen wird, mit der das Problem mit dem geringsten Aufwand gelöst werden kann.

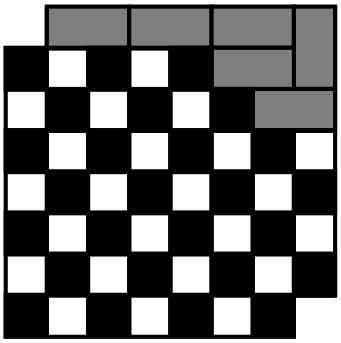

Abbildung 7: Mit Hilfe eines Schachbretts und etwas Nachdenkens erkennt man schnell, dass das im Text beschriebene Problem nicht lösbar ist.

Im Folgenden werden einige Modelle für die Repräsentation von Wissen vorgestellt und näher betrachtet.

2.2.1 Semantische Netze

Die Vorstellung, dass das Wissen allein in den Verbindungen zwischen den einzelnen Neuronen gespeichert ist, ist für das weitere Verständnis zu komplex. Es macht viel mehr Sinn, jeweils viele Neuronen als ein Ganzes zu betrachten, so dass diese jeweils eine größere logische Einheit bilden. So könnte man mit dem Begriff „Vogel“ tausende Durchmesser von bestimmten Dendriten (siehe Kapitel 2.1.1) beschreiben oder eben nur einige Gruppierungen von Neuronen meinen, welche verschiedene Eigenschaften oder Repräsentationen eines Vogels besitzen und miteinander verbunden sind. M. Ross Quillian [30] hat dazu ein Gedächtnismodell entwickelt, welches diesen Zusammenhang sehr einfach visualisiert.

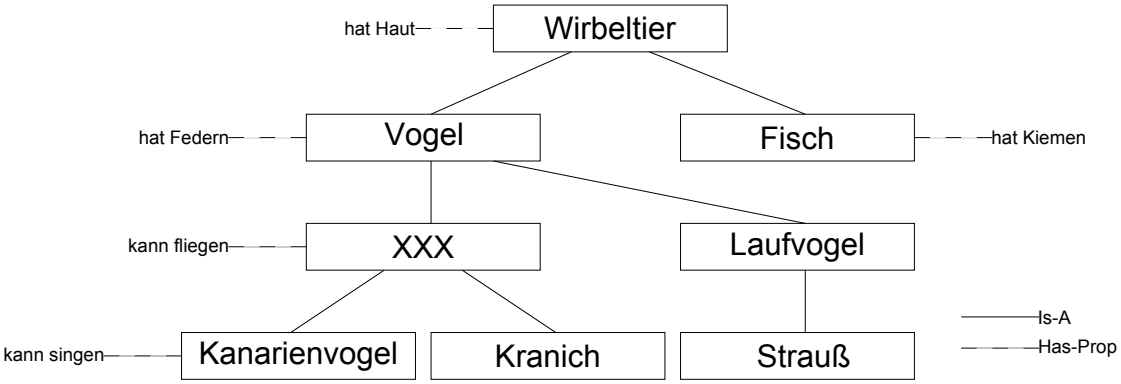

Abbildung 8: Semantisches Netz, welches verschiedene Wirbeltiere beschreibt [13].

Abbildung 8 zeigt das Beispiel eines Ausschnittes eines semantischen Netzes, in welchem Wirbeltiere beschrieben werden. Ein semantisches Netz besteht immer aus Knoten, welche Objekte enthalten. Diese Knoten sind durch „Ist-Ein"(„Is-A")-Beziehung miteinander verbunden. Diese Beziehungen beschreiben eine Ober-/Unterkonzept-Relation. Außerdem gibt es noch die „Hat-Eigenschaft"(„Has-Prop")-Beziehungen, welche den einzelnen Objekten zusätzliche Eigenschaften zuweisen. Mit Hilfe eines solchen Netzes lässt sich umfangreiches, komplexes Wissen abspeichern. Dass dieses Modell der Realität ziemlich nahe kommt, zeigt folgendes Beispiel: Werden einer Versuchsperson zuerst der Begriff „Vogel" und kurz darauf der Begriff „Strauß" genannt, so stellt diese sofort den Zusammenhang beider Wörter her und geht von einem Vogelstrauß aus. Kaum eine Person würde sagen, dass ein Blumenstrauß gemeint war. Viel erstaunlicher ist noch, dass die Versuchsperson einige Zeit nach den genannten Begriffen fest der Meinung ist, dass der Begriff „Vogelstrauß" genannt wurde.

Nicht alles, was über Sinnesorgane aufgenommen wird, wird komplett genauso abgespeichert. Vielmehr wird immer wieder eine Art Zeiger auf vorhandenes Wissen gesetzt. In diesem Beispiel verschwindet also z. B. die Information, dass eigentlich zwei Begriffe genannt wurden - die Referenz auf den Vogelstrauß genügt dem Gehirn.

2.2.2 Scripts

Das natürliche Verhalten eines Menschen, wie bei den genannten Begriffen in Kapitel 2.2.1, ist auch bei den sogenannten Scripts sehr deutlich zu sehen [13]. Scripts sind wie eine Art Drehbuch, welches im Gehirn abgespeichert ist, zu verstehen. In einem Versuch werden einer Person mehrere zusammenhängende Ereignisse, bzw. Aktionen in logischer Reihenfolge erzählt. Das Interessante ist nun, dass diese Person kurze Zeit danach von Ereignissen erzählt, welche gar nicht in der Geschichte genannt wurden. Dazu ein kleines Beispiel:

1. Sie gehen in ein Restaurant.
2. Ein Kellner weist Ihnen einen Platz zu.

3. Der Kellner nimmt Ihre Bestellung auf.
4. Der Kellner bringt das Essen.
5. Sie bezahlen.
6. Sie verlassen das Restaurant.

In dieser Ereignis-Abfolge wurden viele Ereignisse, die während des Essens normalerweise stattfinden, nicht erwähnt, wie z. B. dass der Gast das Essen isst oder der Kellner nachfragt, ob das Essen schmeckt und ob neue Getränke gebracht werden sollen. Trotzdem wird sich nach kurzer Zeit kaum noch jemand an jeden einzelnen Punkt in der Geschichte erinnern können und ist der festen Überzeugung, dass das gebrachte Essen auch gegessen wurde. Dies zeigt, dass der bekannte Restaurantbesuch ohne Details bereits im Gehirn gespeichert war.

2.2.3 Bildhafte Vorstellung

Von allen Sinnen überwiegt die visuelle Wahrnehmung mit ca. 10 Mio. Shannon [36] vor dem Tasten (1 Mio. Shannon), Hören (100.000 Shannon), Riechen (100.000 Shannon) und Schmecken (1.000 Shannon) [3]. Allein durch das Sehen kann man sich schon nahezu perfekt durch die Umgebung bewegen und orientieren. Demnach ist es kaum verwunderlich, dass sich Menschen von vielen Dingen zuerst ein „Bild“ im Kopf machen. Sowohl die einzelnen Ereignisse beim Restaurantbesuch, als auch das Quadrat, welches mit Dominosteinen verdeckt werden soll (siehe Abb. 5), wird zuallererst als mentales Bild ins Gehirn projiziert. Welchen Vorteil das haben kann, erkennt man in folgender Aufgabe [28]:

> „Die schwarze Kugel liegt direkt hinter der weißen Kugel. Die grüne Kugel ist rechts von der weißen Kugel und zwischen den beiden liegt die rote Kugel.“

Die Frage hierzu könnte lauten: „In welcher Richtung meiner Sichtlinie liegt die weiße Kugel, wenn sich die rote Kugel zwischen mir und der schwarzen Kugel befindet?“ Die wenigsten Menschen würden hier z. B. mathematisch vorgehen und sich ein Koordinatensystem mit Vektoren vorstellen. Man versetzt sich vielmehr in die Situation hinein, dass man die Kugeln mit den entsprechenden Farben vor sich hat und „sieht“ dann, dass die weiße Kugel eindeutig schräg links vor einem liegt. Die bildhafte Vorstellung findet jedoch auch bei sehr abstrakten Dingen statt, wie z. B. die anfangs erwähnte Zahl „3“ als drei Äpfel, die auf einer grünen Wiese liegen. Somit bringt die visuelle Vorstellungskraft selbst bei nicht durch die Augen wahrgenommenen Bildern oft Vorteile.

Ein Modell, wie die bildhafte Vorstellung in den Rechner übertragen werden kann, wird im nächsten Abschnitt beschrieben.

2.2.4 Symbolischer Ansatz

Bei dem symbolischen Ansatz geht man davon aus, dass das Gehirn Informationen in Form von Symbolen abspeichert und es geht darum, die Daten in symbolhafter Darstellung zu verarbeiten. So werden alle real existierende Objekte mit ihren Eigenschaften

als Symbole gehandhabt. Diese Symbole können miteinander semantisch oder strukturell zusammenhängen. Ein semantischer Zusammenhang wäre zum Beispiel: Äpfel sind Essen. Ein struktureller Zusammenhang wäre zum Beispiel: Ein Hocker setzt sich aus Beinen und einer Sitzfläche zusammen.

Symbole, die zwangsweise diskret sind, werden mit Algorithmen ausgewertet. Dies entspricht Computerprogrammen, die Symbole (z. B. Zahlen oder Zeichenketten) verarbeiten. Dabei versucht man in Simulationen physikalische und logische Gesetze von mentalen Prozessen zu beachten, um Ergebnisse zu erhalten, die durch psychologische, empirische Experimente validiert werden können. [45]

Der symbolische Ansatz berücksichtigt auch die Möglichkeit zu lernen. Dabei entstehen aus Symbolen und logischen Schlussfolgerungen neue Symbole. Man bezeichnet das als „deduktive Logik“. Die neu erzeugten Symbole können durch Algorithmen genauso verwertet werden, wie die vorgegebenen. Deduktion der Schlussfolgerungen ist beim symbolischen Ansatz wichtig, denn es dürfen keine Ergebnisse durch naives Durchprobieren aller Möglichkeiten erzielt werden. [45]

2.3 Informationsverarbeitung

Die Informationsverarbeitung beschäftigt sich damit, wie die im Gehirn ankommenden Sensordaten vom Gehirn verarbeitet werden. Auf die Verarbeitung der Daten folgt das Resultat, sprich die verarbeiteten Daten rufen Aktionen hervor. Um die Daten zu erfassen, die von Sensoren kommen, wird die Aufmerksamkeit benötigt.

2.3.1 Aufmerksamkeit

Die menschliche Wahrnehmung liefert viele sensorische Daten, die verarbeitet werden müssen. Wenn man alleine die Abtastrate und die Auflösungen der Audio-Visuellen-Wahrnehmung betrachtet, stellt man fest, dass die Menge enorm ist. Angenommen ein Mensch sieht einen farbigen[3] Full-HD-Film mit der Auflösung von 1920x1080 und der Framerate von 25 Frames pro Sekunde, dann nimmt er $1920 \cdot 1080 \cdot 3 \cdot 25 = 155.520.000$ Reize pro Sekunde mit jeweils einem Auge wahr. Bei der Abtastrate des Schalls von 20 kHz, was der Breite des Hörspektrums entspricht, nimmt er 20.000 weitere Reize in jeder Sekunde pro Ohr wahr. In der Summe sind es 311.080.000, d. h. über 300 Millionen Reize pro Sekunde[4]. Hinzu kommen noch taktile (Tastsinn), gustatorische (Geschmackssinn) und olfaktorische (Geruchssinn) Reize. Das menschliche Gehirn ist nicht in der Lage all die Informationen vollständig zu verarbeiten. Deshalb werden Filter angewandt, um relevante Inhalte auszuwählen.

[3] Ein Mensch ohne Sehbehinderung nimmt drei Informationseinheiten pro Punkt auf. Das sind: 1. Blau-gelb-Anteile, 2. Rot-grün-Anteile, 3. Hell-dunkel-Anteile. Deshalb muss man jeden dargestellten Pixel mit drei multiplizieren.

[4] Amplituden (Farbtiefe und Lautstärke) werden analog von einzelnen Perzeptronen wahrgenommen. Das heißt, die Wahrnehmung eines Perzeptrons läuft über einen einzigen analogen Eingang, nicht über mehrere digitalen Eingänge. Deshalb findet keine Multiplikation mit Farbtiefe von 8 Bit pro Farbe und Lautstärkeunterschied von 16 Bit pro Abtasteinheit statt.

Ein Mensch kann gleichzeitig ungefähr 7±2 Chunks (im Kurzzeitgedächtnis merkbare Teile) fokussieren [23]. Entsprechend können die meisten Menschen ohne Anwenden von speziellen Mnemotechniken beim „Koffer packen“-Spiel fünf bis neun Objekte vollständig und in der richtigen Reihenfolge aufzählen.

Damit Menschen die Informationsflut ihrer Sinne auf übersichtliche 7 Chunks reduzieren können, wählt das Gehirn zuerst einen der fünf Sinne (Sehen, Hören, Tasten etc.) aus und dann ein ausgewähltes Sehfeld.

Dies kann mit einem einfachen Selbsttest belegt werden. Man muss nur versuchen die Oberflächenbeschaffenheit von einem verdeckten Objekt mit der Hand zu untersuchen, während man ein Bild betrachtet. Man wird merken, dass die bildhafte Wahrnehmung unterbrochen wird und nur die Eindrücke dieser Hand wahrgenommen werden, aber nicht die der anderen Hand.

Das heißt, im entspannten, unkonzentrierten Zustand „wartet“ ein Mensch auf Reize. Zum Beispiel springt ein Mensch plötzlich auf die Gegenfahrbahn und erscheint im Blickwinkel des Autofahrers. Beim konzentrierten Betrachten erkennt er, dass es ein Mensch aus seinem Bekanntenkreis ist und verwirft gleichzeitig die Reize des Autoradios. Ein anderer Fall wäre: Der Autofahrer fährt ein neues Auto und bemerkt, dass der Schaltknüppel aus Leder ist. Während er die Struktur der Oberfläche genau ertastet, vergisst er alles um sich herum. Robert Solso bezeichnet dieses Phänomen als selektive Aufmerksamkeit [39].

Die selektive Aufmerksamkeit hat den Vorteil, dass Ressourcen des Denkvermögens, z. B. die „Rechenleistung“ dynamisch verteilt werden kann. So kann ein Mensch seine Umgebung mit einem Öffnungswinkel von ca. π betrachten (Abbildung 9), um mögliche Gefahren zu erkennen. Er kann seine Aufmerksamkeit auf einen Punkt konzentrieren, um ein unbekanntes oder schlecht sichtbares Objekt genau zu untersuchen.

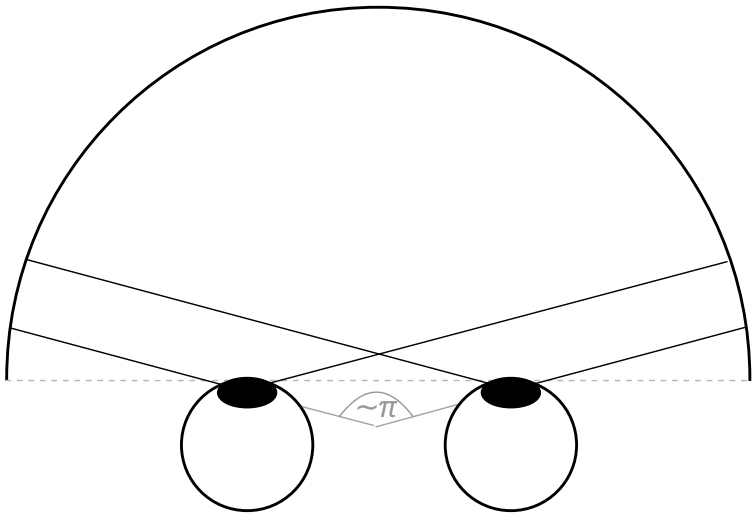

Abbildung 9: Das Sichtfeld eines Menschen. Der Öffnungswinkel des Sichtfeldes ist ungefähr π.

Manche Menschen behaupten, dass sie „multitaskingfähig“ sind und somit ihre Aufmerksamkeit nicht auf einen, sondern auf mehrere Sehfelder gleichzeitig lenken können. In Wirklichkeit ist es ein schnelles Hin-und-Her-Lenken der Aufmerksamkeit zwischen verschiedenen Sehfeldern [12].

Doch worauf richten Menschen ihre Aufmerksamkeit? Meistens sind es unbekannte, interessante und bunte Gegenstände. Allgemein sind es die Objekte, die bei Menschen Emotionen hervorrufen. Dies ist auch in der Werbebranche bekannt. Dort versucht man je nach Thema auf verschiedenste Gefühle der Zielgruppe einzugehen. Dies bestätigt Anja Müller[24]: „Eine emotional starke Marke schaltet nicht nur den Verstand aus, sondern kann auch andere Marken verdrängen. Und auch die Preisgestaltung als Teil des Marketings hat einen Einfluss auf die Vorgänge im Gehirn, zeigt die Neuro-Ökonomin Hilke Plassmann, die seit sieben Jahren unter anderem am renommierten California Institute of Technology und nun im französischen Insead forscht.“

Die Aufmerksamkeit erlaubt es also einem Menschen, die für ihn relevanten Informationen herauszufiltern. Diese Informationen werden anschließend vom Menschen verarbeitet, was durch seine Reaktionen, also sein Verhalten begleitet wird.

2.3.2 Verhalten

Wenn Menschen Aufgaben erledigen, handeln sie. Dabei wählen sie ein Verhalten, um das Problem zu lösen. Eine besondere Herausforderung stellt die Entwicklung der autonomen Agenten[5] dar, weil sie selbstständig mit anderen Agenten und der Umgebung interagieren und somit ein komplexes Verhaltensmodell haben. In diesem Abschnitt wird ein Modell für die Abläufe der Informationsverarbeitung innerhalb eines autonomen Agenten vorgestellt.

Michael Schanz[33] beschreibt Vorgänge in autonomen, d. h. selbstständig handelnden Agenten in seiner Vorlesung „Einführung in die Verteilte Künstliche Intelligenz“ anhand eines Modells des Autonomie-Zyklus. Seine Sichtweise mit Schwerpunkt Robotik kann man vollständig auf Menschen übertragen, denn Menschen interagieren mit der Umwelt ebenfalls autonom, haben Sensoren, Aktuatoren, und verfügen über ein Weltmodell und Planungsvermögen.

In Abbildung 10 erkennt man zwei Bereiche: A und B. Der Bereich A repräsentiert das Untersystem „autonomer Agent“, der Bereich B - seine Umwelt. Der autonome Agent hat vier Wissensteile: „**Weltmodell**“, „**Pläne**“, „**Zustände**“ und „**Aktionen**“. Diese Wissensteile können durch die Aktionen: „*Planung*“, „*Entscheidung*“, „*Auswirkung*“, „*Modifikation*“, „*Verhalten*“, „*Kontrolle*“, „*Handlung*“ und „*Beobachtung*“ aktualisiert und geändert werden. Aktionen sind im Bild durch dicke Pfeile mit Bezeichnungen visualisiert.

Das **Weltmodell** enthält Wissen über die Umwelt, ihre Eigenschaften und mögliche Zustände. Durch eine *Planung* der Aktionen entstehen **Pläne**, was Wissen über mögliche und sinnvolle Handlungsweisen darstellt. Nachdem die *Entscheidung* getroffen wurde, welcher Plan ausgeführt werden soll, wird das Wissen über die Ansteuerung der

[5] Jedes biologische und maschinelle Lebewesen kann als Agent beschrieben werden. Zum Beispiel: Mensch, Katze, Bakterie, Mars-Roboter, Roboterarm etc.

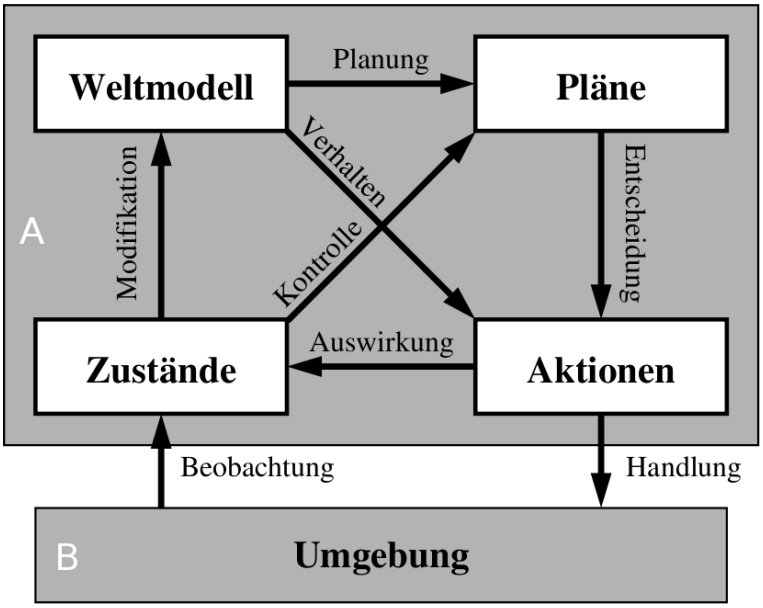

Abbildung 10: Symmetrischer Autonomie–Zyklus [33]. Die möglichen Abläufe innerhalb von einem autonomen Agenten (Bereich A) und seiner Umgebung (Bereich B) sind durch die dicken Pfeile gekennzeichnet.

Aktuatoren (z. B. Handbewegung) verwendet, d. h. eine **Aktion** wird durchgeführt und jede Aktion hat *Auswirkungen* auf den aktuellen **Zustand**. Es ist oft notwendig zu *kontrollieren*, ob der gewählte Plan der richtige war und ob das Produkt zufriedenstellend ist. Da die Abläufe in diesem Modell einen Regelkreis bilden, wird das aktuelle Weltmodell ständig *modifiziert*. So kann ein autonomer Agent lernen und sich an die Umgebung anpassen.

Findet durch eine *Handlung* des Agenten eine Interaktion mit der Umwelt statt, so muss die Reaktion der **Umgebung** durch Sensoren wahrgenommen, d. h. *beobachtet* werden, bevor sie im Zustand des Agenten abgespeichert (siehe dazu Kapitel 2.2) werden kann.

Aus diesem Modell ergeben sich zwei Zyklen: ein interner und ein externer Zyklus. Der interne Zyklus (Abbildung 11, dargestellt durch gepunktete Pfeile) wird bei einem Menschen z. B. beim Denken durchlaufen, wenn er durch logische Rückschlüsse neue Erkenntnisse gewinnt. Bei einem Roboter wäre es z. B. das Prüfen des Ladezustands seiner Batterie und das Entscheiden ob sie aufgeladen werden soll. Der externe Zyklus wird jedes Mal durchlaufen wenn die Umgebung beeinflusst wird (Abbildung 11, dargestellt durch gestrichelte Pfeile). Ein Beispiel wäre das Fertigen eines Werkstücks durch einen Roboterarm: Eine Beeinflussung der Umgebung findet bei jeder Bewegung des Roboterarms und des Werkstücks statt.

Menschen planen nicht immer bevor sie handeln. Sie nutzen zusätzlich zur Planungsfähigkeit die Möglichkeit, auch ohne Plan richtig zu handeln. Solche Handlungsweisen

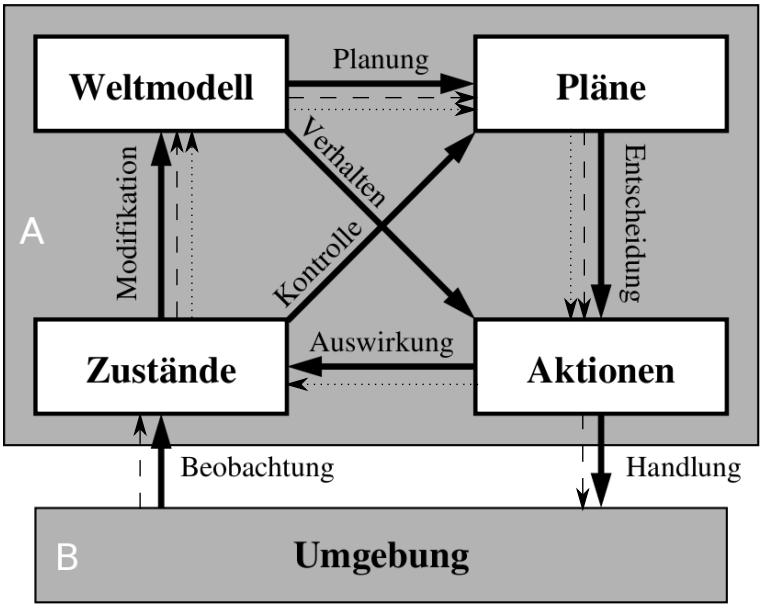

Abbildung 11: Symmetrischer Autonomie–Zyklus [33]. Ein interner Zyklus wird durch gepunktete Pfeile und ein externer kognitiver Zyklus durch gestrichelte Pfeile repräsentiert.

bezeichnet man als reflexartig. Diese werden im nächsten Abschnitt beleuchtet.

2.3.3 Reflexe

Reflexe sind plötzliche Reaktionen auf bestimmte Reize. Zum Beispiel: ein Mensch entfernt die Hand von der heißen Herdplatte. Dies macht er unverzüglich. Reflexe haben den Zweck, Lebewesen vor Gefahren zu schützen. Damit eine Reaktion schnell ausgeführt wird, ist es wichtig, dass der Verarbeitungsweg kurz ist. Nach einem Reiz der Sensoren wird die Planungsphase für eine Aktion übersprungen und reaktives *Verhalten* anhand eines Verhaltensmusters, das angeboren oder im Laufe des Lebens erlernt wurde, ausgeführt. Ist ein Verhaltensmuster nicht vorhanden oder ist der Reiz nicht stark genug einen Reflex hervorzurufen, dann wird zwangsweise eine Planung angestoßen.

2.3.4 Problemlösung und Verarbeitung

Die Planungsphase des oben beschriebenen Modells ist die komplizierteste. In dieser Phase wird eine sinnvolle Handlungsweise ausgearbeitet, um Probleme zu lösen.

In dieser Ausarbeitung wird Bezug auf humanoide Robotik genommen, denn diese Wissenschaft beschäftigt sich damit Roboter zu erschaffen, die Menschen möglichst gut abbilden. So kann man prüfen, ob man mit Annahmen in der Kognitionswissenschaft richtig liegt, indem man die Erkenntnisse, die man in der Forschung gewonnen hat, auf einem Roboter umsetzt. Die Validitätsprüfung zeigt, ob die Annahmen richtig waren.

Es gibt Probleme, die immer wieder in gleicher Form vorkommen, wie z. B. „Türme von Hanoi“[41][6] oder „Optical Character Recognition“[42][7], und es gibt solche Probleme, bei denen ein neuer Lösungsansatz erarbeitet werden muss (vgl. Roboter-Bombe-Batterie-Problem [8][8]).

Grundsätzlich geht ein Roboter bei beiden Arten von Problemen von einem Modell aus. Ein Modell beschreibt die Umwelt des Roboters und seine Möglichkeiten. Die Umwelt eines Sachspiel-Roboters kann z. B. ein Schachbrett sein, und seine Möglichkeiten wären die Regeln, nach denen er die Figuren verschieben darf. Ein Sachspiel-Roboter ist in der Lage gut Schach zu spielen, aber er ist nicht fähig Tic-Tac-Toe zu gewinnen, weil das Modell die anderen Spielregeln nicht vorsieht. Somit beschränkt sich das Können des Roboters auf sein Modell.

Es stellt sich als sehr schwer heraus, ein umfangreiches Modell zu erstellen. Deshalb ist es auch schwierig Roboter für einen universellen, alltäglichen Einsatz auszubauen.

Wie es im Unterkapitel 2.2 beschrieben ist, bilden neuronale Netze die Funktionsweise des menschlichen Gehirns nach. Die neuronalen Netze können zwar selbstständig, ohne Eingabe von Soll-Resultaten lernen, z. B. Gesichter auf Fotos wiedererkennen [20], aber sie sind noch nicht in der Lage, wie ein Mensch, vernünftige neuronale Reorganisationen durchzuführen, um komplexere Sachverhalte zu erlernen.

Es gibt Versuche, das Modell-Problem zu umgehen, indem man Roboter das Modell im Laufe ihrer Existenz selbstständig erweitern lässt wie es neugeborene Lebewesen tun. Wenn man versucht einen Roboter zu erschaffen, der sich wie ein kleines Kind entwickelt, dann stößt man auf extrem große Hindernisse. Es wurde herausgefunden, dass Kinder bereits seit der Geburt über einige wichtige kognitive Fähigkeiten verfügen [20], die aber schwer nachzubauen, d. h. in ein Modell zu überführen, sind. Manuela Lenzen [20] schreibt: „Aus der Kleinkindforschung ist bekannt, dass schon Neugeborene in der Lage sind, Gesichter als solche zu erkennen und Gesichtsausdrücke, etwa das Herausstrecken der Zunge oder das Öffnen des Mundes, nachzuahmen.“

Man kann eine Stufe in der Kognitionswissenschaft herauszoomen und versuchen eine primitive Bakterie statt einen Menschen nachzubauen. Sie muss in der realen Umgebung existieren, nicht als eine Computersimulation, sprich sie muss mechanisch-elektrisch sein. Lässt man sie zu einem elektrischen Menschen evolutionieren, kann man deren elektronische und informationelle Struktur untersuchen, um die internen Abläufe im Gehirn eines biologischen Menschen nachzuvollziehen. So wäre es denkbar das Modell-Problem zu lösen.

[6] In Türme von Hanoi geht es darum, einen vorgegebenen Turm, der aus Scheiben besteht, möglichst effizient zu bewegen, in dem man nur die Scheiben ab- und aufstapelt.

[7] Mit Optical Character Recognition (OCR) ist es möglich einzelne Buchstaben in einem eingescannten Dokument zu erkennen.

[8] Beim Roboter-Bombe-Batterie-Problem wird ein Roboter mit einer Situation konfrontiert, in der er seine Batterie, die neben einer tickenden Bombe liegt, retten muss. Er muss dazu selbstständig eine Lösung finden.

2.4 Grenzen der Kognitionssimulation

Mit der Vision einer vollständigen Kognitionssimulation, lassen sich drei Bereiche herausarbeiten, bei denen diese an ihre Grenzen stößt. Dies ist die Nachbildung eines Bewusstseins, von Ethik und Emotionen, sowie von Kreativität. Zwar haben diese Bereiche keine direkte Notwendigkeit, um Probleme zu lösen, aber sie beeinflussen Entscheidungen und Handlungsweisen beim Menschen doch maßgeblich. Daher werden diese drei Bereiche im Folgenden auf ihre Problematik bei der Simulation untersucht.

2.4.1 Bewusstsein

Mit dem Stand der heutigen Technik ist es möglich, in begrenzten Bereichen, intelligentes Verhalten zu simulieren. Dies ist jedoch nach wie vor beschränkt auf die zugrunde liegenden, vorher implementierten Algorithmen. Die Entwicklung eines eigenen Bewusstseins, welches sich eigene Ziele setzt und eigene Entscheidungen trifft, ist noch weit entfernt. Dazu muss die Komplexität der Systeme erst noch deutlich steigen.

Auch stellt sich die Frage, ob dies wirklich gewollt ist. Eine ausreichende Menge medialer Produktionen beschreibt eine sehr negative Sichtweise. Ein Beispiel wären die unterhalb Zitierten „Drei Gesetzte der Robotik“ von Isaac Asimov [4]. Auch wenn es sich dabei um eine Erfindung aus der Science-Fiction handelt, werden diese immer wieder gerne verwendet und sind vermutlich jedem, der im Gebiet der Robotik forscht oder entwickelt, bekannt. Betrachtet man diese Gesetze einmal genauer, stellt man fest, dass diese dazu dienen, die Fähigkeiten einer Künstlichen Intelligenz einzuschränken auf die Ebene eines Sklaven. Vor allem um einen Schutz davor aufzubauen, dass sich die Maschinen gegen ihre Schöpfer wenden. Dies spiegelt sich auch neben den Gesetzen selbst immer wieder in seinen Geschichten wieder.

> 1. Ein Roboter darf kein menschliches Wesen verletzen oder durch Untätigkeit gestatten, dass einem menschlichen Wesen Schaden zugefügt wird.
> 2. Ein Roboter muss den ihm von einem Menschen gegebenen Befehlen gehorchen – es sei denn, ein solcher Befehl würde mit Regel eins kollidieren.
> 3. Ein Roboter muss seine Existenz beschützen, solange dieser Schutz nicht mit Regel eins oder zwei kollidiert.
>
> Isaac Asimov [4]

2.4.2 Ethik und Emotionen

Ist die erste Hürde des Bewusstseins genommen, kann auf die Begründung von Zielen und Handlungen eines intelligenten Wesens eingegangen werden. Viele Handlungsweisen unter Menschen lassen sich nicht einfach in richtig oder falsch einteilen und logisch begründen. Somit sind sie nur schwer modellierbar. Tatsächlich widersprechen immer wieder rein logisch richtige Entscheidungen jeder moralisch vertretbaren Entscheidung.

Hier gelangen wir an das ethische Verständnis, welches sich nicht allein durch Algorithmik darstellen lässt.

Dabei ist zu beachten, dass das ethische Verständnis auch von Emotionen gebildet wird und viele Entscheidungen maßgeblich durch Emotionen beeinflusst werden. Ein recht einfaches Beispiel hierfür ist der Kauf eines Autos. Die Entscheidung wird hier meist weitgehend subjektiv nach dem „Bauchgefühl“ getroffen und die objektiven Merkmale werden vernachlässigt.

2.4.3 Kreativität

Richtige Kreativität und Kunst basiert nicht nur auf Zufall, sondern auf bewusstem und unterbewusstem Verarbeiten von Sinneseindrücken und Wissen. Ähnlich wie dies beim Träumen geschieht.

Systeme, welche heutzutage automatisch Kunst erzeugen, basieren noch auf einem von zwei Ansätzen. Beim ersten Ansatz erzeugt ein System mit reinem Zufall Ergebnisse, in welche der Mensch selbst Deutungen hinein interpretiert, da unser Verstand immer versucht in unseren Beobachtungen Muster zu erkennen. Der andere Ansatz ist, dass zwar wirklich komplexe Ergebnisse mit einem deutbaren Inhalt entstehen - und nicht nur vom Menschen so interpretiert werden -, dies aber aufgrund aufwändiger Algorithmen geschieht. Somit liegt die Kreativität also beim Entwickler und nicht bei der Künstlichen Intelligenz. Die KI ist immer auf die Verarbeitungsmöglichkeiten beschränkt, welche ihr bei der Programmierung mitgegeben wurden.

Poetron [14] zum Beispiel erzeugt aus Wortlisten und Templates algorithmisch Gedichte. Interessanterweise lassen sich die Ergebnisse dieses simplen Gedichtgenerators durchaus mit Werken von so manchen modernen Künstlern vergleichen.

Eine Huldigung an die Kunst Maschinenwesen.

Ach kreatives Wesen du!
Kunst, mein wolkiger Streit.
Jenseits von Eden in Future und Past!
Erzeugt!
Denkt!
Welch faszinierendes Erbeben!
Maschinenwesen du.
Heiss in zeitloser Galaxy.
Kunst zwischen Wiegen und Denken.
Maschinenwesen meist ach so breit.

Poetron [14] mit den Stichworten „Maschinenwesen“, „Kunst!“, „erzeugen“, „kreativ“

Echte Kreativität, die nicht Ergebnisse aus Algorithmen und Templates erzeugt, sondern wirklich frei einen eigenen Stil findet, wurde noch nicht erreicht.

3 Künstliche Intelligenz

Nach der Untersuchung der Kognition ist auch die technische Seite zu betrachten. Hier geht es darum, wie mit technischen Mitteln Probleme gelöst werden oder kognitive Fähigkeiten simuliert werden können. Allgemein sind Probleme Aufgabenstellungen, welche einer Lösung bedürfen.

Die klassische künstliche Intelligenz (KI) befasst sich ausschließlich mit formalisierbaren Problemen. Dabei werden alle Regeln und Verhaltensweisen in Formeln ausgedrückt und bei der Problemlösung entsprechende Algorithmen verwendet. Diese Art Probleme zu betrachten fällt auch unter den Begriff des symbolischen Ansatzes (Kapitel 2.2.4), im Gegensatz zum konnektionistischen Ansatz (Kapitel 2.1.2). Die Konsequenz aus dieser Art der Problembetrachtung ist, dass eine KI dieser Art nur schlecht mit unerwarteten Situationen umgehen kann, wenn diese nicht in den Regeln der modellierten Welt vorgesehen sind.

Klassische Beispiele für auf diese Art lösbare Probleme sind Spiele mit einer statischen Welt, wie beispielsweise Schach. Reale-Welt-Probleme dagegen sind nicht vollständig beobachtbar und beinhalten zu viele Einflüsse und Aspekte, um diese symbolisch darzustellen.

Im Folgenden wird anhand oberer Beispiele darauf eingegangen, wie formalisierbare Probleme gelöst werden können.

3.1 Suchprobleme

In der klassischen KI werden diese formalisierbaren Probleme oft als Suchprobleme in Zustandsbäumen dargestellt, da für diese diskrete Wege zur Lösungsfindung existieren.

Als Beispiel, wie ein Problem als Baum dargestellt werden kann, kann man wieder einfach an Schach denken. Der Wurzelknoten des Baumes ist der Anfangszustand des Spielbrettes, mit dem Standort aller Figuren und welcher Spieler an der Reihe ist. Die Kanten zu den Kindern des Wurzelknotens stellen die möglichen Züge dar und die Kindknoten somit den Spielstand nach dem Zug. Auf diese Art werden wieder für jeden Kindknoten weitere Kinder gebildet bis zu den Zuständen, an welchen das Spiel zu Ende ist. Abbildung 12 zeigt einen vereinfachten Zustandsbaum für ein Tic-Tac-Toe-Spiel.

Bei den Suchalgorithmen unterscheidet man zwischen zwei Gruppen: der „uninformierten Suche“ und der „informierten Suche“.

Die uninformierte Suche betrachtet nur die bis zum jeweiligen Knoten angefallenen Pfadkosten. Diese Kosten geben den Aufwand an, der nötig ist um den jeweiligen Knoten von der Wurzel aus zu erreichen. Die Suchstrategien unterscheiden sich hierbei darin, in welcher Reihenfolge die Knoten abgesucht werden. Grundstrategien sind hierbei die Tiefensuche und die Breitensuche.

Im Gegensatz zur uninformierten Suche arbeitet die informierte Suche mit Heuristiken, welche zu jedem Knoten eine Vermutung der Kosten bis zum Ziel aufstellen. Dies wäre zum Beispiel bei der Suche eines Weges durch einen Straßenplan eine Angabe der Luftlinienentfernung zum Ziel (Abbildung 15). Auf diese Weise lässt sich beim Durchsuchen des Baumes leichter zwischen günstigen und ungünstigen Pfaden unterscheiden

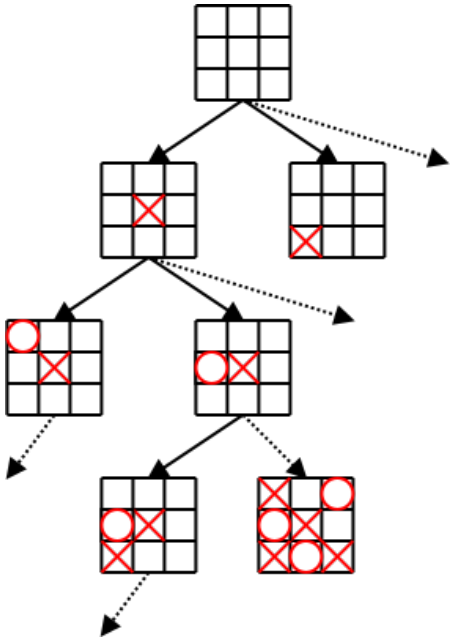

Abbildung 12: Zustandsbaum für ein Tic-Tac-Toe-Spiel mit möglichen Zügen der Spieler. Gepunktete Linien bedeuten, dass der Baum hier gekürzt wird.

und schneller eine Lösung finden.

Solche Heuristiken sind besonders in der kognitiven künstlichen Intelligenz von besonderer Bedeutung. Dies wird in Kapitel 4 weiter ausgeführt.

3.2 Expertensysteme

Für Probleme wie Schach ist der Ansatz einer einfachen Baumsuche jedoch nicht ausreichend. Um mit (guten) menschlichen Spielern mitzuhalten, muss die KI die Züge für einige Runden voraussehen, wenn dies schon nicht bis zum Spielende versucht wird. Zusammen mit der großen Zahl der möglichen Züge in jeder Runde, ergibt sich daraus ein komplexer Suchbaum, welcher unmöglich komplett durchsucht werden kann.

Bei einem Verzweigungsfaktor von etwa 35 möglichen Zügen und durchschnittlich 100 Halbzügen (50 Runden) pro Spiel, sind dies 35^{100}, also 10^{154} Zustände, welche durchsucht werden müssen.

Hier kommen so genannte Expertensysteme zum Einsatz. Diese sind nicht dazu gedacht allgemeine Probleme zu lösen. Dafür lösen sie effizient und hoch spezialisiert sehr eng eingegrenzte Aufgaben. Auf diese Weise wird das Lösen solch komplexer Aufgaben, in akzeptabler Geschwindigkeit, überhaupt erst möglich.

Im Beispiel eines Expertensystems für Schach bedeutet dies, dass unter anderem Datenbanken von Spielsituationen, Strategien und Fachwissen von Schachgroßmeistern in das System eingepflegt werden.

3.3 General Problem Solver

Der Gegensatz zu diesen hoch spezialisierten Systemen sind so genannte allgemeine Problemlöser. Ein Vorstoß in diese Richtung ist der „General Problem Solver“ (GPS) von Herbert Simon und Allen Newell [27]. Beim Lösen von Problemen zerlegt dieser die Probleme in kleinere Teilprobleme mit leichter zu erfüllenden Teilzielen. Dieses Prinzip wird Problemreduktion genannt. Letztendlich stellte sich heraus, dass auch der GPS keineswegs allgemein Probleme lösen konnte, sondern nur formalisierte Probleme aus beispielsweise den Bereichen Logik und Geometrie.

Die Schwierigkeiten und das Scheitern von Versuchen echte allgemeine Problemlöser zu entwickeln, führte dazu, dass vorwiegend Expertensysteme entwickelt werden. Aus dem GPS Paradigma wurde jedoch später das Kognitionsframework Soar entwickelt (Mehr zu Soar im Kapitel 8).

4 Kognitive Künstliche Intelligenz

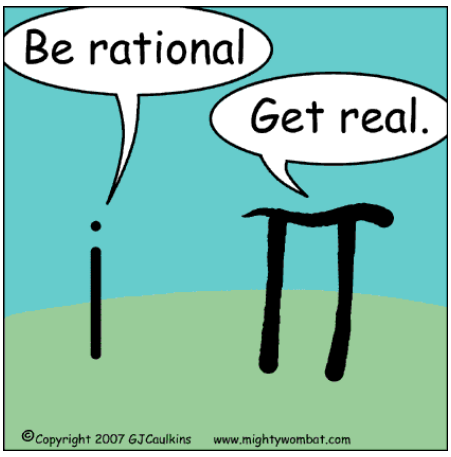

Abbildung 13: Rationalität ist nicht immer die Lösung [48]

Auch wenn der Begriff der kognitiven künstlichen Intelligenz in der Literatur nicht auftaucht, wird hier dennoch eine Spezialisierung der reinen künstlichen Intelligenz vorgenommen, da so verdeutlicht werden soll, dass damit ein größeres Augenmerk auf die natürlichen Denkprozesse im menschlichen Gehirn gelegt wird. Die einzelnen Algorithmen werden in der Kognitiven künstlichen Intelligenz grundsätzlich nicht nur so entworfen, dass die Lösung an sich korrekt ist, sondern auch der Weg dorthin dem menschlichen Denkprozess ähnelt. Dabei dürfen die Lösungen sowohl rational als auch emotional geprägt sein. Um diesen Prozess simulieren zu können, ist es im Hinblick auf die kognitive künstliche Intelligenz notwendig, auch die Hardware in einem Computer ähnlich einem Gehirn in einzelne Komponenten zu unterteilen. Die Software eines Computers, bzw. die Kommunikation zwischen den Hardwarekomponenten übernimmt dabei eine Metaebene, welche ebenfalls in jedem Gehirn zu sein scheint.

4.1 Die „Hardware“

Ein wichtiges Element des menschlichen Gehirns ist das Kurzzeitgedächtnis. Der Großteil dessen, was im Kurzzeitgedächtnis gespeichert ist, ist dem Menschen aktuell bewusst. Es ist somit naheliegend, dass das Kurzzeitgedächtnis unter anderem das Bewusstsein beschreiben könnte. Alles andere, was in der Zeit länger zurückliegt oder aktuell nicht benötigt wird, gilt als unterbewusst und befindet sich zum Großteil im Langzeitgedächtnis. Wichtig ist, dass einem Menschen nie alles gleichzeitig bewusst ist, sondern dass man kleine Wissens-Pakete zuerst aus dem Unterbewusstsein „laden“ muss. Kein Mensch wird z. B. ständig bewusst daran denken, wo er sich gerade befindet. Fragt man ihn jedoch danach, wird er dieses Wissen „laden“ und sofort auf die Frage antworten können. Auch verschwindet dieses Wissen aus dem Kurzzeitgedächtnis nicht sofort, sondern bleibt noch kurze Zeit danach bestehen.

Abstrahiert man nun dieses Modell, erkennt man einen großen Bezug zur Computer-Hardware. Das Kurzzeitgedächtnis ist der Arbeitsspeicher, das Langzeitgedächtnis die Festplatte und das „Nachglühen“ von gerade ins Bewusstsein geholtem, welches eventuell kurz darauf wieder benötigt werden könnte, ist der Cache. Die Steuerung dieses Ablaufs im Gehirn erfolgt durch eine Ebene, welche sich außerhalb der einzelnen Komponenten befindet. Diese „Metaebene“ kann z. B. als der Geist interpretiert werden. Im Hinblick auf einen Computer würde sie dem Abarbeiten eines Softwareprogramms gleich kommen.

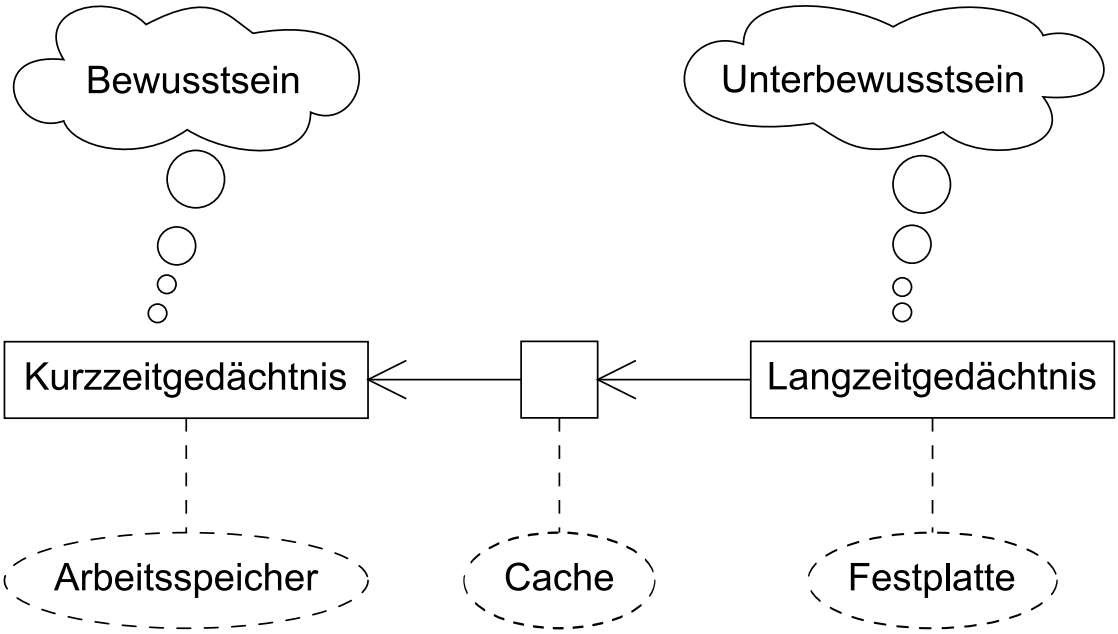

Abbildung 14: „Ladevorgang“ vom Unterbewusstsein ins Bewusstsein: Unterbewusste Gedanken, welche im Langzeitgedächtnis (Festplatte) gespeichert sind, werden über den „Cache“ ins Bewusstsein (Kurzzeitgedächtnis/Arbeitsspeicher) geholt.

4.2 Die „Software“

Im Bereich der reinen künstlichen Intelligenz besteht Software meist nur aus Algorithmen, welche Probleme auf sehr rationaler Ebene lösen und dabei keinerlei Bezug auf die Denkweise des Menschen nehmen. Allerdings können nicht alle kognitiven Prozesse rational beschrieben werden. Im Bereich der Mathematik beispielsweise ist nicht ausschließlich der Verstand gewinnbringend. Da es z. B. keinen allgemeinen Algorithmus gibt, welcher mathematische Beweise führen kann [35], spielt Intuition eine sehr große Rolle. Ein Mathematiker, der lange an der Lösung eines Problems arbeitet, kann plötzlich während eines Spaziergangs die bisher fehlende Idee haben. Hierbei ist es schwierig die Grenze zwischen Rationalität und Intuition zu ziehen. Aufgrund der vielen äußeren

Einflüsse und Wahrnehmungen bei einem Spaziergang (frische Luft, Beobachten von Tieren...) „verschwimmt“ die Rationalität, sodass von Intuition geredet werden kann. Sitzt der Mathematiker hingegen vor seinem Schreibtisch und ist nur in seinen Beweis vertieft, tendiert das Gehirn aufgrund der geringen äußeren Einflüsse zu rationalem Handeln.

Auch wenn die Wissenschaft noch weit davon entfernt ist, das intuitive Handeln von Lebewesen zu verstehen, so gibt es dennoch ein paar Ansatzpunkte. Als Beispiel betrachtet man dazu eine Landkarte. Lautet die Aufgabe, einen möglichst kurzen Weg von einer Stadt zur nächsten zu finden, so beginnt man für gewöhnlich rein intuitiv, einen möglichst kurzen Weg nachzuzeichnen. Die Frage ist nun, was dies für eine „Intuition“ ist an der sich die Person festhält? Sehr wahrscheinlich wird die Versuchsperson sagen, dass sie sich an der Luftlinienentfernung orientiert hat oder bekannte kürzeste Strecken zwischen je zwei Orten nutzt (Abbildung 15). Genau diese Intuition wird auch in heutigen Navigationsgeräten eingesetzt. Dort wird oft der sogenannte A*-Algorithmus verwendet, in welchem die Luftlinienentfernungen miteinbezogen werden [16]. In der Algorithmik wird die hier erwähnte Intuition allgemein als „Heuristik“ bezeichnet.

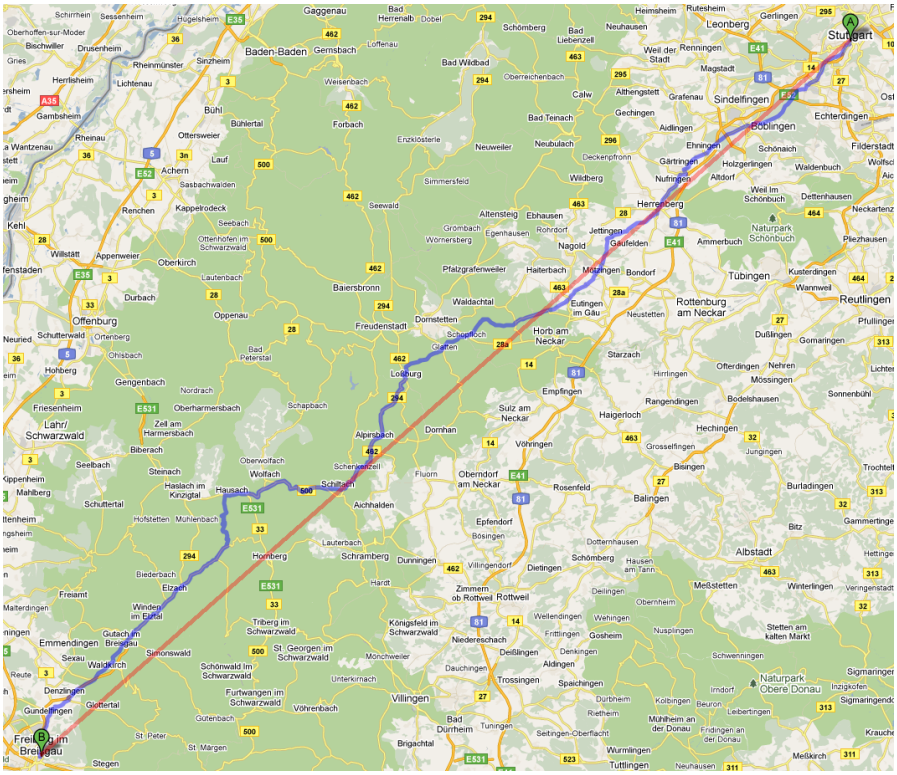

Abbildung 15: Die Luftlinie zwischen Stuttgart und Freiburg (blau). Die kürzeste Strecke auf Straßen orientiert sich an dieser Linie (blau). (Bild: Google Maps [11])

Da der A*-Algorithmus allein durch diese Anwendung jedoch noch viel zu langsam wäre, wird zusätzlich die Heuristik der gespeicherten kürzesten Wege verwendet. In ei-

nem Navigationsgerät sind diese zwischen großen Städten fest eingespeichert, auf welche jederzeit zugegriffen werden kann.

Aufgrund von mehr Regeln, ist das Simulieren eines Schachspielers viel schwieriger als das Finden des kürzesten Weges beim Navigationsgerät. Theoretisch könnte das Schachprogramm zu jedem Zeitpunkt alle Möglichkeiten des Gegners und für diese, alle seine eigenen Möglichkeiten vorberechnen. So könnte das Programm jederzeit den Schritt durchführen, der langfristig gedacht den höchsten Gewinn erbringen wird. Allerdings gibt es da ein großes Problem: Wie bereits in Kapitel 3.2 angemerkt, gibt es eine enorm große Menge an Spielmöglichkeiten. Alle Wege auszuprobieren würde also viel zu lange dauern. Die Frage ist jedoch, wie es sein kann, dass einige Schachweltmeister scheinbar mehr Schritte vorausdenken können als die meisten Schachcomputer. Auch hier liegt die Antwort wieder in der Intuition. Ein Mensch mit Schachspiel-Erfahrung wird niemals alle Möglichkeiten durchdenken, sondern nur die, die seiner „Heuristik“ nach den höchsten Gewinn versprechen. Wie in Kapitel 2.3 beschrieben, kann man die durchdachten Möglichkeiten beim Schach auch als „Chunks“ bezeichnen und daraus schließen, dass es weit weniger Möglichkeiten pro Spielzug sind, die ein Schachspieler ausprobiert als ein Computerprogramm [23].

Ein weiteres Beispiel wäre eine grafische Benutzeroberfläche. Der Benutzer wird beim Suchen der Druckfunktion zuerst ein halbes Dutzend Buttons im linken, oberen Bereich des Programms ins Auge fassen bevor er, falls nicht gefunden, die anderen Bereiche nach dem entsprechenden Druck-Button durchsucht. Diese Heuristik lässt sich auf die Leserichtung der lateinischen Schrift zurückführen und dementsprechend auch teilweise in intelligenten Programmen umsetzen. Viel komplizierter ist das bei der Schach-Heuristik. Heutzutage ist es noch nicht möglich, vollständig zu beschreiben, welche Chunks in welchem Schritt ausgewählt und weiterverfolgt werden müssen. Eine der größten Herausforderungen der kognitiven künstlichen Intelligenz ist es also, für alle möglichen Aufgaben Heuristiken zu finden, welche dem menschlichen Denkprozess ähneln und damit in angemessener Zeit zu sinnvollen Lösungen führen.

Um der Lösung etwas näher zu kommen und man sich nicht bei jedem Problem erneut Gedanken machen muss, verwendet man häufig sogenannte Kognitionsframeworks. Drei davon werden im folgenden Kapitel näher beschrieben und miteinander verglichen.

5 Kognitionsframeworks

Wir betrachteten in den Kapiteln 3 und 4 die Intelligenz aus verschiedenen Sichtweisen. Nun geht es darum, das menschliche Verhalten mit technischen Hilfsmitteln zu simulieren. Dabei müssen die Denkweise und die Vorgänge im menschlichen Gehirn möglichst realitätsnah abgebildet werden.

Zunächst muss geklärt werden was Frameworks sind. Nach der Begriffsklärung wird auf die Einsatzmöglichkeiten der Kognitionsframeworks eingegangen und anschließend wird vorgestellt, welche Frameworks in dieser Ausarbeitung untersucht wurden.

5.1 Begriffsklärung

Ein Framework ist etwas Ähnliches wie ein Entwurf, eine Attrappe des zu simulierenden Wesens. Ein Framework schreibt vor, wie einzelne Eingaben zu verarbeiten sind, in welcher Form Wissen gespeichert wird und wie die Ausgabe der Ergebnisse stattfindet.

Es gibt Frameworks, die nur als eine Theorie oder ein Konzept existieren, aber keine Implementierung haben. Zum Beispiel ist „Society of Mind“ ein solches Framework. Push Singh[38] schreibt über das „Society of Mind“ von Marvin Minsky: „Despite the great popularity of the book The Society of Mind, there have been few attempts to implement very much of the theory. One difficulty is that Minsky presents the theory in fragments and at a variety of levels, and the more 'mechanical' aspects of the theory are largely distributed throughout the text, and only specially distinguished in the glossary.“

Alle Frameworks, die diese Fachstudie untersucht, verfügen über eine ausführbare Implementierung. Dies erlaubt eine Ergebnisbasierte Bewertung der Frameworks.

Kognitionsframeworks können nicht nur Menschen und Tiere simulieren. Man kann mit ihnen ein neues Wesen mit einer komplett anderen Denkweise, Gehirnstruktur und Kognitionsfähigkeit simulieren. Denkbar ist auch eine Simulation mit neuartigen Neuronen, deren Aufbau vom aktuellen Forschungsstand abweicht. So kann man mit Frameworks auch Neuronenforschung betreiben und untersuchen wie sich Wesen mit einer weiterentwickelten, oder vollständig neuen Art der Neuronen und neuronalen Netzen verhalten. Dies eröffnet Wissenschaftlern neue Möglichkeiten, das Gehirn zu verstehen.

Es gibt Frameworks, die den konnektionistischen Ansatz (siehe Kapitel 2.1.2) verfolgen (z. B. Leabra), sowie die, die den symbolischen Ansatz (siehe Kapitel 2.2.4) umsetzen (z. B. Soar, Kapitel 8). Bei manchen Frameworks werden die beiden Ansätze auch kombiniert. Entsprechend dem Aufbau der Frameworks unterscheiden sie sich in ihrem Einsatzgebiet. So eignen sich konnektionistische Frameworks für Kognition und die symbolischen Frameworks für die algorithmischen, logischen und mathematischen Aufgaben.

Was die einzelnen Einsatzbereiche der Kognitionsframeworks bereits heute sind, wird im nächsten Abschnitt erklärt.

5.2 Einsatzmöglichkeiten der Kognitionsframeworks

Das grundlegende Ziel der Kognitionsframeworks ist das menschliche Gehirn zu simulieren. In der Theorie kann man mit Hilfe eines Frameworks einer Maschine die menschlichen kognitiven Fähigkeiten verleihen. Dies bedeutet, dass der Einsatz von kognitiven Frameworks sehr vielfältig sein kann. Im Kapitel 9 werden wir feststellen, dass die Frameworks in der Praxis die menschlichen kognitiven Fähigkeiten nur ansatzweise simulieren können. Das heißt, dass die Anwendungsgebiete in der Praxis weniger vielfältig, aber dennoch enorm sind.

Kognitive Frameworks finden zurzeit größtenteils Anwendung in der Forschung und das sogar in einer Art, die man nicht sofort erwarten würde. So werden einige davon dafür benutzt, die Funktionsweise des Gehirns zu verstehen und nicht nur formalisierte Probleme zu lösen. Dazu werden Programme erstellt, welche den vermuteten Abläufen im Gehirn im Rahmen des dabei verwendeten kognitiven Modells ähneln. Stimmen die Ergebnisse der Simulation mit denen von menschlichen Probanden überein, kann man daraus schließen, dass das Gehirn ähnlich arbeitet. Im nächsten Schritt kann man in der Richtung weiterforschen, um auch andere psychologische Verhaltensweisen verstehen zu können. Man könnte sich vorstellen, dass menschliche Probanden bei einigen Versuchen überflüssig werden, da deren Verhalten komplett simuliert werden kann. Daraus würde dann z. B. die Werbebranche sehr profitieren.

Jedoch gibt es auch weitere Bereiche, wie die Spiele- und Unterhaltungsbranche. So können Video-Spiele und Avatare der Unterhaltungsanwendungen mit einem menschen- oder tierähnlichen Verhalten versehen werden, was Produkte für Kunden noch attraktiver machen würde.

Ein Beispiel für ein erfolgreiches Spielzeug ist der Roboter-Hund „Aibo“ von Sony (Abbildung 16). Aibo zeigt für Hunde typische Verhaltensweisen. Er ist lernfähig und kann zudem mit der Außenwelt durch Schall und Aktuatoren kommunizieren. In diesem Spielzeug wird ein Kognitionsframework der Carnegie Mellon University „Tekkotsu“ verwendet [44].

Um tieferes Verständnis der Materie zu erhalten, wurden im Rahmen dieser Fachstudie Kognitionsframeworks im praktischen Einsatz erprobt. Es wurden drei Kognitionsframeworks ausgewählt, untersucht und miteinander verglichen.

5.3 Getestete Frameworks

Die drei Kognitionsframeworks, die für die praktische Untersuchung gewählt wurden, sind:

- „ACT-R“ (siehe Kapitel 6)
- „Apex“ (siehe Kapitel 7)
- „Soar“ (siehe Kapitel 8)

Bei der Wahl wurde auf die Diversität der Eigenschaften (wie z. B. Einsatzbereiche und Architektur) von Frameworks geachtet, damit ein möglichst breites Wissen des

Abbildung 16: Roboterhund Aibo von Sony. [40] Sein kognitives Können hat er dem Framework Tekkotsu zu verdanken.

praktischen Einsatzes gewonnen wird. Des Weiteren hat jedes kognitive Framework seine eigenen Stärken, die sich bei den ausgewählten Frameworks kaum überlappen.

Es wurden Frameworks getestet, die nicht nur theoretisch existieren, sondern auch über eine Implementierung verfügen und auf einem PC ausführbar sind. Wichtig ist auch, dass sie immer noch entwickelt werden oder ihre Entwicklung erst vor kurzem eingestellt wurde, um die Aktualität sicherzustellen.

So wird **ACT-R** oft in der Literatur referenziert und gilt als ein mächtiges Werkzeug, um kognitive Prozesse zu simulieren. Es ist universell im Einsatz und löst ein breites Spektrum an formalisierbaren Problemtypen.

Dagegen befasst sich **Apex** verstärkt mit der Entwicklung von autonomen Systemen auf Basis von Agenten und nur in Randbereichen mit kognitiven Prozessen. Apex wird von NASA entwickelt und eingesetzt.

Auch **Soar** wird ebenfalls häufig in der Literatur erwähnt und gilt als Vorbild für ACT-R. Es wird auch in einem großen Bereich eingesetzt, jedoch hat dieses Framework andere Stärken (siehe Kapitel 9).

Der Startpunkt der Untersuchungen (siehe Kapitel 6, 7 und 8) ist die Beschreibung des Aufbaus des Frameworks, was den theoretischen Hintergrund beleuchtet. Die praktische Anwendung der einzelnen Frameworks ist im Unterkapitel zu Einsatzgebieten beschrieben. Die Frameworks haben Ein- und Ausgabeschnittstellen, welche Auskunft über die Vorgänge innerhalb des Kognitionsframeworks geben oder sie beeinflussen. Da Frameworks auch ihre Schwachstellen/Grenzen haben, sind diese ebenfalls im Text beschrieben. Bevor die Frameworks in der Praxis eingesetzt werden können, müssen sie installiert werden, weshalb für jedes Framework eine kurze Anleitung für die Inbetriebnahme gegeben ist.

Die nächsten drei Kapitel beschreiben nun die einzelnen Kognitionsframeworks ausführlich.

6 Framework ACT-R

ACT-R ist ein Kognitionsframework, das Ende der 90er Jahre entstanden ist und hauptsächlich von John Robert Anderson (Abbildung 17) geprägt wurde. Seine Inspiration dafür war der Kognitionswissenschaftler Allen Newell. [46]

Abbildung 17: Foto von J. R. Anderson, dem Entwickler des Kognitionsframeworks ACT-R [2]

Dieses Kognitionsframework verwendet ein Dialekt der Programmiersprache Lisp [7]. Es ist modular aufgebaut. Es gibt Module, die bestimmte Aufgaben haben; Man kann sie mit Gehirnregionen vergleichen. Die Verbindung zwischen Modulen wird durch Buffer sichergestellt. Buffer sind Kanäle, durch die Informationseinheiten transportiert werden.

Eine Besonderheit von ACT-R ist, dass es Metriken beim Arbeiten erstellt. Zum Beispiel kann man nach der Ausführung eines ACT-R-Programms einsehen wann eine bestimmte Aktion zum ersten Mal ausgeführt wurde. Deshalb können die Arbeitsergebnisse von ACT-R und die Ergebnisse der Messung eines physischen Experiments verglichen werden, um die Validität der Simulation zu prüfen.

ACT-R verfügt über eine grafische Oberfläche, die Zugriff auf alle Programmfunktionen bereitstellt. Es empfiehlt sich, eine hohe Bildschirmauflösung zu verwenden, damit alle GUI-Elemente dargestellt werden können, ansonsten fehlt der Zugriff auf einige Funktionen (Abbildung 18).

6.1 Aufbau/Architektur

Die Architektur von ACT-R besteht aus der Wissensspeicherung, die von Modulen durch Buffer verwaltet wird. Der pattern matcher ermöglicht das sinnvolle Verwalten des Wis-

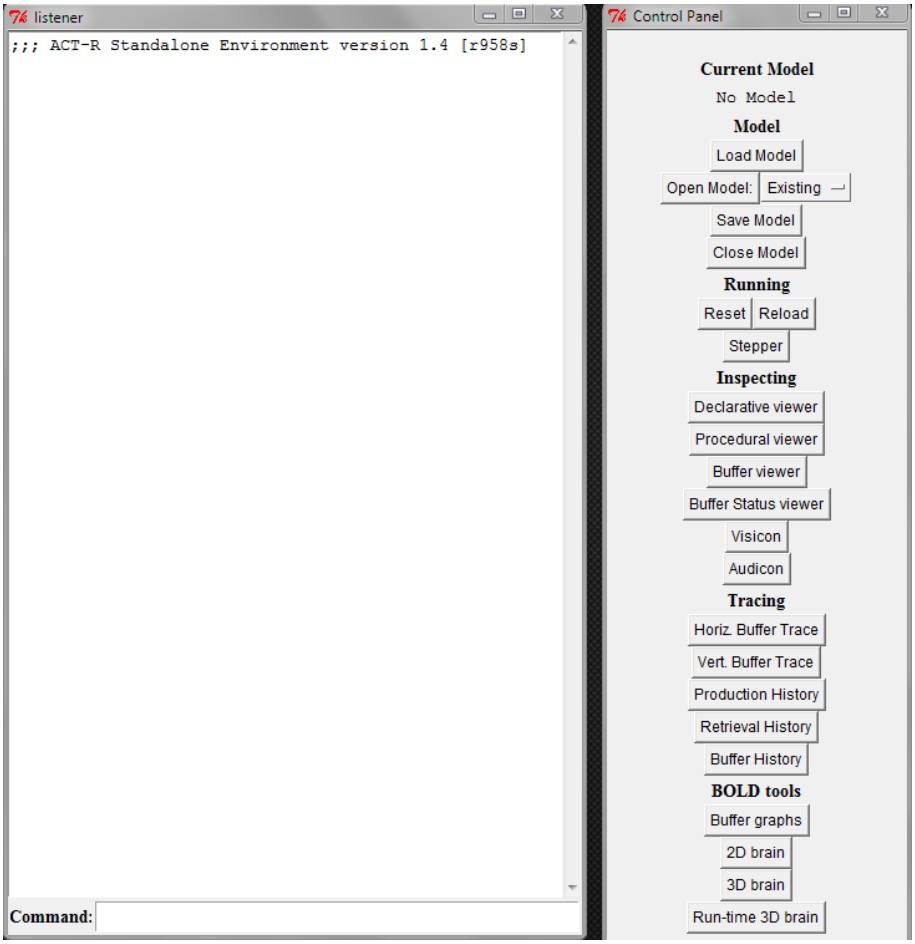

Abbildung 18: Oberfläche des Kognitionsframeworks ACT-R. Die Bedienung erfolgt über Buttons im „Control Panel“ und über Kommandos im „Listener“.

sens. Der hybride Ansatz gibt dem Framework mehr Freiheitsgrade. Und mit Hilfe der Wissensproduktion ist ACT-R lernfähig.

Deklaratives und prozedurales Wissen ACT-R unterscheidet zwei Arten von Wissen: deklaratives und prozedurales Wissen. So werden Fakten im deklarativen und Regeln im prozeduralen Gedächtnis gespeichert.[6]

Das **deklarative Wissen** wird in ACT-R durch so genannte Chunks repräsentiert. Diese Chunks verfügen über einen individuellen Namen, den Typ, der angibt zu welcher Kategorie der Chunk gehört und die Slots, die Attribute in Form von „Key-Value-Paaren“ speichern (Abbildung 19) [6].

Das **prozedurale Wissen** wird durch WENN-DANN Regeln repräsentiert, z. B.: *WENN „Erde trocken“ DANN „gieße Pflanze“*. Ist der Bedingungsteil erfüllt, so wird

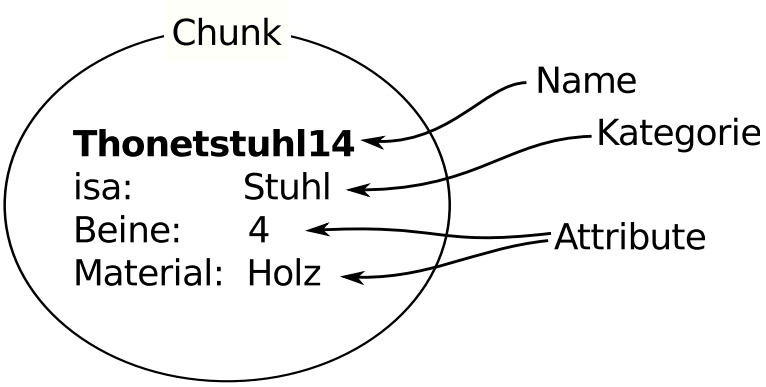

Abbildung 19: Grafische Darstellung eines Chunks. Er enthält seinen Namen, seine Kategorie, sowie Attribute, die ein Objekt beschreiben.

der Schlussfolgerungsteil ausgeführt. Eine solche Bedingungsanweisung wird als „Produktion“ bezeichnet [6]. Tabelle 1 stellt einige Produktionsregeln tabellarisch dar.

Bedingung	Schlussfolgerung
Erde trocken	Gieße Pflanze
Raum ist dunkel	Mache Licht an
Akku fast leer	Akku aufladen

Tabelle 1: Tabelle mit beispielhaften Produktionen. Jede Zeile ist eine Produktion mit zwei Teilen: dem WENN-Teil und dem DANN-Teil.

Module und Buffer Zur ACT-R-Theorie gehören Module und Buffer. Module verarbeiten die einzelnen kognitiven Aufgaben oder simulieren die Arbeitsweise eines ganzen Gehirnbereichs. So existieren Module für das prozedurale, deklarative Wissen, sowie für die Verarbeitung der Perzeption und Motorik. Es ist möglich weitere Module anzuschließen, die ACT-R erweitern. [6]

Einzelne Module für sich sind wie Inseln. Sie brauchen eine Verbindung zueinander, um Informationen auszutauschen und zusammenarbeiten zu können. Dafür gibt es Buffer. So kann ein Chunk im Buffer zwischen zwei Modulen übergeben werden, was den Informationsaustausch ermöglicht. [6]

Pattern matcher Der Pattern matcher betrachtet den aktuellen Zustand des ausgeführten Modells, also die einzelnen Einträge in Buffern und sucht nach einer zu diesem Zustand passenden Produktion und führt diese aus [7]. Es kann immer nur eine Produktion pro Zeitschritt ausgeführt werden. Wenn eine Produktion feuert (ausgeführt wird), verändert sich der Zustand des Modells und es muss neu entschieden werden, welche

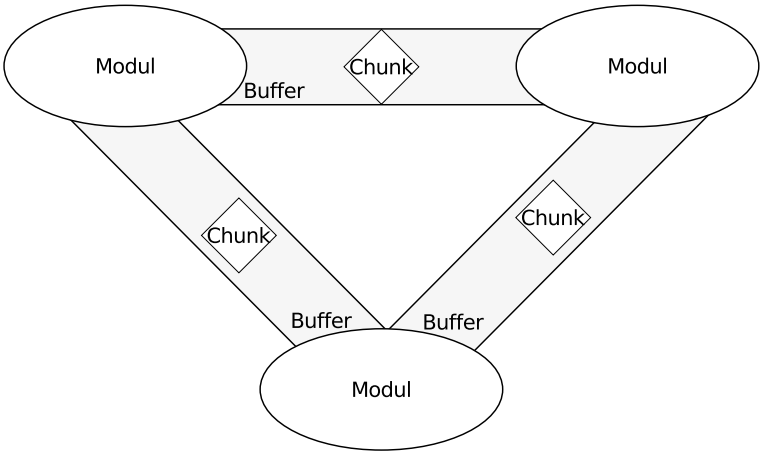

Abbildung 20: Darstellung der Kommunikationsstruktur der Module, Buffer und Chunks. Die Struktur hängt von den verwendeten Modulen ab.

Produktion die passende für den neuen Zustand ist.

Hybride Vorgehensweise ACT-R arbeitet auf Basis des symbolischen Ansatzes, indem es konkrete Objekteigenschaften und Regeln verwendet. ACT-R verwendet zusätzlich zum symbolischen Ansatz den „subsymbolischen Ansatz“, mit dem die konnektionistische Komponente (siehe „Konnektionistischer Ansatz“ in Kapitel 2.1.2) durch eine probabilistische, bzw. kombinatorische Komponente simuliert wird. Sie sorgt dafür, dass alle möglichen Produktionen mit Wahrscheinlichkeiten versehen werden. Die Produktion mit der höchsten Wahrscheinlichkeit wird letztendlich ausgeführt. Die Wahrscheinlichkeit setzt sich aus der Bewertung des Gewinns, der Erfolgswahrscheinlichkeit[9] und des Rauschens (für Nichtdeterminismus) zusammen [7] [21].

Wissensproduktion Das Wissen wird in Chunks gespeichert. Ein Mensch kann neues Wissen gewinnen, indem er lernt. Ein Lernprozess wurde auch bei ACT-R implementiert.

ACT-R kann beim Ausführen der Produktionen neue Chunks erzeugen, sodass sie beim Ausführen der nächsten Produktion verwendet werden können. Der Lernprozess sieht es nicht vor, neue Produktionsregeln zu erzeugen, was dazu führt, dass das simulierte Gehirn neue Abläufe nicht erlernen kann. Der Lernprozess kann nur faktenbasiertes Wissen erweitern. Das heißt, dass das Gehirn zum Beispiel Eigenschaften der neuartigen Objekte erlernt, aber keine neuen Umgangsweisen.

[9] Die Erfolgswahrscheinlichkeit wird vom Modul „goal module“ geschätzt. Die Basis für Schätzung ist das in Produktionsregeln angegebene Ziel. [32]

6.2 Einsatzgebiete

Der Einsatz von ACT-R findet in verschiedensten Wissenschaften statt. Das Framework wird in der Psychologie, den Neurowissenschaften, der Bildung und Robotik, sowie in dem Bereich der Mensch-Maschine-Interaktion angewendet (Abbildung 21).

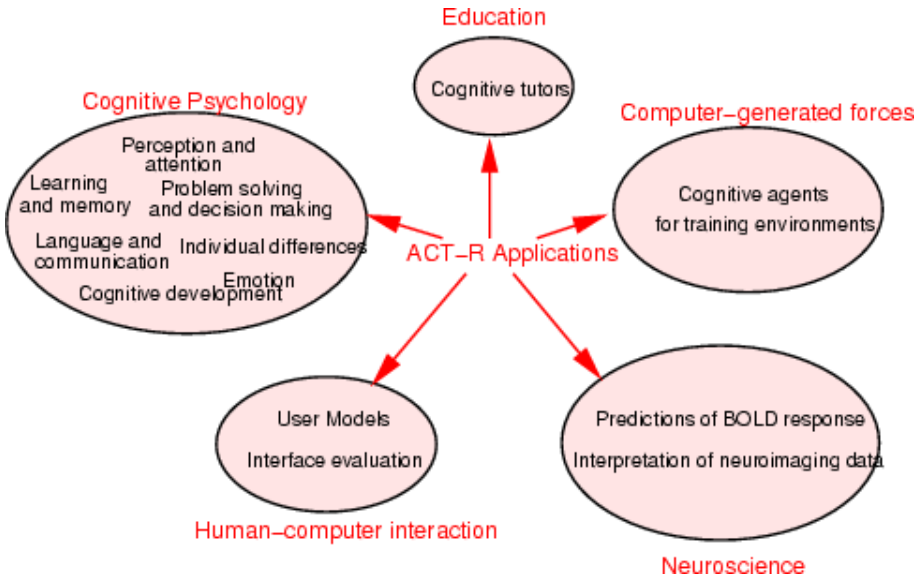

Abbildung 21: Grafische Darstellung der Einsatzgebiete von ACT-R [7]. Die einzelnen Einsatzgebiete, dargestellt als Ovale enthalten konkrete Anwendungsbereiche.

Zum Beispiel können mit ACT-R folgende Simulationen durchgeführt werden: [7]

- Lernen und Merken von Wortlisten oder Texten
- Sprache und Kommunikation (Sprachverstehen)
- Flugzeugsteuerung
- Problemlösen und Urteilen (z. B. Türme von Hanoi)
- Wahrnehmung und Aufmerksamkeit beim Menschen
- Kognitive Entwicklung von Schülern
- Bedienung verschiedener Schnittstellen zum PC
- Probleme, die beim Verstehen des Lernstoffes auftreten können
- Neurologische Vorgänge beim Menschen

Neben den Simulationen selbst bietet ACT-R auch einen Einblick in die Abläufe der Simulationsprozesse. Dies eröffnet neue Möglichkeiten, wie zum Beispiel das Untersuchen der GUI-Prototypen, um menschliches Verhalten vorherzusagen. Aus der Untersuchung dieser Simulationen kann man bessere Bedienungsanleitungen für Programme schreiben und diese anschließend mit ACT-R und menschlichen Probanden gegentesten. Desweiteren kann man die Verständnisprobleme von Schülern lokalisieren und im Unterricht speziell darauf eingehen, weil der Verständnisprozess dem Lehrer offenbart wird. Auch die Sprachforschung profitiert, indem Forscher die Aussprache der Muttersprachler mit Leuten, die diese Sprache erlernen wollen, vergleichen und untersuchen. [46]

6.3 Ergebnisvisualisierung

Der Ablauf und Endzustand der Simulation können auf verschiedene Weisen dargestellt werden. So gibt es die Möglichkeit das deklarative und prozedurale Gedächtnis als Text darzustellen.

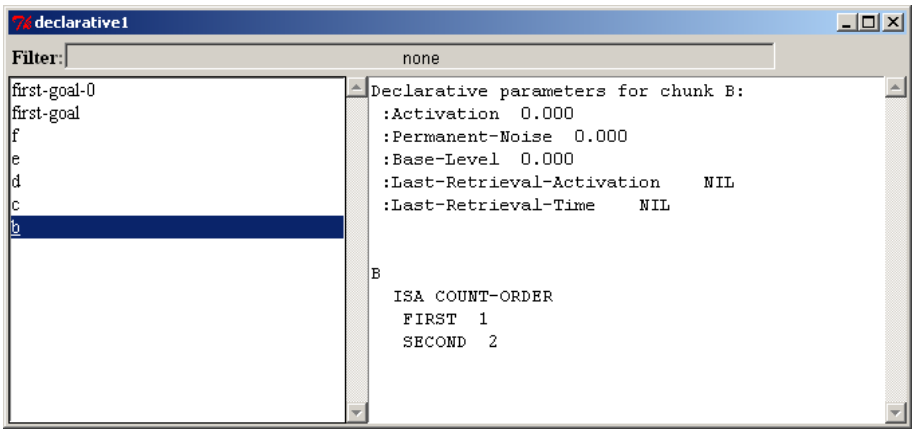

Abbildung 22: Darstellung der Chunks als Text. Links sind die einzelnen Chunks, rechts die Metriken und Inhalte des gewählten Chunks abgebildet.

So wird das deklarative Gedächtnis, also die einzelnen Chunks so dargestellt, wie sie in dem zu simulierendem Modell beschrieben werden. Die erzeugten Chunks erhalten Namen wie „b“, „c“, „d“ und so weiter. Links in der Liste kann man einen Chunk auswählen und im rechten Teil des Fensters werden die Einzelheiten zu dem gewählten Chunk angezeigt (Abbildung 22). Es wird nicht nur der Inhalt des Chunks angezeigt, sondern auch zusätzliche Daten, z. B. wann wurde das letzte Mal auf den Chunk zugegriffen.

Das prozedurale Gedächtnis wird im Laufe der Simulation nicht verändert, aber man kann einsehen zu welcher Zeit sie zum ersten Mal ausgeführt wurde. Links in der Liste kann man eine Produktion auswählen und rechts werden nähere Informationen über die Produktion angezeigt (Abbildung 23).

Des Weiteren ermöglicht die ACT-R-Oberfläche eine grafische Darstellung der aktiven Gehirnregionen beim Ausführen von Produktionen (Abbildung 24 und 25).

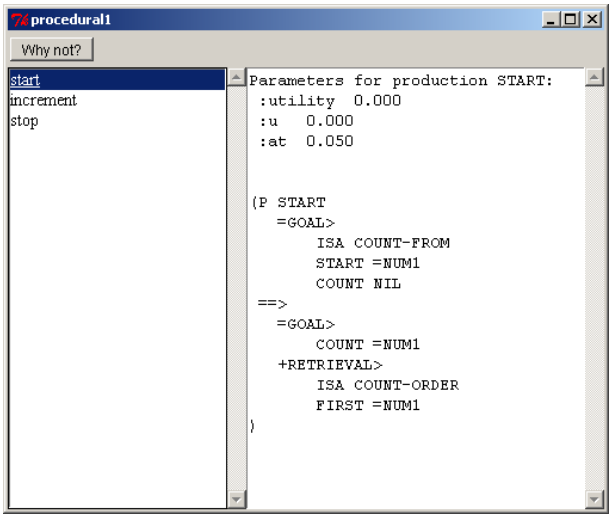

Abbildung 23: Darstellung des prozeduralen Gedächtnisses. Links sind die Produktionen, rechts Metriken und der Inhalte der gewählten Produktion.

Die Oberfläche für die 2D-Darstellung bietet 25 Felder mit jeweils einem Längsschnitt des Gehirns (Abbildung 24). Die aktiven Gehirnregionen werden durch farbige Kästchen repräsentiert. Je stärker eine Region angesprochen wird, desto heller wird das Kästchen. Wird ein Gehirnteil nicht angesprochen, dann ist das Kästchen schwarz. Mit dem Schieberegler kann man in der Simulation zeitlich vor und zurück gehen. Die Intensität der Aktivität von Gehirnregionen wird entsprechend dem gewählten Zeitpunkt dargestellt.

Die Oberfläche der dreidimensionalen Darstellung bietet die Möglichkeit die Gehirnaktivitäten anhand eines Drahtgittermodells des Gehirns darzustellen (Abbildung 25). Auch hier gibt es einen Schieberegler für die Zeitangabe und die Kästchen, die je nach Gehirnaktivität heller oder dunkler sein können. Je heller das Kästchen, desto aktiver ist die Gehirnregion. Den Betrachtungswinkel kann man mit der Maus verändern, indem der Benutzer auf das Bild klickt und die Maus bei gedrückter linken Maustaste verschiebt. So erhält man einen räumlichen Eindruck vom Gehirn.

Die Aktivität der Gehirnregionen kann auch als ein Liniendiagramm dargestellt werden. Es wird die Intensität der Aktivierung abhängig von der Zeit visualisiert (Abbildung 26). Im linken Teil des Fensters hat der Benutzer die Wahl zwischen einzelnen Gehirnregionen, für die das Diagramm erstellt werden soll.

Damit der Benutzer sich einen Überblick über die gesamte Gehirnaktivität verschaffen kann, hat er die Möglichkeit die Aktivität einzelner Gehirnregionen gesammelt in einem Diagramm einzusehen (Abbildung 27). Auf der Y-Achse sind die einzelnen Gehirnregionen aufgelistet und auf der X-Achse befindet sich die Zeit.

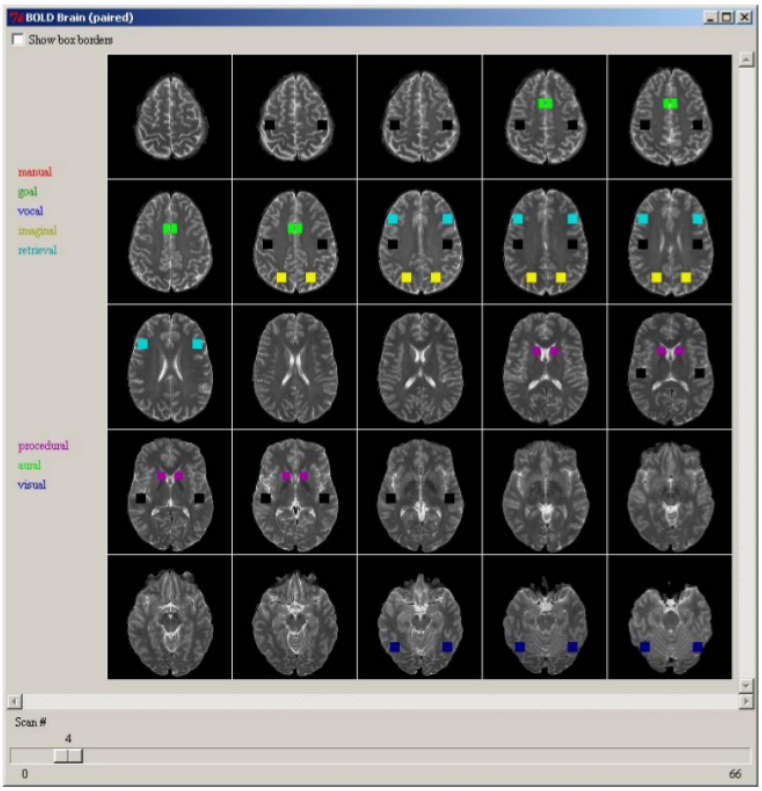

Abbildung 24: 2D-Darstellung des Gehirns in ACT-R als 25 Längsschnitte des Gehirns mit aktiven Regionen, die mit farbigen Kästchen gekennzeichnet sind [5].

6.4 Grenzen

Das Framework ACT-R kann zwar neues Faktenwissen erlernen, jedoch ist es nicht in der Lage sich über das vorgegebene prozedurale Wissen hinaus zu entwickeln. Es kann keine neuen Verhaltensmuster erlernen oder selbst entwickeln. Somit stößt es automatisch auf das im Kapitel 2.3.4 beschriebene „Modell-Problem“.

6.5 Inbetriebnahme

Um ACT-R auf einem PC zum Laufen zu bringen, benötigt man zuerst das Programm selbst. Es kann auf der offiziellen Homepage unter `http://act-r.psy.cmu.edu/actr6/` heruntergeladen werden. Der Kern der Anwendung besteht aus einer Lisp-Dateisammlung und einer vorkompillierten EXE-Datei. Die GUI kann optional hinzugeschaltet werden. Hier wird der „einfache“ Weg mit der Standalone-Version beschrieben. Die Standalone-Version bringt auch einen Lisp-Prozessor mit sich, was eine zusätzliche Installation von einem Lisp-Interpreter überflüssig macht. Nach dem Herunterladen und dem Entpacken des Archivs, wird die Anwendung mit „Start Environment.exe“ gestartet. Es erscheint die grafische Benutzeroberfläche, die das Bedienen des Programms ab sofort möglich

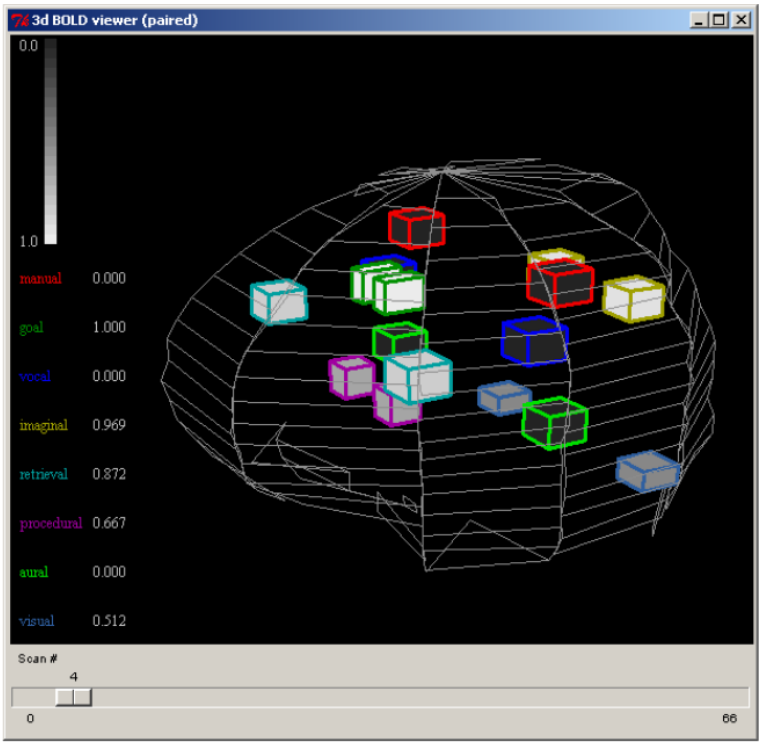

Abbildung 25: 3D-Darstellung des Gehirns in ACT-R als Drahtgittermodell mit farbigen Kästchen, die aktive Regionen darstellen [5]

macht.

Mit dem Knopf „Load Model“ kann man ein Lisp-Programm laden, was deklaratives und prozedurales Wissen enthält. Mit einem Befehl *(run 5)*, den man im Fenster „Listener“ eingeben muss, kann man die Simulation für maximal 5 Sekunden laufen lassen. Die Simulation stoppt früher, wenn keine Produktionen mehr feuern können.

Mit der Buttongruppe „Inspecting“ kann man Inhalte der einzelnen Buffern, des deklarativen und des prozeduralen Wissens anzeigen lassen. Mit den Buttons „2D Brain“ und „3D Brain“ werden Gehirndarstellungen angezeigt. Und mit dem Button „Horiz. Buffer Trace“ bekommt der Benutzer eine Übersicht über die Aktivitäten der Gehirnregionen. Alle Hilfsansichten bis auf „Declarative viewer“ und „Procedural viewer“ benötigen spezielle Lisp-Anweisungen im geladenen Modell. Sind diese Anweisungen im Modell nicht vorhanden, bleiben die Fenster leer. Die richtige Verwendung dieser Anweisungen ist im Handbuch der ACT-R-Bedienoberfläche [5] beschrieben. Um eine eigene Simulation mit ACT-R durchzuführen oder eine Simulation genau zu inspizieren, werden gute Kenntnisse der Sprache Lisp und der ACT-R-Architektur benötigt.

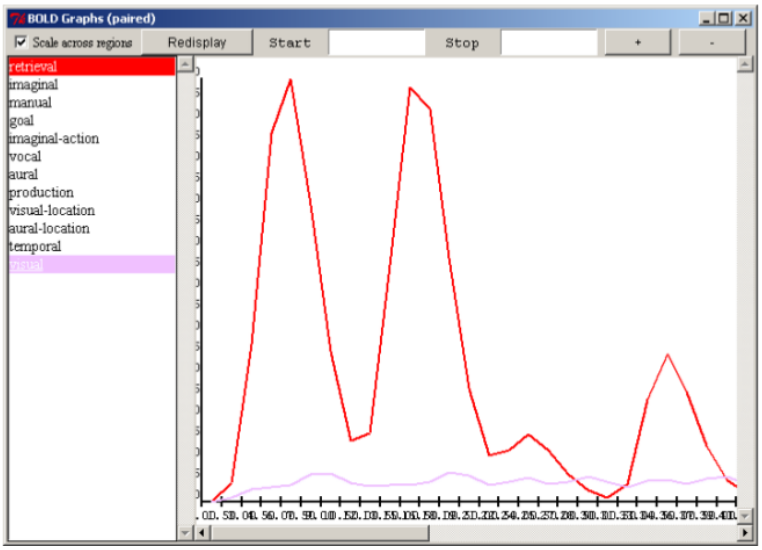

Abbildung 26: Diagramm für die Gehirnaktivität einzelner Gehirnregionen [5]. Man erkennt, dass die visuelle Region (als blaue Linie dargestellt), im Gegensatz zum Wissensabruf (als rote Linie dargestellt), kaum Aktivität zeigt.

6.6 Fazit

Hinter ACT-R steht ein einfacher Grundgedanke. Das Framework besteht grob aus Modulen und Buffern, deklarativem und prozeduralen Wissen. Auf dieser Basis wird mit Hilfe von komplexen Modulen die Gehirnsimulation ermöglicht. Dadurch, dass ACT-R Metriken über Chunks und Produktionen erstellt, kann man durch Vergleich der Simulationen die Validität des Frameworks und des programmierten Modells überprüfen. Die Ergebnisse der Simulationen werden auf verschiedenste Weisen dargestellt: Text, Diagramme, 2D-Gehirnansicht und 3D-Gehirnansicht. Jedoch ist es nicht möglich das Framework als Universalwerkzeug zu verwenden, weil es keine neuen Verhaltensweisen erlernen kann. Neues Wissen wird in Form von neuen Objekten abgespeichert. Die Inbetriebnahme ist relativ einfach, wenn man sich mit der Lisp-Sprache und der ACT-R-Architektur gut auskennt.

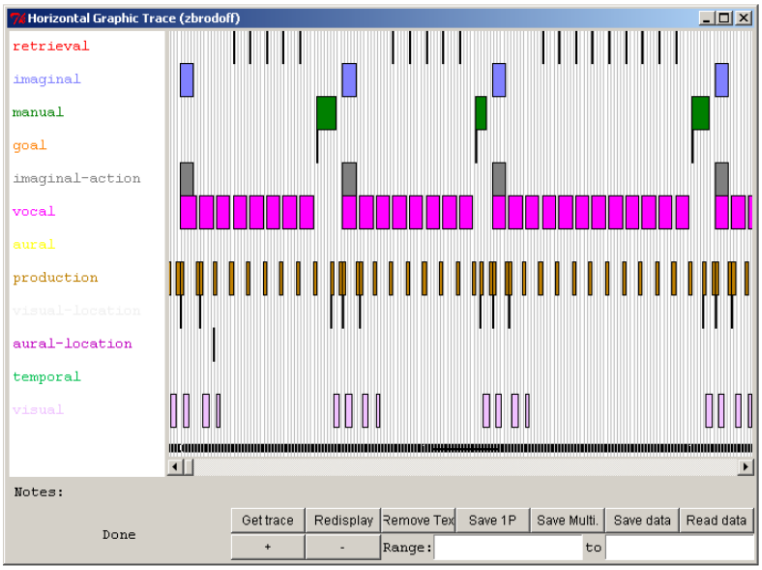

Abbildung 27: Diagramm für die Gehirnaktivität einzelner Gehirnregionen als Übersicht [5]. Die Gehirnregion, die für Sprache zuständig ist (als pinke Kästchen dargestellt), zeigt die meiste Aktivität. Dabei sieht man, dass die Produktionen ständig feuern (als braune Kästchen dargestellt). Dies deutet auf eine Simulation der Sprachausgabe.

7 Framework Apex

Apex ist ein Framework, welches zur Entwicklung von autonomen, intelligenten Agenten und deren Simulation dient. Es bietet ein Sprachframework zur Definition der Verhaltensweisen und Tools zur Überprüfung und Visualisierung der Vorgänge.

Das Framework wurde von der NASA Intelligent Systems Division am Ames Research Center entwickelt. Die Webseite[10] des Projektes wurde im Laufe der Erstellung dieses Dokumentes offline genommen. Da Apex jedoch ein Open-Source-Projekt ist, ist es noch auf der zum Projekt gehörende SourceForge Seite[11] zu finden. Veröffentlicht wurde das Framework unter dem „NASA Open Source Agreement“. Da das letzte Release im August 2006 stattfand, liegt die Vermutung nahe, dass die Weiterentwicklung eingestellt wurde.

7.1 Aufbau/Architektur

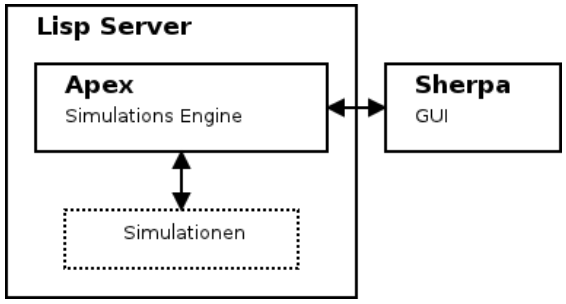

Abbildung 28: Apex Architektur: „Apex“ Simulationsumgebung in einer Lisp-Serverumgebung und die Benutzeroberfläche „Sherpa“.

Das Framework besteht aus zwei Teilprogrammen (Abbildung 28). Das Erste ist Apex selbst und beinhaltet die Simulations-Engine. Es läuft auf einem Lisp Server (Abbildung 29) und ist dem entsprechend in Lisp implementiert. Mit der Simulator-Engine enthält Apex auch einen Interpreter für die „Procedure Definition Language“ (PDL), mit welcher die Verhaltensweisen der Agenten modelliert werden. Diese ist eine ebenso auf Lisp basierende Schnittstelle zu Apex und bietet drei verschiedene Simulationsmodelle:

Native simulation applications sind vollständig in Apex eingebettet und verwenden die interne Simulationsengine. Dieser Anwendungstyp ist eine ereignisgesteuerte Simulation, wobei Ereignisse diskrete Aktionen sind. Hierbei können Agenten miteinander oder mit ihrer Umgebung interagieren. Die Simulation ist in keiner Weise an die Echtzeit gekoppelt, sondern hängt von der Komplexität der Anwendung und der Verarbeitungsgeschwindigkeit des Systems ab.

[10] http://ti.arc.nasa.gov/projects/apex/
[11] http://sourceforge.net/projects/apex-autonomy/

Real-time applications verwenden die interne Simulationsengine dagegen nicht. Agenten arbeiten in Echtzeit, welche durch den Systemtakt gemessen wird. Damit können Beispielsweise autonome Fahrzeuge gesteuert werden, welche ein Interface für Apex anbieten.

Foreign simulation applications dienen dazu, Apex mit externen Simulationsumgebungen zu verbinden, wodurch in Apex implementierte Agenten mit externen Simulationen interagieren können. Im letzten Stand der Software war dieser Anwendungstyp noch in der Entwicklung und wurde daher wie die „Real-time applications“ gehandhabt. Ein Beispiel für diesen Anwendungstyp ist die Interaktion mit dem Flugsimulator X-Plane. Dabei steuert ein Apex-Agent ein Flugzeug über die von X-Plane bereitgestellte API. Beispielanwendungen hierzu liegen Apex bei.

Sollten diese drei Grundtypen nicht ausreichen, können auch noch eigene Anwendungstypen definiert werden.

Innerhalb des Rahmens, der von den Schnittstellen der Simulationsumgebung vorgegebenen wird, lassen sich die Anwendungen frei programmieren. Sie sind also nicht auf Zuweisungsregeln der Art „IF A THEN B, C“ beschränkt.

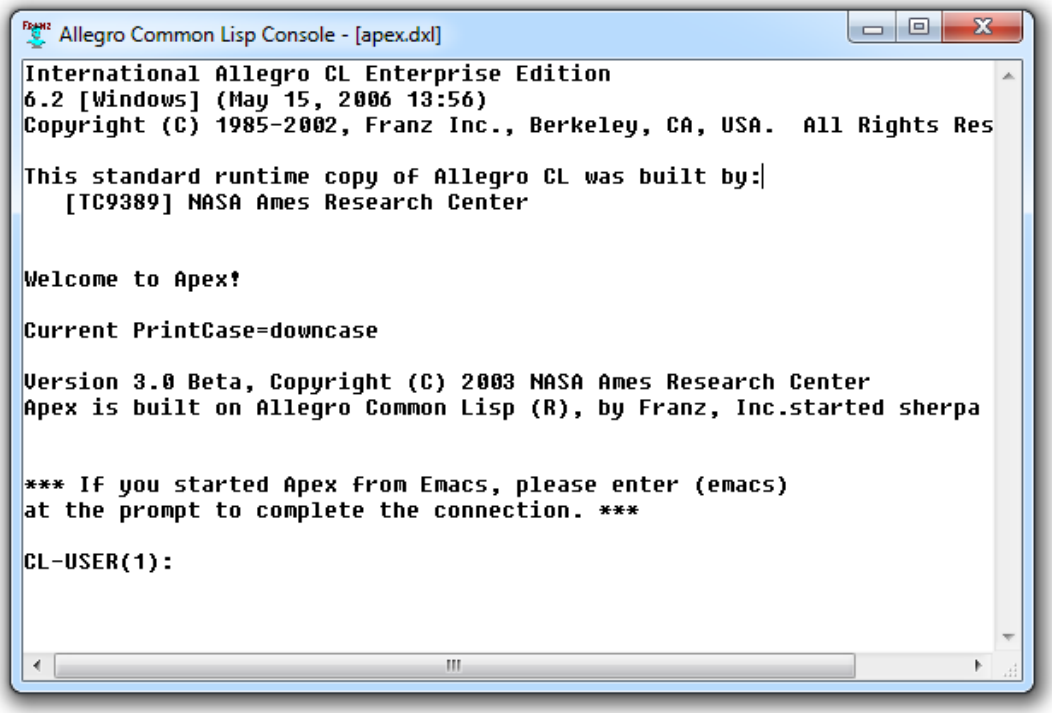

Abbildung 29: Lisp / Serverumgebung von Apex: Auf dieser werden die Simulationen durchgeführt.

Zur Steuerung der Simulationen wird Sherpa (Abbildung 30) verwendet. Dies ist ein auf Java basierender Client, welcher auch auf anderen Systemen gestartet werden kann,

als die Simulationsumgebung selbst. Mit diesem kann der Ablauf einer Simulation Schritt für Schritt überwacht werden. So können die simulierten Vorgänge analysiert und auf Fehler überprüft werden.

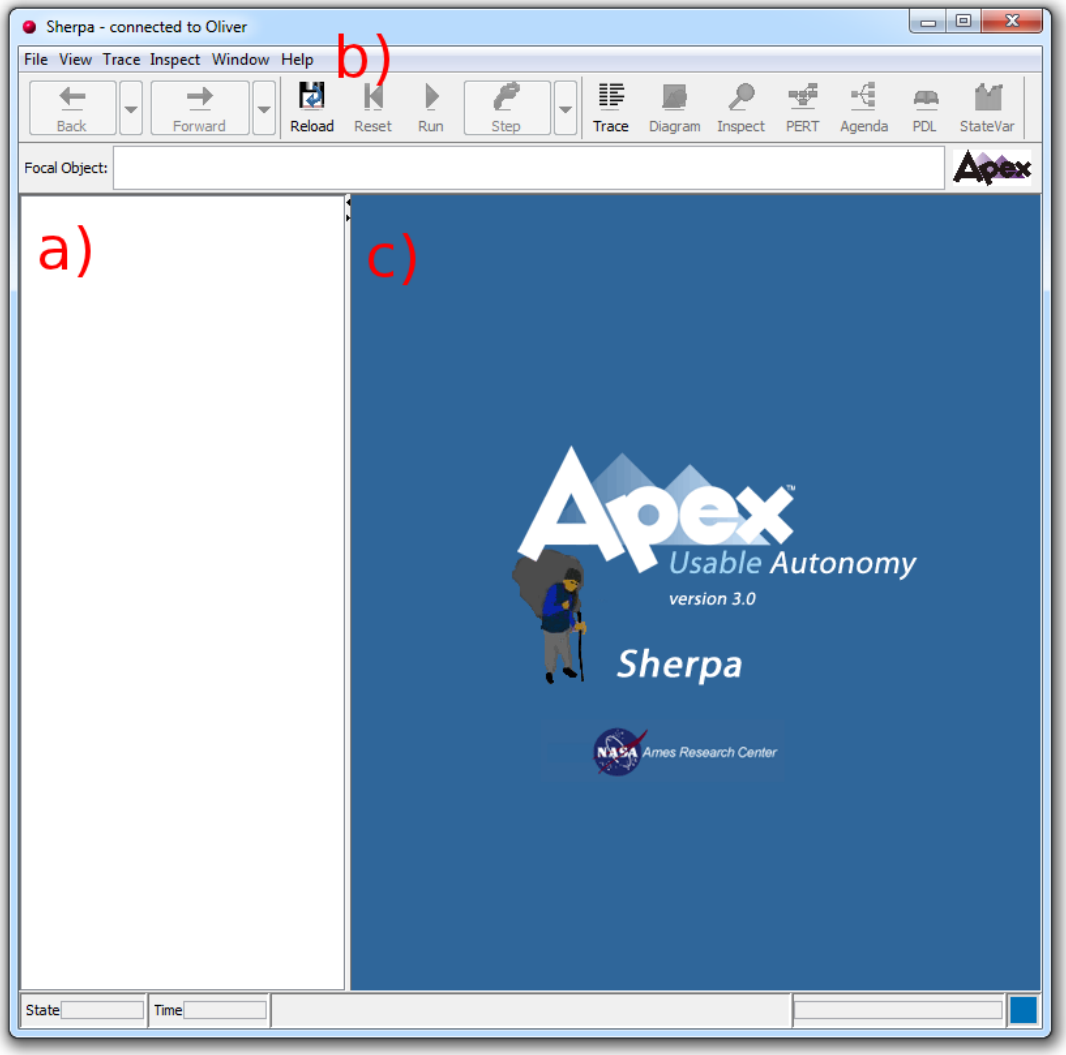

Abbildung 30: Sherpa, die zu Apex gehörende GUI, mit dem Bereich für den Objektbaum (a), der Werkzeugleiste für Simulationsablauf und Analyse (b), sowie dem Arbeitsbereich (c)

7.2 Einsatzgebiete

Die Apex Website listet unterschiedliche Anwendungsszenarien auf, für welche Apex bereits erfolgreich verwendet wurde:

- Bei der Flugsteuerung und dem Missionsmanagement eines autonomen Helikopters, der ein Gebiet überfliegt und beobachtet.

- Bei der Simulation des Mars Rovers, um Parameter der autonomen Steuerung zu untersuchen.

- Bei der Simulation von menschlichen Fluglotsen und Piloten, um festzustellen wie sich Änderungen in der Ausrüstung oder den Verfahren auswirken könnten. („Virtual Airspace Modeling and Simulation Project“ (VAMS) [26])

- Bei der Vorhersage von Dauer und Abfolge routinemäßiger menschlicher Verhaltensweisen. („Cognitive, Perceptual, and Motor - Goals, Operators, Methods, and Selection“ CPM-GOMS [25])

- An Universitäten zur Lehre der Bereiche „kognitive Architekturen“ und „Human-System Engineering“. HSE beschäftigt sich mit Charakteristiken von Menschen, bezogen auf Organisation, Sozialverhalten und Kognition in der Verbindung mit technischen Systemen [43].

- Als ein intelligentes Entscheidungsfindungsmodell unter einem Dialog-Management-System.

- In einem kollaborativen Entscheidungsfindungsexperiment. Als künstliches menschliches Subjekt.

- Als programmierbarer, menschlicher Agent in einer Luftraumsimulation.

- Bei der Modellierung und Untersuchung des Verhaltens von Astronauten bei der Entscheidungsfindung in kritischen Situationen, wie der Startphase eines Space Shuttle Fluges.

Für die Projekte zur Entscheidungsfindung sind jedoch leider keine weiteren Informationen hinterlegt, wie diese mit Apex durchgeführt wurde.

7.3 Ergebnisvisualisierung

Apex bietet Möglichkeiten, um die Aktionen der Agenten genau zu beobachten. Der Simulationsablauf selbst kann jedoch nur in Einzelschritten beobachtet werden, wenn diese durch die Anwendung unterstützt werden. Entsprechendes ist bei „Real-time applications“ nicht möglich, da diese zwingend auf den Zeitablauf reagieren können müssen. Im Folgenden werden die einzelnen Oberflächen, welche der Analyse dienen, aufgeführt:

Trace Ein Log mit Zeitangabe, wann welche Aktionen von welchem Agenten durchgeführt wurden. So kann die Reihenfolge der Abläufe und die Kommunikation zwischen den Agenten beobachtet werden (Abbildung 31).

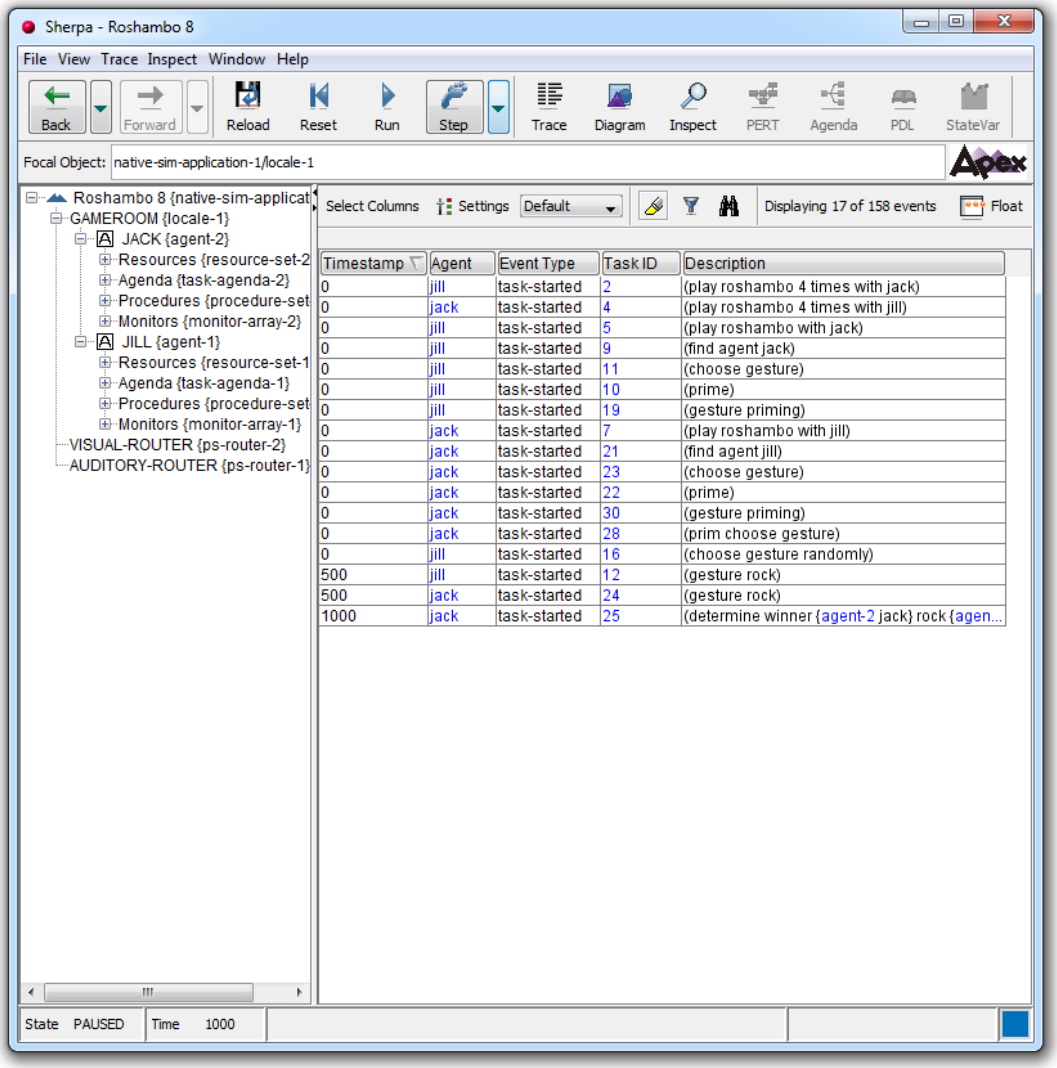

Abbildung 31: Apex, Trace-Ansicht, mit der Anzeige der von den Agenten durchgeführten Aktionen.

Diagramm Zeigt die nächste Ebene von Objekten unter dem momentan aktiven Objekt (Abbildung 32). Das Diagramm ist navigierbar und springt beim Auswählen von Knoten auf die nächste Ebene. Zu den Objekten in den verschiedenen Ebenen gehören die Simulationsumgebung, die Agenten und deren Eigenschaften.

PERT Zeigt ein PERT Chart bzw. eine Art Gantt Diagramm der bereits abgelaufenen Aktionen im zeitlichen Verlauf. Die Aktionen werden dabei auch in Unteraktionen und Abhängigkeiten eingeteilt (Abbildung 33).

Agenda Zeigt eine Liste der Aufgaben der Agenten und deren Status an. Also ob die

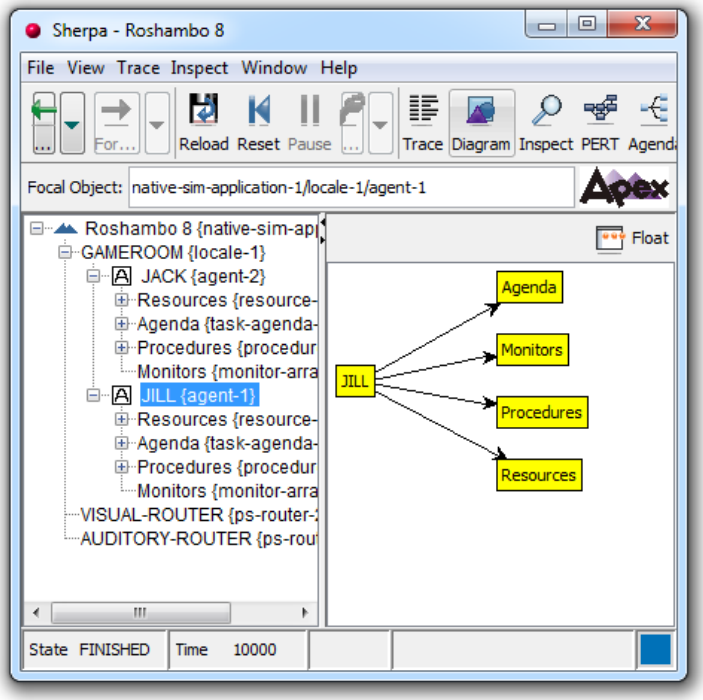

Abbildung 32: Apex, Diagramm-Ansicht des Objektbaums. Dabei werden die Objekte gezeigt, welche dem momentan aktiven Objekt untergeordnet sind.

Aufgaben bereits ausgeführt wurden, in Bearbeitung sind oder noch nicht gestartet wurden (Abbildung 34).

PDL Eine Auflistung der Prozeduren, die ein Agent beinhaltet. Es handelt sich um eine baumartige Darstellung des PDL Sourcecodes (Abbildung 35). Dabei wird die Aufrufhierarchie der Prozeduren dargestellt.

Inspect und Statevar Beide Ansichten zeigen verschiedene Informationen über die Agenten an, sowie die momentane Belegung diverser Statusvariablen.

7.4 Grenzen

Die Hauptaufgabe von Apex liegt in der Entwicklung und Simulation der Steuerung komplexer, autonomer Systeme. Daher ist die Fähigkeit von Apex, die menschliche Kognition zu simulieren, nur sehr eingeschränkt. Das Unterprojekt CPM-GOMS [25] geht in diese Richtung. Es wird benutzt, um die nach außen sichtbaren Handlungsweisen von Menschen und deren Dauer zu simulieren. Dies wird verwendet, um die Auswirkungen von Umgebungsänderungen auf den Menschen zu simulieren und wie sich diese auf den Arbeitsablauf auswirken. Ein Beispiel hierzu ist die Simulation von Änderungen in der Ausrüstung der Flugsicherung und die Auswirkung auf das Arbeitsverhalten der Flug-

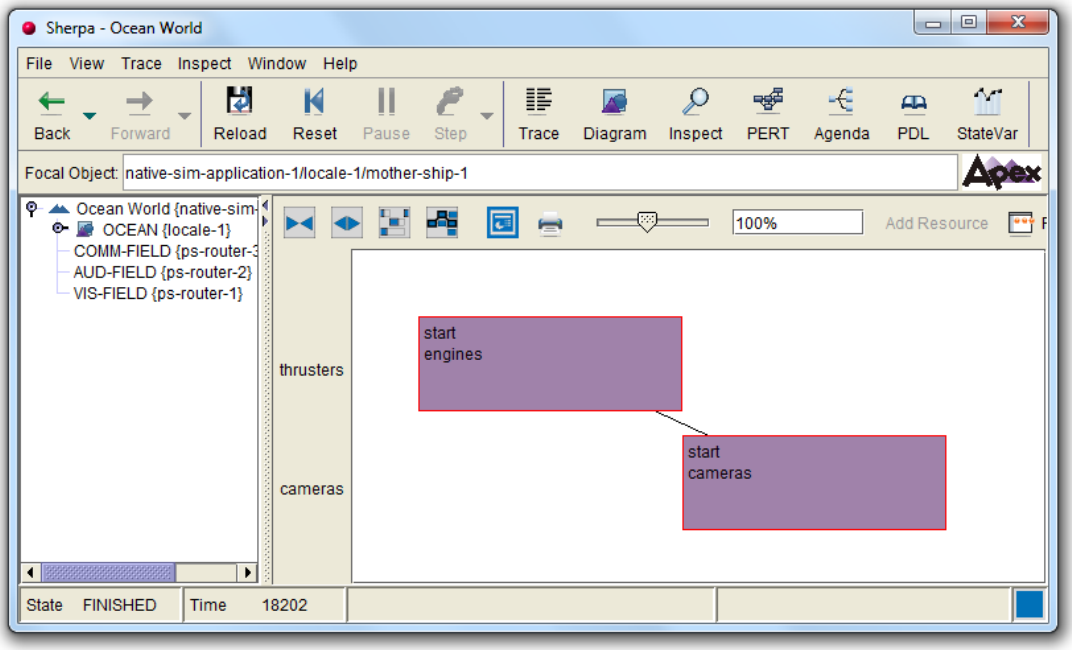

Abbildung 33: Apex, PERT Diagramm eines Agenten über die bereits ausgeführten Aktionen. Dabei werden auch die Abhängigkeiten durch Verbindungen zwischen den Aktionen gezeigt.

lotsen (Projekt VAMS [26]). Das Projekt kann jedoch nicht die internen Vorgänge beim menschlichen Denken simulieren.

7.5 Inbetriebnahme

Apex benötigt keine Installation. Unter Windows kann das Paket einfach entpackt und gestartet werden. Dabei muss zuerst die LISP Umgebung (Apex.exe) gestartet werden und erst danach die Benutzeroberfläche Sherpa, die sich dann mit der LISP Umgebung verbindet.

Anschließend kann eine Anwendung geladen werden. Apex liefert zur Einführung über den Anwendungsaufbau einige Beispiele mit. Der folgende PDL-Code (Listing 1) zeigt einen Ausschnitt aus einem Beispiel für Agenten, welche das Spiel „Schere-Stein-Papier“ spielen. „primitive“ Funktionen sind die grundlegenden Aktionen, welche ein Agent durchführen kann. Im Beispiel werden bei diesen auch immer „duration“ Angaben gemacht, um die Dauer der Aktion anzugeben. „procedure“ Funktionen können umfangreichere Abläufe beinhalten. Das „step“ Keyword dient zur Definition von Einzelschritte, welche parallel durchgeführt werden, sofern nicht durch „waitfor“ zwingende Sequenzen oder Vorbedingungen definiert wurden.

Für die Kommunikation zwischen den Agenten dienen „router“. An diesen registrieren sich die Agenten und versenden Ereignisse. Router sind in diesem Ausschnitt aber nicht

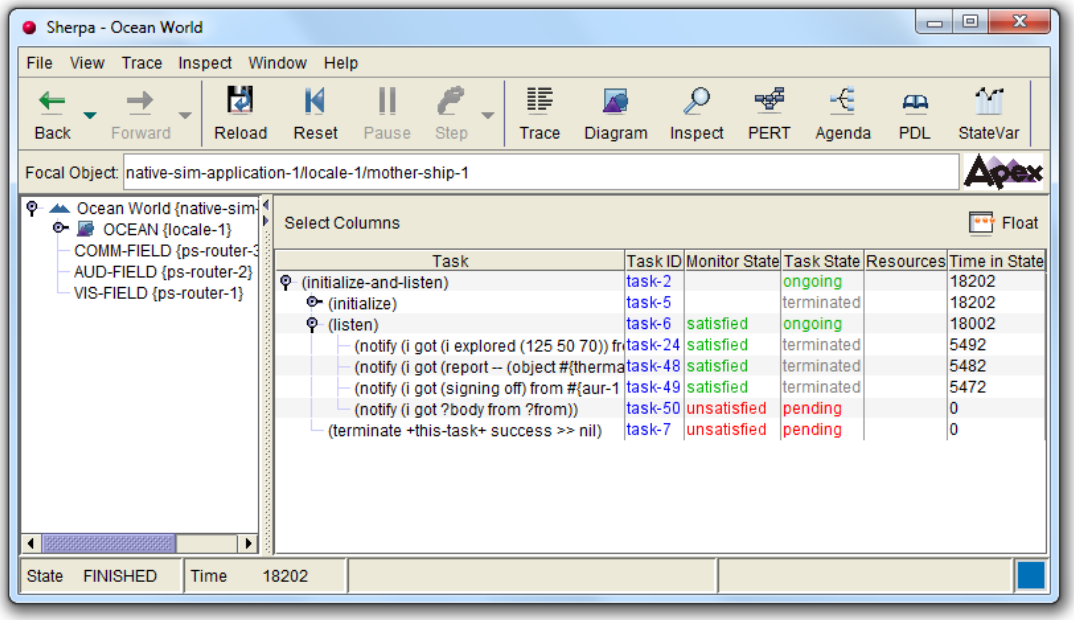

Abbildung 34: Apex, Agenda mit den Aufgaben eines Agenten. Dabei wird deren Bearbeitungsstatus gezeigt.

gezeigt.

Listing 1: Ausschnitt aus roshambo8.lisp, mit zwei Aktionen (primitive) für eine Zufällige oder gezielte Gestenauswahl und der Hauptprozedur (procedure), welche unter den Vorbedingungen eine der der beiden Aktionen durchführt.

```
(primitive (choose gesture randomly)
  (profile brain)
  (duration (500 ms))
  (return (random-elt '(rock paper scissors))))

(primitive (choose gesture by last ?last)
  (profile brain)
  (duration (500 ms))
  (return (ecase ?last (rock 'paper) (paper 'scissors)
                                     (scissors 'rock))))

(procedure (choose gesture)
  (step s1 (choose gesture randomly => ?gesture)
        (waitfor (:not (:measurement (game-gesture opponent = ?)
                                     :timestamp (> 0)))))
  (step s2 (choose gesture by last ?last => ?gesture)
```

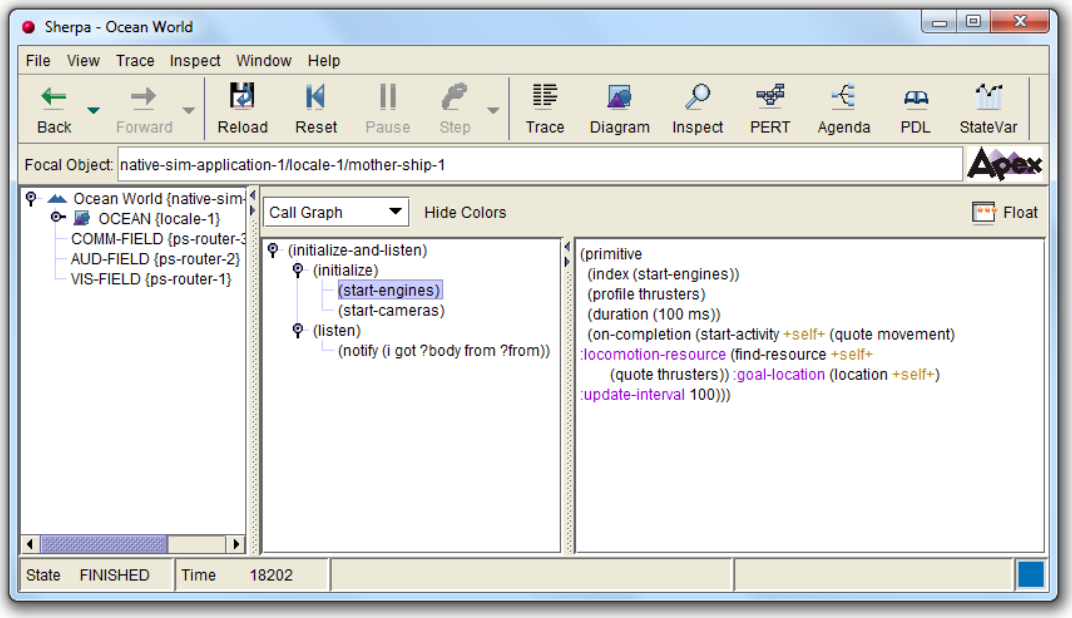

Abbildung 35: Apex, PDL Ansicht mit einer baumartigen Darstellung des PDL Sourcecodes.

```
        (waitfor (:measurement (game-gesture opponent = ?last)
                               :estimation (:persist))))
(step (terminate >> ?gesture)
        (waitfor (:or ?s1 ?s2))))
```

7.6 Fazit

Apex bietet ein Framework zur Entwicklung und Simulation von autonomen Agenten. Wie im Unterkapitel „Einsatzgebiete“ beschrieben umfasst es dabei weit mehr, als die autonome Steuerung von Robotern. Gerade durch erweiternde Unterprojekte, wie VAMS [26] und CPM-GOMS [25], wird es auch zur Verhaltensanalyse in definierten Umgebungen und der Lehre eingesetzt. Detaillierte, interne kognitive Vorgänge und deren Untersuchung sind dabei nicht das Ziel von Apex, wie dies dagegen bei den anderen untersuchten Frameworks das Ziel ist.

Die Bedienung von Apex lässt sich leicht erlernen und bietet eine Vielzahl an Möglichkeiten, um die Agenten in der Simulation zu untersuchen. Mit der Hilfe der mitgelieferten Beispiele und der Dokumentation lässt sich auch die Entwicklung der Agenten ebenso leicht erlernen und bietet mit PDL als Lisp-Dialekt eine gut lesbare Sprache.

8 Framework Soar

Soar wurde 1983 an der Carnegie Mellon Universität von John Laird, Allen Newell, und Paul Rosenbloom erstellt und wird fortwährend weiterentwickelt. Der Kern von Soar ist in C geschrieben und als BSD-Lizenz verfügbar. Es besteht die Möglichkeit für größere Projekte den Soar-Debugger zu benutzen, welcher in Java geschrieben ist. Dieser Debugger und der Soar-Kern befinden sich in der „Soar Suite“, welche mittlerweile (Stand: 03.06.2011) in der Version 9.3.0 verfügbar ist. Mit Hilfe einer Schnittstelle und Bibliotheken für C++, Java, C# und Tcl lässt sich Soar mit verschiedenen Programmiersprachen ansteuern und somit in beliebige Programme integrieren.

8.1 Aufbau/Architektur

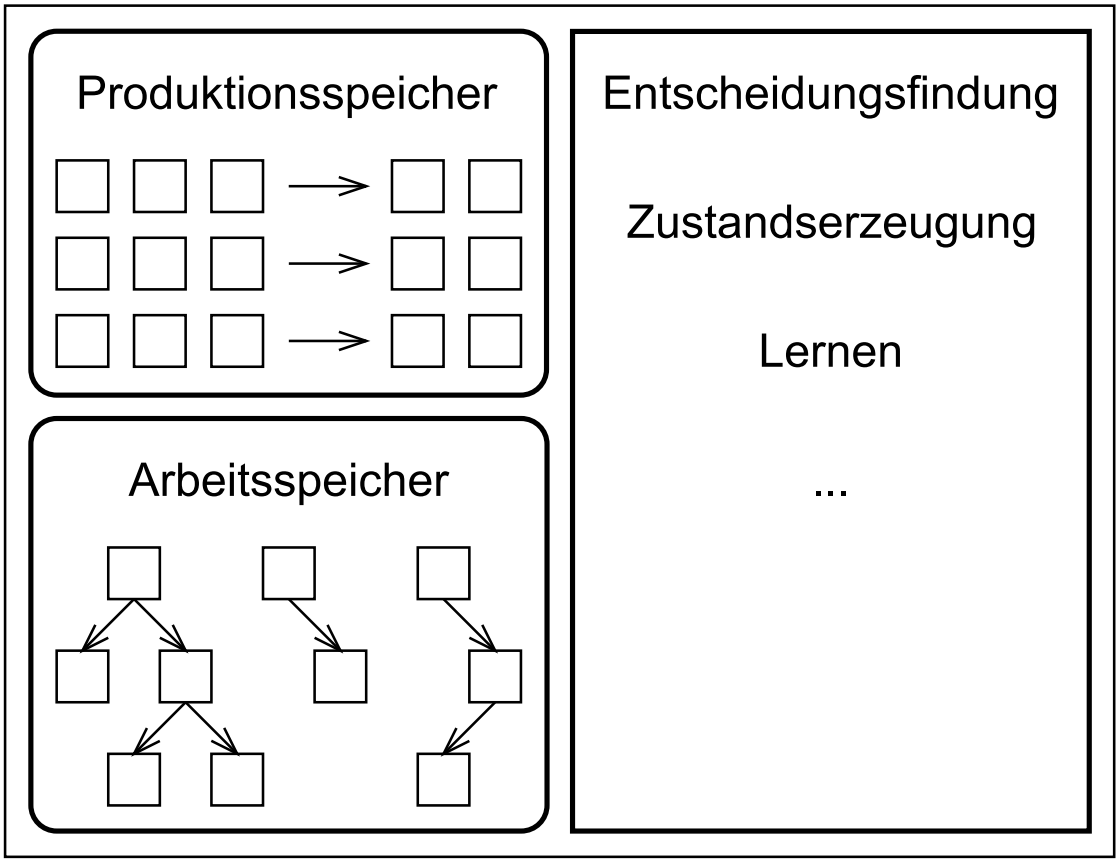

Abbildung 36: Die Architektur von Soar ist in 3 Teile gegliedert: Im Produktionsspeicher wird das dauerhafte Wissen in Form von Produktionen gespeichert, der Arbeitsspeicher beschreibt den aktuellen Zustand und im dritten Teil befinden sich die Algorithmen für Entscheidungsfindung, Zustandserzeugung, Lernen usw. [17]

.

Soar ist grob in drei Komponenten zu unterteilen. Diese sind einerseits Produktions- und Arbeitsspeicher, welche sich zur Wissensrepräsentation zusammenfassen lassen. Der andere Teil enthält die eigentliche Verarbeitungslogik, welche unter anderem die Entscheidungsfindung und den Lernprozess berechnet (Abbildung 36). Im Produktionsspeicher speichert Soar dauerhaftes Wissen in Form von sogenannten „Produktionsregeln“. Der Arbeitsspeicher enthält, wie der Name schon sagt, temporäres Wissen, welches für die aktuelle Verarbeitung benötigt wird in Form von sogenannten „Objekten“.

Produktionsregeln Die Produktionsregeln im Produktionsspeicher sind in Soar immer nach folgender Syntax aufgebaut:

Listing 2: Produktionsregel in Soar

```
sp {Name
    (Bedingung1)
    (Bedingung2)
    ...
    -->
    (Aktion1)
    (Aktion2)
    ...}
```

Eine Produktionsregel beginnt immer mit „sp“ („Soar production“), gefolgt von einer öffnenden geschweiften Klammer, in welcher sich der „Körper“ der Regel befindet. Dieser Körper besteht aus einem beliebigen Namen und einer oder mehreren Bedingungen, welche jeweils in runden Klammern stehen müssen. Außerdem folgt daraufhin ein „–>“, welches, wie in einer If-Bedingung, als die Trennung zum „then“-Teil gesehen werden kann. Nach einer oder mehreren Aktionen, welche wie die Bedingungen, ebenfalls in Klammern stehen müssen, wird die Produktionsregel mit einer geschlossenen geschweiften Klammer abgeschlossen. Allgemein kann also gesagt werden, dass Produktionsregeln die If-Bedingungen in Soar darstellen.

Objekte Ein Objekt ist in etwa gleichzusetzen mit einer instanziierten Klasse in objektorientierten Programmiersprachen. Ein Objekt besitzt demnach eine Menge von Eigenschaften, welche wiederum Objekte mit Eigenschaften sein können.

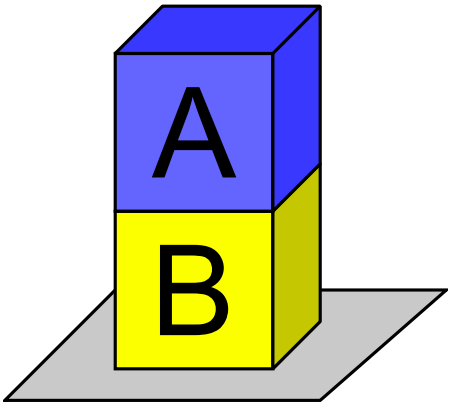

Abbildung 37: Zwei Holzblöcke (A und B) auf einem Tisch zur Verdeutlichung der Darstellung von Objekten in Soar [17]

Die in Abbildung 37 gezeigten Bausteine könnten in Soar als Objekte folgendermaßen beschrieben werden:

Listing 3: Objektbeschreibung der Bausteine in Soar

```
(s1 ^baustein b1 ^baustein b2 ^tisch t1)
(b1 ^farbe blau ^name A ^auf b2 ^typ baustein)
(b2 ^farbe gelb ^name B ^auf t1 ^typ baustein)
(t1 ^farbe grau ^name Tisch ^typ tisch)
```

Alle Bezeichnungen hierbei sind frei wählbar und sollten so gewählt werden, dass die Semantik leicht zu erkennen bleibt. „s1“ beschreibt z. B. einen Zustand, welcher die Bausteine „b1“, „b2“ und den Tisch „t1“ enthält. Der Baustein „b1“ hat dabei die Farbe „blau“, den Namen „A“, den Typ „baustein“ und liegt auf „b2“. Die Bezeichnung eines Parameters wird immer mit einem vorgestellten „^“ geschrieben. Der Wert des Parameters ist ein einfacher String, welcher durch weitere Beschreibung in einem zusätzlichen Tupel automatisch zu einem Objekt werden kann.

Diese Zusammenhänge lassen sich nun auch folgendermaßen in einem Graph darstellen:

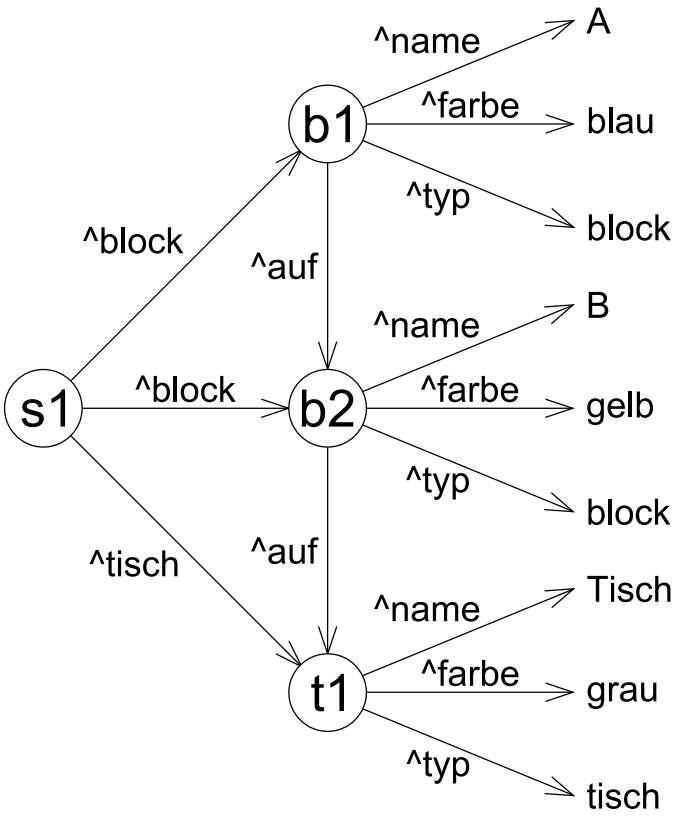

Abbildung 38: Beschreibung des Zustands s1, welcher in Abbildung 37 zu sehen ist, in Form eines Graphs [17]

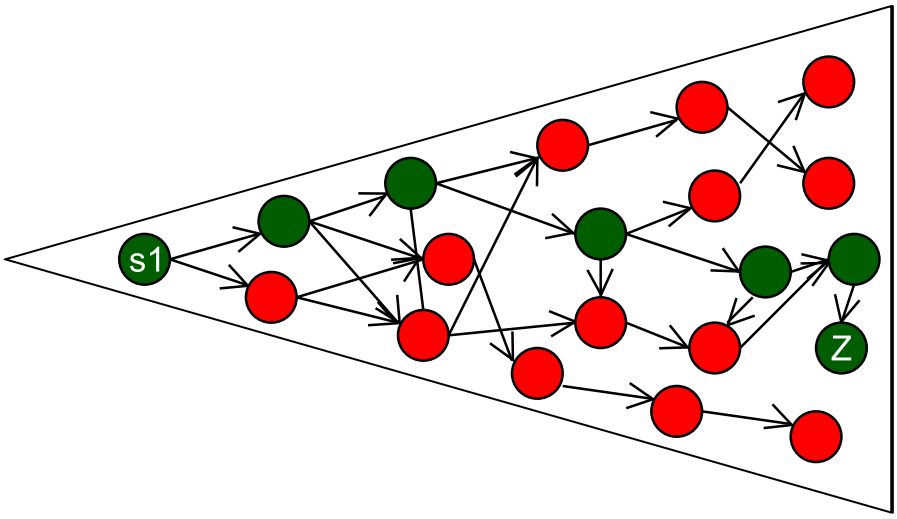

Abbildung 39: Alle Zustände, in die vom Startzustand aus gelangt werden kann, spannen einen Problemraum auf. Die Darstellung als Dreieck verdeutlicht, dass sich die Zahl der möglichen Zustände in jedem Schritt vergrößern kann. Die grünen Zustände zeigen den Lösungsweg von s1 nach Z.

Soar kann nun mit Hilfe der Produktionsregeln und Objekte theoretisch alle Probleme lösen (siehe General Problem Solver 3.3) bei denen von einem definierten Anfangszustand zu einem definierten Endzustand gelangt werden kann (Abbildung 39). Dazu wendet es entsprechende Produktionsregeln als einen Operator auf den aktuellen Zustand an, woraus durch die Manipulation in der jeweiligen Produktionsregel ein anderer Zustand resultiert. Dies wiederholt Soar so lang, bis es den vorgegebenen Endzustand erreicht hat. Während dieses Vorgehens gibt es meist mehrere mögliche Operatoren zwischen denen Soar entscheiden muss. Dazu durchläuft Soar einen sich ständig wiederholenden Zyklus, der entscheidet, welcher Operator in dem aktuellen Zustand der beste ist (Abbildung 40).

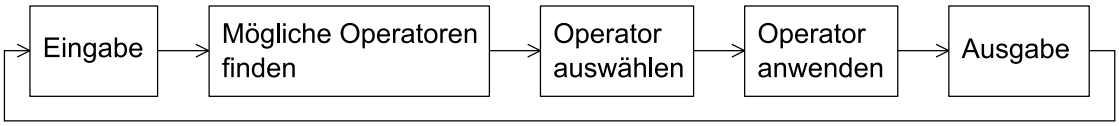

Abbildung 40: Der Ablaufzyklus in Soar. Nachdem Eingaben von Außen gesammelt wurden, werden alle Operatoren gesucht, deren Bedingungen mit dem aktuellen Zustand übereinstimmen. Aufgrund von Präferenzen wird daraufhin entschieden, welcher Operator ausgeführt wird. Danach wird der entsprechende Operator angewandt und der daraus resultierende Zustand ausgegeben.

Beim ersten Durchlauf wird, sofern der Programmierer keine Präferenzen vergeben hat, rein zufällig entschieden, welcher Operator ausgeführt wird. Gelangt Soar in eine Sackgasse, also in einem Zustand aus dem es nicht mehr entweichen kann, wird zu dem

letzten Zustand zurückgesprungen und durch Anwendung anderer Operatoren die Sackgasse umgangen. Jedes Mal, wenn dies erfolgreich war, speichert Soar diesen Zwischenweg in so genannten „Chunks“ mit höherer Präferenz ab. Wiederholt sich diese Situation in einem weiteren Durchlauf, so „feuert“ der zuvor generierte Chunk sofort und sorgt dafür, dass Soar kein zweites Mal in diese Sackgasse läuft. Auf diese Weise findet in Soar ein Lernvorgang statt.

8.2 Einsatzgebiete

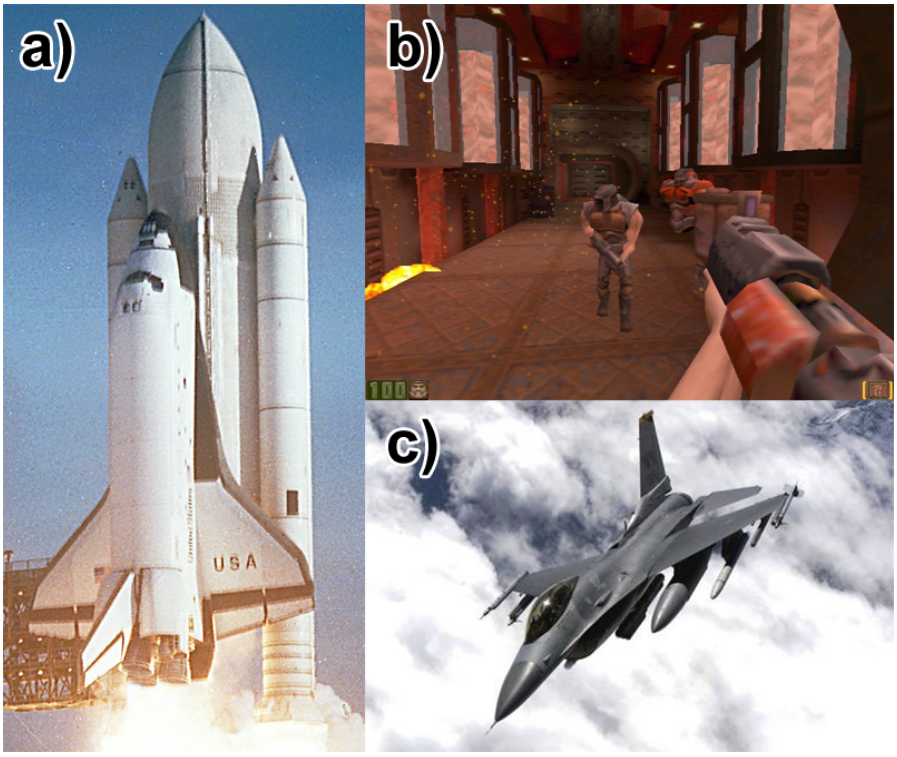

Abbildung 41: Die Einsatzgebiete von Soar sind vielseitig: a) Raumfahrt [31], b) Computerspiele [29] und c) Militär [1]

Soar findet in vielen Bereichen Anwendung: von der Psychologie, über Roboterkontrollarchitekturen und dem Militärbereich bis hin zu Computerspielen. So wird Soar z. B. in dem bekannten Ego-Shooter „Quake II“ für die Steuerung der Bots benutzt [10]. Sehr umfangreichen Einsatz findet die Architektur jedoch in militärischen Projekten. Laut [17] sind die beiden Programme „TacAir-Soar“ und „RWA-Soar“ daraufhin programmiert, menschliche Piloten zu ersetzen und autonome Flugmissionen inklusive Einhaltung der Kommandostruktur zu absolvieren. Ein weiteres Projekt namens „Soar Moutbot“ simuliert individuelle menschliche Gegner in Kriegsgebieten und hat dadurch einige besondere Fähigkeiten, wie z. B. reaktive Aktionen und die Kommunikation mit anderen „Soar Moutbots“. Auch die NASA nutzt Soar in dem Projekt „NTD-Soar“, welches Space-Shuttles vor dem Start testet und mit dem Raketenstartteam kommuniziert

[17].

8.3 Ergebnisvisualisierung

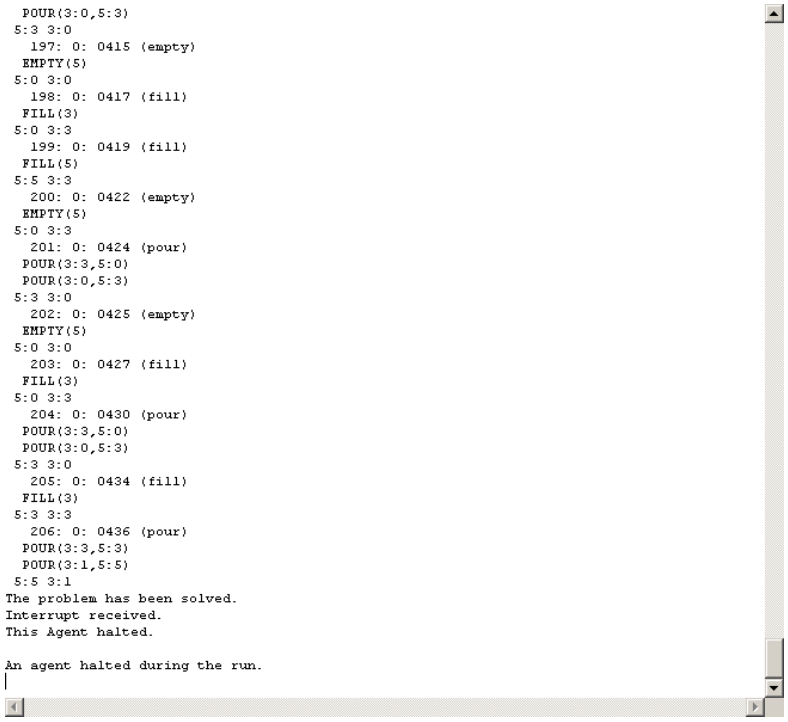

Abbildung 42: Die textuelle Ausgabe von Soar Debugger. In diesem Beispiel wurde die mitgelieferte Demo „water-jug“ ausgeführt. Die Ausgabe beschreibt, welcher Krug in welchen gefüllt (fill), bzw. geleert (empty) wird.

Das mit Soar ausgelieferte Tool „Soar Debugger“ bietet die Möglichkeit, in Soar geschriebene Programme auszuführen und währenddessen zu überwachen. So gibt es eine textuelle Ausgabe (Abbildung 42), welche einfache Textausgaben, die in dem jeweiligen Soar-Programm an einzelnen Stellen vermerkt sind, anzeigen kann.

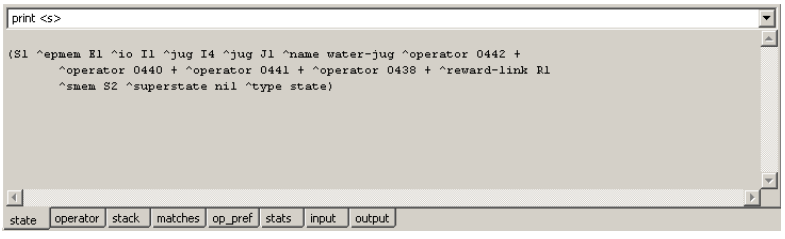

Abbildung 43: Die Anzeige des aktuellen Zustands nach Ausführung der „water-jug“-Demo. Unten befinden sich verschiedene Tabs, von denen in dieser Abbildung „state“ ausgewählt ist. Dadurch beschreibt der obere Teil des Fensters den aktuellen Zustand des Soar-Programms.

Außerdem bietet der Soar-Debugger die Möglichkeit, zusätzliche Eigenschaften während des Programmablaufs abzurufen. So kann man sich beispielsweise den aktuellen Zustand, die angewandten Operatoren sowie den Ausführungsstack oder statistische Informationen anzeigen lassen (Abbildung 43).

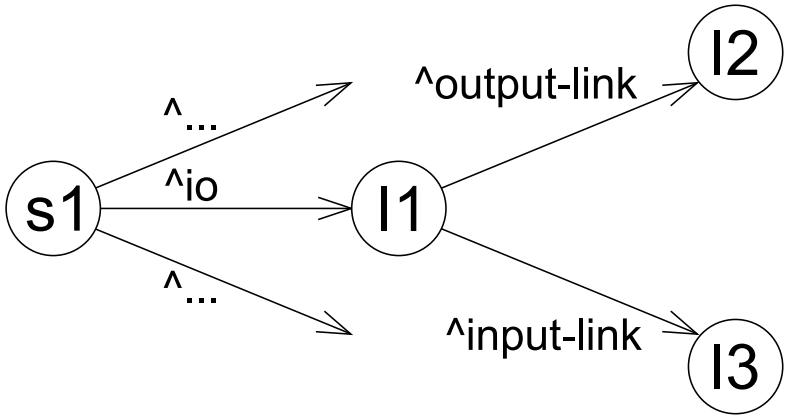

Abbildung 44: Jeder Zustand hat automatisch über das Attribute „ìo“ ein Objekt (I1), welches durch zwei weitere Objekte mit Ein- (I3) und Ausgabedaten (I2) versehen ist.

Zu den selbst einprogrammierten Objekten, vergibt Soar zusätzlich jedem Zustand, ein weiteres Objekt. Dieses Objekt hat die beiden Attribute „output-link“ und „input-link“, welche für die Ein- und Ausgabe zu anderen Programmen verantwortlich sind (Abbildung 44). Durch diese Schnittstelle lassen sich beliebige Oberflächen programmieren, welche auf Soar zugreifen und die Ergebnisse visualisieren können. Das in der Soar Suite mitgelieferte Demo-Programm „Eaters“ stellt ein solches komplexeres Beispiel mit grafischer Oberfläche dar (Abbildung 45).

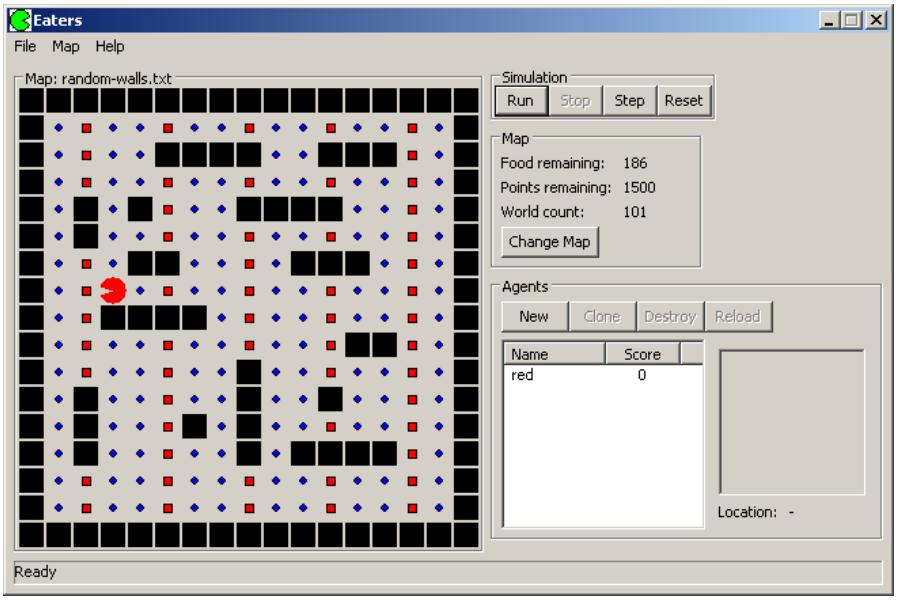

Abbildung 45: Demo-Programm Eaters: Die Figuren müssen in möglichst kurzer Zeit das Futter der gesamten Map auffressen.

8.4 Grenzen

Soar verwendet zur Problemlösung Chunks und vorgegebene Produktionsregeln, mit welchen es bei einer Aufgabenstellung im Prinzip alle Möglichkeiten durchprobiert. Die Frage ist, ob man dies als Intelligent bezeichnen kann? Soar lernt zwar nach jedem Durchlauf hinzu und optimiert die durchzuprobierenden Wege. Ob es dabei jedoch „weiß“ was es tut sei zu bezweifeln. Neue Aufgaben können ebenfalls nicht von Soar selbst erlernt werden, sondern müssen mit Hilfe der Produktionsregeln zuerst einprogrammiert werden.

Das Durchprobieren aller Möglichkeiten erkennt man gut an der in der Soar Suite mitgelieferten Demo „Missionaries and Cannibals“. In dem Beispiel gilt es für drei Missionare und drei Kannibalen, welche auf der einen Seite eines Flusses stehen, an das Ufer auf der anderen Seite zu gelangen. Dazu gibt es ein Boot, welches maximal zwei Personen transportieren kann. Die Schwierigkeit ist nun, dass an jeder Stelle (die beiden Ufer und das Boot) jeweils mindestens gleich viele Missionare wie Kannibalen sein müssen. Zur Lösung des Problems führt Soar mit jedem Klick auf den Button „Step“ eine scheinbar beliebige Aktion durch. So kann es vorkommen, dass sogar mehrmals dieselbe Aktion ausgeführt und wieder rückgängig gemacht wird. Da der Quellcode des Java-Programms den Autoren nicht zugänglich war und es mehrere Soar-Implementierungen des „Missionaries and Cannibals“-Problems gibt (im Installationsverzeichnis von Soar: share/soar/Demos/mac/*.soar), welche teilweise die Lernfähigkeit in Soar deaktivieren (leran –off), wurde dort wahrscheinlich solch eine Implementierung eingebunden. Somit kann es vorkommen, dass das Programm statt der mindestens elf, 100 oder noch mehr Schritte bis zum Ziel benötigt. Mit dieser Einstellung macht Soar nicht den Eindruck von Intelligenz, sondern von reinem „trial and error“.

8.5 Inbetriebnahme

Soar kann unter http://sitemaker.umich.edu/soar/home heruntergeladen werden. Aufgrund der Plattformunabhängigkeit durch Java läuft Soar sowohl unter Windows, OS X als auch Linux. Eine Installation ist nicht nötig. Es genügt, die heruntergeladene Datei in einen Ordner zu entpacken und z. B. das Programm „SoarJavaDebugger.jar“ im Ordner „bin“ zu starten. Im Hauptverzeichnis befinden sich einige Demo-Programme. Das folgende Beispiel initialisiert die Zustände des bekannten Problems der Türme von Hanoi und befindet sich ebenfalls in dem Ordner:

Listing 4: Initialisierung der Zustände für die Türme von Hanoi in Soar

```
sp {towers-of-hanoi*apply*initialize
    (state <s> ^operator.name initialize)
    -->
    (<s> ^disk <disk-1> <disk-2> <disk-3> <disk-4>
        <disk-5> <disk-6> <disk-7> <disk-8>
        <disk-9> <disk-10> <disk-11>
         ^peg <peg-a> <peg-b> <peg-c>
         ^holds <h1> <h2> <h3> <h4> <h5> <h6> <h7>
         <h8> <h9> <h10> <h11>
         ^last-disk1-peg <peg-b>
         ^last-disk-moved <disk-2>)
    (<h1> ^disk <disk-1> ^above <disk-2> ^on <peg-a>)
    (<h2> ^disk <disk-2> ^above <disk-3> ^on <peg-a>)
    (<h3> ^disk <disk-3> ^above <disk-4> ^on <peg-a>)
    (<h4> ^disk <disk-4> ^above <disk-5> ^on <peg-a>)
    (<h5> ^disk <disk-5> ^above <disk-6> ^on <peg-a>)
    (<h6> ^disk <disk-6> ^above <disk-7> ^on <peg-a>)
    (<h7> ^disk <disk-7> ^above <disk-8> ^on <peg-a>)
    (<h8> ^disk <disk-8> ^above <disk-9> ^on <peg-a>)
    (<h9> ^disk <disk-9> ^above <disk-10> ^on <peg-a>)
    (<h10> ^disk <disk-10> ^above <disk-11> ^on
        <peg-a>)
    (<h11> ^disk <disk-11> ^above none ^on <peg-a>)
    (<disk-1> ^name 1 ^size 1)
    (<disk-2> ^name 2 ^size 2)
    (<disk-3> ^name 3 ^size 3)
    (<disk-4> ^name 4 ^size 4)
    (<disk-5> ^name 5 ^size 5)
    (<disk-6> ^name 6 ^size 6)
    (<disk-7> ^name 7 ^size 7)
    (<disk-8> ^name 8 ^size 8)
    (<disk-9> ^name 9 ^size 9)
    (<disk-10> ^name 10 ^size 10)
```

```
(<disk-11> ^name 11 ^size 11)
(<peg-a> ^name |A|)
(<peg-b> ^name |B|)
(<peg-c> ^name |C|)
(<s> ^desired <d1> <d2> <d3> <d4> <d5> <d6> <d7>
    <d8> <d9> <d10> <d11>)
(<d1> ^disk <disk-1> ^above <disk-2> ^on <peg-c>)
(<d2> ^disk <disk-2> ^above <disk-3> ^on <peg-c>)
(<d3> ^disk <disk-3> ^above <disk-4> ^on <peg-c>)
(<d4> ^disk <disk-4> ^above <disk-5> ^on <peg-c>)
(<d5> ^disk <disk-5> ^above <disk-6> ^on <peg-c>)
(<d6> ^disk <disk-6> ^above <disk-7> ^on <peg-c>)
(<d7> ^disk <disk-7> ^above <disk-8> ^on <peg-c>)
(<d8> ^disk <disk-8> ^above <disk-9> ^on <peg-c>)
(<d9> ^disk <disk-9> ^above <disk-10> ^on <peg-c>)
(<d10> ^disk <disk-10> ^above <disk-11> ^on
    <peg-c>)
(<d11> ^disk <disk-11> ^above none ^on <peg-c>)}
```

In dieser Form wäre das Programm nicht ausführbar, bzw. würde in eine Endlosschleife geraten. Da nur eine Produktionsvorschrift existiert und kein Endzustand definiert wurde, wird die Produktionsregel nach jedem Schritt ausgeführt ohne eine Änderung der aktuellen Zustände. Damit das Programm anhält, müsste am Ende der Befehl „(halt)“ aufgerufen werden.

8.6 Fazit

Der Ansatz der Aufteilung in Produktions- und Arbeitsspeicher sowie einer Logikeinheit erscheint zunächst sinnvoll und dem menschlichen Gehirn ähnlich (siehe Kapitel 4). Das anfänglich unkoordinierte Herumprobieren, um ein Problem zu lösen, mag eventuell auch einem Kleinkind ähnlich sein. Mit deaktivierter Lernfähigkeit erscheint Soar absolut nicht mehr intelligent. Schaltet man diese Funktion jedoch ein, ergeben sich schon nach kurzer Zeit erstaunlich kurze Wege durch den entsprechenden Problemraum.

Das Schreiben eines Soar-Programms ist relativ kompliziert, da ausschließlich Produktionsregeln erstellt werden und somit der Überblick über die eigentliche Funktion der einzelnen Zeilen fehlt und sie somit nicht sehr intuitiv sind. Der Code erinnert an ein Assembler-Programm. Um mit Soar überhaupt umgehen zu kennen, ist ein sehr fundiertes Wissen über die Architektur und Arbeitsweise nötig.

9 Vergleich und Bewertung der Frameworks

In diesem Kapitel werden die Frameworks, ihre Ideen sowie Schwächen und Stärken miteinander verglichen.

Auch wenn Kognitionsframeworks das gleiche Ziel verfolgen, nämlich die menschliche Kognition zu simulieren, sind ihre Einsatzgebiete unterschiedlich (Abbildung 46). So bietet ACT-R beste Voraussetzungen für die Forschung der Laufzeitmessungen von Aktivitäten des Gehirns. Apex vertritt menschliche Steuerung von autonomen Vehikeln und die Soar-Autoren versuchen das ursprüngliche Ziel des General Problem Solvers zu erreichen.

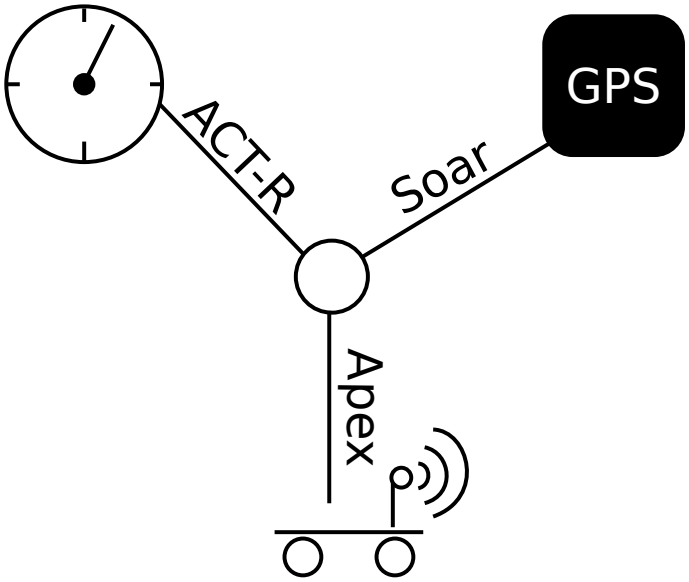

Abbildung 46: ACT-R bietet viele Visualisierungsmöglichkeiten, was für Gehirnforschung besonders wichtig ist. Apex wird für Robotersteuerung vewendet und Soar ist ein Nachfolger des GPS.

ACT-R ist ein Modulbasiertes Framework, was symbolischen und subsymbolischen Ansatz miteinander vereint. In dieser Arbeit wird er als hybrid gewertet, auch wenn es keinen echten konnektionistischen Ansatz implementiert. Dieses Framework lässt sich einfach mit einem Doppelklick starten und verfügt über eine GUI, die die Bedienung des Programms erheblich erleichtert. Die GUI macht es auch möglich, simulierte Vorgänge im menschlichen Gehirn, grafisch darzustellen. ACT-R verfügt über ein breites Spektrum an Einsatzgebieten.

Dennoch gibt es einige Schwächen. So war es erst nach langer Recherche möglich, eine der mitgelieferten Simulationen zu bedienen. Es zeigt, dass ACT-R aufgrund seiner hohen Komplexität für Anfänger nicht geeignet ist. Ein weiterer Punkt ist die Lernbarkeit des Frameworks. ACT-R kann die chunkbasierte, aber nicht die prozeduralbaiserte

Wissensbasis erweitern.

Apex ist ein Framework, welches eine Plattform für die Entwicklung von autonomen Systemen bietet. Damit besteht, im Gegensatz zu ACT-R oder Soar, seine Hauptaufgabe nicht in der Simulation von Gehirnfunktionen. Es bietet jedoch ein breites Einsatzspektrum in der Autonomie. Des Weiteren erschließen sich mit auf Apex arbeitenden Projekten weitere Einsatzbereiche. Beispielsweise befasst sich ein Projekt mit der Simulation von menschlichen Verhaltensweisen, um in bestimmten Einsatzbereichen Änderungen in der Umgebung auf deren Auswirkungen zu analysieren.

Die Bedienung von Apex lässt sich leicht anhand der mitgelieferten Beispiele erlernen. Dies gilt ebenso für die Sprache, in der die Simulationen entwickelt werden.

Soar verwendet wie ACT-R einen symbolischen Ansatz zur Wissensrepräsentation und -verarbeitung. Es arbeitet ebenso mit anfangs fest einprogrammierten Regeln, welche sowohl in ACT-R, als auch in Soar als „Produktionen“ bezeichnet und ähnlich verwendet werden.

Der Begriff „Chunk“ hat in ACT-R und Soar eine völlig unterschiedliche Bedeutung. In Soar sind Chunks Produktionsregeln. Diese werden durch Lernvorgänge automatisch generiert und bei jedem Problem zuvor anzuwenden versucht.

Der bei Soar mitgelieferte Debugger in Form einer grafischen Oberfläche lässt auch dieses Framework leicht bedienen und bietet somit für den Einsteiger beste Voraussetzungen. Die vielen verschiedenen Einsatzgebiete zeigen, dass Soar sehr flexibel und für alle möglichen Bereiche anwendbar ist.

Tabellenrepresentation Tabelle 2 beschreibt nochmals die wichtigsten Eigenschaften der untersuchten Frameworks.

Framework	ACT-R	Apex	Soar
Lizenz	LGPL	NASA Open Source Agreement	BSD
Sprache	Lisp	Lisp	Java/C
Einsatzgebiete	Forschung, Robotik, Sprache	Autonome Agenten, Luft- & Raumfahrt, Forschung	Raumfahrt, Computerspiele, Militär
Ansatz	symbolisch, subsymbolisch	symbolisch	symbolisch
Autoren	J. R. Anderson	NASA Ames Research Center	J. Laird, A. Newell, P. Rosenbloom
Ergebnisvisualisierung	Text, Diagramme, Bilder	Text, Diagramme	Text
Webseite	http://act-r.psy.cmu.edu/actr6	http://ti.arc.nasa.gov/projects/apex http://sourceforge.net/projects/apex-autonomy	http://sitemaker.umich.edu/soar

Tabelle 2: Vergleich der untersuchten Frameworks

10 Fazit

> Ever since the first computers, there have always been ghosts in the machine.
>
> Alfred Lanning, Film „I, Robot“

Ziel dieser Ausarbeitung war es, verschiedene kognitive Frameworks näher zu untersuchen und zu vergleichen. Dazu sollte darauf eingegangen werden, in welchen verschiedenen Bereichen die jeweiligen Frameworks Anwendung finden und worin ihre Stärken und ihre Schwächen liegen. Um die Funktionsweise der Frameworks besser zu verstehen, war dazu eine kurze Einführung in die Grundlagen der Kognitionswissenschaft und künstlichen Intelligenz verlangt. Außerdem sollte so festgestellt werden, wo die grundsätzlichen Grenzen von bestehenden kognitiven Architekturen liegen und ein Ausblick auf zukünftige Entwicklungen vorgestellt werden.

In diesem letzten Kapitel wird aus allen Informationen das Sieger-Framework herausgestellt und ein Ausblick in die weitere Entwicklung gegeben.

Das Vorgehen bei der Erstellung der Fachstudie, war zuerst zu untersuchen, was Kognition ist und welche Bereiche die Kognition umfasst. Anschließend wurden Techniken zur Künstlichen Intelligenz untersucht, sowohl klassisch als auch kognitionsbasiert. Somit waren die Grundlagen gelegt um die Frameworks auf ihre Eigenschaften zu untersuchen. Dabei wurden die Frameworks ACT-R, Apex und Soar untersucht und anschließend verglichen.

ACT-R ist ein populäres Framework mit einem breiten Spektrum an Einsatzgebieten und Ergebnisvisualisierungsmethoden, was aber nur schwer zu bedienen ist. Apex dagegen spezialisiert sich auf autonome Roboter und ist leicht erlernbar. Soar hat eine Ähnlichkeit zu ACT-R. Es ist universell einsetzbar und implementiert den symbolischen Ansatz. Ein gravierender Unterschied ist, dass Chunks generierte Produktionen und keine Informationselemente für den subsymbolischen Ansatz sind. Die Bedienung ist mit Hilfe des Debuggers einsteigerfreundlich und erlaubt den Einblick in die ablaufenden Prozesse.

Je nach Anwendungsfall eignet sich eher das eine oder das andere Framework besser. Soll das jeweilige Framework eher in der Kognitionsforschung eingesetzt und z. B. die Aktivitäten der einzelnen Gehirnregionen bei unterschiedlichen Aktionen untersucht werden, so ist ACT-R die beste Wahl. Für den praktischen Einsatz in der Robotik eignet sich Apex am besten, da in diesem Bereich damit bereits viel Erfahrung gesammelt wurde und die Überwachung der einzelnen Abläufe in der Ergebnisvisualisierung klar dargestellt werden können. Ist der Einsatzbereich eher allgemein und soll eine einfache Programmiersprache eingesetzt werden, so überzeugt Soar damit, dass es praktisch in

allen Anwendungsbereichen verwendet wird und mit vielen unterschiedlichen Programmiersprachen (C++, Java, C#, Tcl) angesteuert werden kann. Soar ist außerdem sehr gut strukturiert und dokumentiert und einfach aufgebaut. Diese Punkte machen Soar zu einem optimalen Framework in den meisten Bereichen, wodurch es als Gewinner dieser Ausarbeitung hervorgeht.

Die hier untersuchten Frameworks und die Vielfalt weiterer Frameworks, welche die Autoren hier nicht ausführlich untersuchen konnten, zeigen wie weit die Entwicklung in einigen Teilbereichen bis heute fortgeschritten ist. Denn auch die Forscher der Informatik sind an der tatsächlichen Funktion des Gehirns interessiert, um Algorithmen zu entwerfen, welche effizienter arbeiten als bisherige. Die Forschung auf diesem Gebiet entwickelt sich immer weiter. Somit könnte es in Zukunft möglich sein, dass Computer Probleme noch schneller und besser lösen können als wir Menschen und, dass im Endeffekt Maschinen tatsächlich unsere Hausarbeit übernehmen. Es wäre möglich Software für einen Roboter zu bauen, der im Haushalt Dinge wie Staubsaugen, Spülen oder sogar Aufräumen kann.

Vermutlich wird auch die Hardware-Entwicklung in den nächsten Jahren mit immer komplexeren Werkzeugen verbessert, die wiederum die Entwicklung noch komplexerer Werkzeuge und Systeme ermöglichen. In der modernen Prozessorentwicklung mit mehreren hundert Millionen Transistoren ist schon lange keine Entwicklung auf unteren Ebenen, wie mit einzelnen Logik-Gattern, mehr möglich und ebenso ergeht es in der Software-Entwicklung. Eines Tages werden wir es dann mit Robotern wie „Data“ aus „Star Trek - The Next Generation“ zu tun haben, welche erst mit Fortschritten auf beiden Seiten ermöglicht werden.

Interessant wäre auch die Möglichkeit, künstlich intelligente Haustiere bauen zu können, welche dann z. B. genauso reagieren wie ein Hund es tut und nicht mehr von biologischen Tieren zu unterscheiden sind. Der Erfolg des Spielzeughunds Aibo zeigt, dass das Nachbilden der Lebewesen, obwohl nur rudimentär, möglich ist. Schon heute zeigt sich die Akzeptanz in der Bevölkerung für künstliche Lebewesen[12]. Noch weiter könnte man gehen, wenn Roboter irgendwann jede Arbeit für uns erledigen und wir sie nur noch kontrollieren wie im Film „I, Robot“.

[12]Mehr zu diesem Thema in der Fachstudie „Avatar-Frameworks“[9]

Literatur

[1] ABENDBLATT.DE: *Hamburger Abendblatt*. http://www.abendblatt.de. Version: 2011. – [Online; Stand 25. April 2011]

[2] ANDERSON, John R.: *Foto*. http://act-r.psy.cmu.edu/people/ja/ja.jpg. Version: 2011. – [Online; Stand 2. Mai 2011]

[3] ANZEIGER, Gießener: *Gießener Anzeiger - Vortragsreihe Physik im Blick widmet sich dem Thema Sinne*. http://www.giessener-anzeiger.de/lokales/hochschule/9882978.htm. Version: 2011. – [Online; Stand 30. Mai 2011]

[4] ASIMOV, Isaac: *Meine Freunde, die Roboter*. Überarb. Neuausg. Heyne Verlag, 2002. – ISBN 978–3453215313

[5] BOTHELL, Dan: *ACT-R Environment Manual*. http://act-r.psy.cmu.edu/actr6/EnvironmentManual.pdf. – [Online; accessed 3-July-2011]

[6] BRÜSSOW, Sven ; HOLT, Daniel: *Einführung in die kognitive Modellierung mit ACT-R*. http://www.psychologie.uni-heidelberg.de/ae/allg/mitarb/sb/www/actr/seminar/folien/actr-2007-10-24.pdf, Abruf: 30.04.2011

[7] BUDIU, Raluca: *About ACT-R*. http://act-r.psy.cmu.edu/about/, Abruf: 05.05.2011

[8] DENNETT, D.: *Cognitive Wheels: The Frame Problem of AI*. http://www.uibk.ac.at/psychologie/mitarbeiter/leidlmair/cognitive-wheels_leopold-meuer_thomas-rieger.pdf, Abruf: 18.03.2011

[9] DUSCHEK, Alexander ; SCHUSTER, Philipp ; TU, Xi: *Fachstudie: Avatar-Frameworks*. Universität Stuttgart, 2011

[10] GHOLAMSAGHAEE, Ehsan: *SOAR*. www.dfki.de/~kipp/seminar/folien/Ehsan_SOAR.pdf. Version: 2011. – [Online; Stand 3. Juni 2011]

[11] GOOGLE ; INC.: *Google Maps*. http://maps.google.de. Version: 2011. – [Online; Stand 11. April 2011]

[12] GORLICK, Adam: *Media multitaskers pay mental price, Stanford study shows*. http://news.stanford.edu/news/2009/august24/multitask-research-study-082409.html, Abruf: 02.03.2011

[13] GÖRZ, Günther: *Einführung in die künstliche Intelligenz*. 2. Auflage. Addison Wesley, 1995. – ISBN 978–3893198580

[14] GÜNTER GEHL: *Poetron*. http://www.poetron-zone.de, Abruf: 02.03.2011

[15] Hoffmeister, H.: *Impulsfortleitung an der Nervenzelle*. http://de.wikipedia.org/w/index.php?title=Datei:Impulsfortleitung_an_der_Nervenzelle.png&filetimestamp=20101201105237. Version: 2005. – [Online; Stand 15. Juli 2011]

[16] Julia Portl, Christoph H.: *Softwarepraktikum: A-Stern Algorithmus*. http://pille2.iwr.uni-heidelberg.de/~astar01/. Version: 2011. – [Online; Stand 24. Juni 2011]

[17] Kaczmarczyk, Peter P.: *SOAR Eine Kognitive Architektur*. www.dfki.de/~kipp/seminar_ws0607/reports/Soar.pdf. Version: 2011. – [Online; Stand 22. April 2011]

[18] Kuhl, Matthias: *Anatomie und Physiologie des Gehirns*. cgi.server.uni-frankfurt.de/fb05/fspsych/modules.php?name=Downloads&d_op=getit&lid=20. Version: 2011. – [Online; Stand 05. Juni 2011]

[19] LadyofHats ; NEUROtiker: *Complete neuron cell diagram*. http://commons.wikimedia.org/w/index.php?title=File:Complete_neuron_cell_diagram_de.svg&oldid=32050250&uselang=de. Version: 2007. – [Online; Stand 21. Juni 2011]

[20] Lenzen, Manuela: *Natürliche und künstliche Intelligenz*. Frankfurt/Main : Campus, 2002

[21] Liadal, Terese: *ACT-R: A cognitive architecture*. http://www.dfki.de/~kipp/seminar_ws0607/reports/ActR.pdf, Abruf: 29.06.2011

[22] Major, David: *Geschichte des Konnektionismus*. http://www.logic.at/lvas/185170/13-Major.pdf. Version: 2004. – [Online; Stand 16. Mai 2011]

[23] Miller, George A.: *The Magical Number Seven, Plus or Minus Two - Some Limits on Our Capacity for Processing Information*. http://www.psych.utoronto.ca/users/peterson/psy430s2001/Miller%20GA%20Magical%20Seven%20Psych%20Review%201955.pdf, Abruf: 04.06.2011

[24] Müller, Anja: *Im Hirn gelandet*. http://www.handelsblatt.com/politik/oekonomie/nachrichten/im-hirn-gelandet/3253500.html. Version: 2009. – [Online; Stand 23. Mai 2011]

[25] NASA: *Human-Computer Interaction Analysis (CPM-GOMS)*. http://ti.arc.nasa.gov/projects/apex/projectHCI.php, Abruf: 7.5.2011

[26] NASA: *Virtual Airspace Modeling and Simulation Project (VAMS)*. http://ti.arc.nasa.gov/projects/apex/projectVAMS.php, Abruf: 7.5.2011

[27] Newell, Allen ; Simon, H. A.: *Computers & thought*. Cambridge, MA, USA : MIT Press, 1995

[28] Obermaier, Claudia: *Mentale Modelle und kognitive Täuschungen*. http://www.uni-koblenz.de/~beckert/Lehre/Seminar-LogikaufAbwegen/obermaier_ausarbeitung.pdf, Abruf: 28.02.2011

[29] Pugh, David: *Player's Choice Video Game Superstore*. http://www.playerschoicegames.com. Version: 2011. – [Online; Stand 25. April 2011]

[30] Quillian, M. R.: *Semantic Information Processing*. 1. Auflage. Marvin L. Minsky, 1969. – ISBN 978–0262130448

[31] Reubenbarton: *Shuttle profiles*. http://commons.wikimedia.org/w/index.php?title=File:Shuttle_profiles.jpg&oldid=54672297. Version: 2005. – [Online; Stand 21. Juni 2011]

[32] Salvucci, Dario D. ; Kushleyeva, Yelena ; Lee, Frank J.: *Toward an ACT-R General Executive for Human Multitasking*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.1378&rep=rep1&type=pdf. – [Online; accessed 29-June-2011]

[33] Schanz, Michael: *Einführung in die Verteilte Künstliche Intelligenz*. 05.11.2008

[34] Schwabe, Willmar: *Anatomie des Gehirns*. http://www.mental-aktives-lernen.de/typo3temp/pics/f6bbdb6e40.jpg, Abruf: 27.05.2011

[35] Schöning, Uwe: *Theoretische Informatik - kurz gefasst*. 5. Auflage. Spektrum Akademischer Verlag, 2008. – ISBN 978–3–8274–1824–1

[36] Shannon, C. E.: *A Mathematical Theory of Communication*. http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf, Abruf: 04.06.2011

[37] Siegfried Brehme, Irmtraut M.: *Wissensspeicher Biologie*. Cornelsen Verlag, 1998. – ISBN 978–3–06–011731–4

[38] Singh, Push: *Examining the Society of Mind*. http://web.media.mit.edu/~push/ExaminingSOM.html, Abruf: 15.07.2011

[39] Solso ; Robert, L.: *Kognitive Psychologie*. Heidelberg : Springer, 2005

[40] Sony: *AIBO ERS-7 (MIND 3) Produktbroschüre*. http://support.sony-europe.com/aibo/downloads/de/AIBO_MIND3_DE.pdf, Abruf: 19.04.2011

[41] Sürer, Fatma: *Der Turm von Hanoi und Turm von London auf dem Tablet-PC: Untersuchung des Problemlöseverhaltens von gesunden Kontrollpersonen und von Patienten mit umschriebenen Hirnläsionen*. http://edoc.ub.uni-muenchen.de/10637/1/Suerer_Fatma.pdf, Abruf: 18.03.2011

[42] Tanchoco, Jose M.: *Optical Character Recognition*. http://cobweb.ecn.purdue.edu/~tanchoco/MHE/ADC-is/OCR/main.shtml, Abruf: 18.03.2011

[43] Technology, Massachusetts I.: *Human-Systems Engineering (HSE).* http://esd.mit.edu/hse/, Abruf: 7.7.2011

[44] Touretzky, David S. ; Tira-Thompson, Ethan J.: *Tekkotsu: A Framework for AIBO Cognitive Robotics.* http://www.tekkotsu.org/media/CogRobotics-Touretzky-AAAI05.pdf. Version: 2005. – [Online; accessed 21-June-2011]

[45] Wagner, Martin: *Symbolic vs Connectionist?* http://www.logic.at/lvas/185170/16-Wagner.pdf. Version: 2004. – [Online; accessed 11-June-2011]

[46] Wikipedia: *ACT-R — Wikipedia, The Free Encyclopedia.* http://en.wikipedia.org/w/index.php?title=ACT-R&oldid=422243095. Version: 2011. – [Online; accessed 4-May-2011]

[47] Wikipedia: *Gehirn — Wikipedia, Die freie Enzyklopädie.* http://de.wikipedia.org/w/index.php?title=Gehirn&oldid=87206693. Version: 2011. – [Online; Stand 9. April 2011]

[48] Wombat, Mighty: *Mighty Wombat.* http://www.mightywombat.com. Version: 2011. – [Online; Stand 11. April 2011]