Hauptseminar über
Lernen in multiagenten System

Maschinelles Lernen und Multiagenten System

---

von Richard Klinger
SS 99, 8. Semester

betreut von M. Schulé

Motivation

Heute wie vor 500 Jahren versuchen die Menschen Maschinen zu bauen, die intelligenter, schneller und robuster als die Menschen selbst sind. Was damals mit mechanische Rechenwerke, primitiven Maschinen und einfache Materialien anfing wird heute mit Computertechnologie, Robotern und neue chemischen Verbindungen fortgesetzt. Dabei stößt der Mensch immer wieder auf neue Probleme, die ihn anscheinend immer weiter weg von seinem ursprünglichen Ziel bringt einen künstlichen Menschen zubauen. Die Komplexität des Menschen in seiner Denkweise und in seinem Handeln wurde erst in den letzten Jahrzehnten bewußt.

Seit ca. 40 Jahren versuchen Wissenschaftler aller Welt auf elektronischem Wege künstliche Intelligenz (KI) den Maschinen beizubringen. Doch mit mäßigem Erfolg, da das menschliche Gehirn zu facettenreich ist, um es auf eine einfache Art und Weise nachzubilden bzw. zu simulieren. Die heutigen Erfolge in Bezug auf die KI beschränken sich auf kleine Teilbereiche wie Expertensystem, neuronale Netzwerke, selbstlernende Algorithmen usw.

Interessant für die KI sind Systeme (Agenten), die man nicht expliziert programmieren muß, sondern die selbständig nach einer optimalen Lösung suchen. Um solche Systeme zu realisieren gibt es verschiedene Ansätze, wie Analogie, adaptives Lernen usw.
Für uns interessant ist das adaptive Lernen, da dort schon vereinzelt Erfolge erzielt worden sind. Das adaptive Lernsystem trifft Entscheidungen aufgrund von Erfahrungen, die es in der Vergangenheit gemacht hat.

Einführung

Um einzelnen Teilbereiche, wie autonome Agenten, multiagenten System, multiagenten Kooperation, Echtzeitverarbeitung, KI, Robotik, Sensorik „unter einen Hut“ zu bringen wurde 1992 RoboCup ins Leben gerufen [RoboCup92].
Deshalb sind hier die vorgestellten Verfahren über adaptives Lernen auf das RoboCup Spiel zugeschnitten. Die dort angewendete Strategie bzw. Technik läßt sich auch auf andere Situationen bzw. Systeme übertragen.
Wie oben schon erwähnt sind Erfahrungen für ein adatives Lernsystem sehr wichtig, da es sonst keine richtige Entscheidungen fällen kann.
Erfahrungen meistens durch Testen gesammelt. Nun wäre es sehr mühsam einen Roboter 10.000 mal einen Ball zuzuspielen und das Resultat dem Roboter mitzuteilen.
Deshalb geht man dazu über einen Simulator zu verwenden. Mit dem Simulator ist man nun in der Lage einen definierten Zustand herzustellen. Der simulierte Roboter führt eine Aktion aus und bekommt nach der Simulation der Aktion das Resultat mitgeteilt.
Dieser Vorgang kann nun beliebig wiederholt werden bis genügend Erfahrungen gesammelt wurden, um eine richtige Entscheidung fällen zu können.
Nach der Lernphase müssen die Erfahrungen auf den realen Roboter übertragen und evtl. angepaßt werden. Danach kann der reale Roboter mit den simulierten Erfahrungen in der realen Welt agieren.

Autonome Agenten

Definition

Eine feste Definition von Agenten gibt es momentan nicht, da der Begriff erst seit kurzem in der Informatik verwendet wird. Russel und Norvig definieren den Agenten wie folgt: *„ Anything that can be viewed as perceiving its environment through sensors and acting upon that environment throught effectors.“* [Russel95]

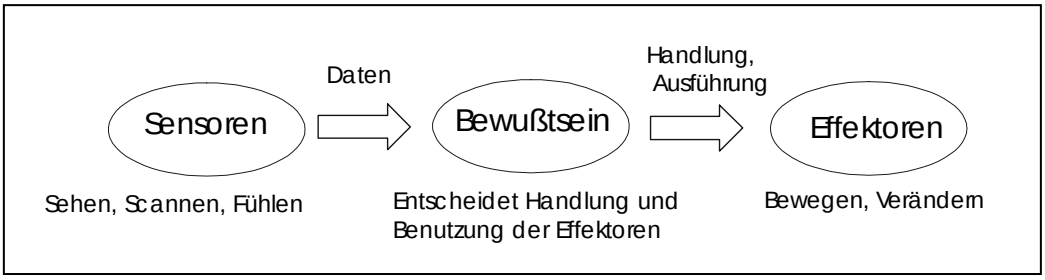

Abb. 1 Grundarchitektur eines autonomen Agenten

Ein Agent kann ein eigenständiges „Wesen“ mit Wissen, Ziele und Handlungen sein. Er ist fähig aufgrund seiner Sensoren autonom mit seinen Effektoren zu agieren, um seine spezifizierten oder selbst erzeugten Ziele zu erreichen (Abb. 1). Das spezifizierte Verhalten der autonomen Agenten fällt in den Unterbereich Verteilte KI (VKI).

Merkmale

Von Software Agenten (softbots) [Jenning96] bis zu autonome mobile Roboter bzw. autonome Fahrzeugen sind autonome Agenten anzutreffen. Adaptives, robustes, taktisches und vielseitiges Verhalten sind wichtige Eigenschaften der Agenten. Adaptives Verhalten paßt den Agenten an die jeweilige Umgebung an. Fehlertoleranz macht ihm robuster. Taktisches situationsbedingtes Verhalten, um eines der mehren Ziele zu verfolgen macht den Agenten vielseitig. Die unterschiedlichen Eigenschaften der Agenten kann zur Klassifikation benützt werden (Tab. 1) [MSCI]. Trotz der heutigen Technologie ist es nicht möglich alle Eigenschaften zu vereinen.

Eigenschaft	**Bedeutung**
reagieren	Antwort auf eine Veränderung in seiner Umgebung in einer bestimmten Zeit
autonom	Ausführung eigener Entscheidungen
zielorientiert	Absicht verfolgend, keine einfachen Reaktionen auf veränderte Umgebung
kommunikativ	Kommunikation mit anderen Agenten, bzw. Menschen
adaptives Lernen	Verhalten wechselt je nach Erfahrung
mobil	in der Lage sein sich selbst zu bewegen
flexible	keine festgelegten Ausführungen
Charakter	Gefühlszustand und Persönlichkeit

Tabelle 1 Klassifizierung von Agenten

Abbildung 2 zeigt die Untergliederung der einzelnen autonomen Agenten [Keil89]. Das Grundmodell der Agenten ist dabei entweder biologischer oder mathematischer Natur. In der ersten Stufe werden die Agenten in biologische, Roboter ähnliche und rechnerspezifische Typen unterteilt. In der nächsten Stufe wird unterschieden in Software und künstliche Agenten. Aufgaben spezifische, in der Unterhaltungsbranche tätige Agenten, sowie Viren fallen in die Software Agenten Sparte.
Weiter Klassifikationen könnten anhand von der Kontrolstruktur, Umgebung (Datenbank, Filesytem, Netzwerk, Internet), Sprache und Applikationen gemacht werden [MSCI].

Autonome Agenten sind hilfreich um in verseuchten Gebieten zu operieren, den Behinderten zu helfen oder im Weltall, in der Tiefsee oder auf Planeten zu agieren oder sie können die Produktivität und Qualität steigern.

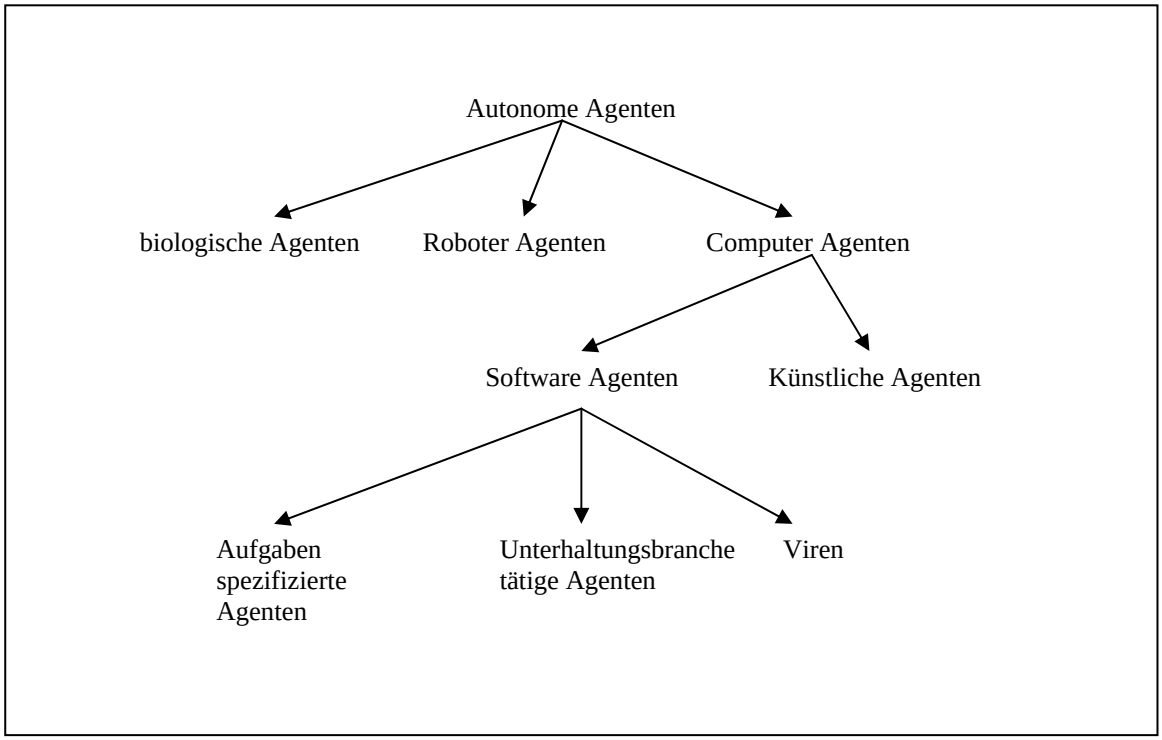

Abb. 2 Unterteilung von autonomen Agenten

Lernende Multiagenten

Das Lernen mit mehreren Agenten ist eine Mischung in den Bereich zwischen den multiagenten Systemen und dem maschinellen Lernen in der KI (Abb. 3) [Stone98].

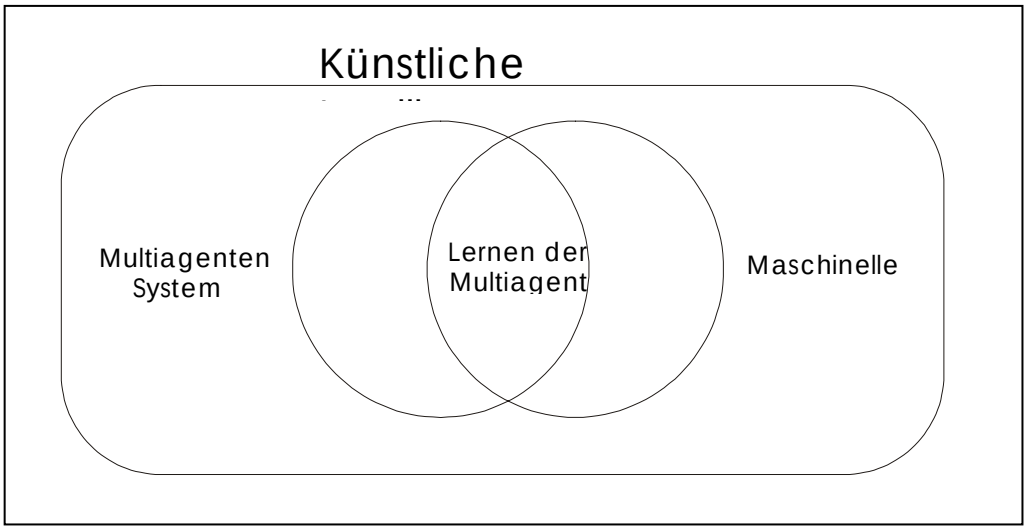

Abb. 3 Zuordnung der Lernenden Multiagenten

Beim konventionellen maschinellen Lernen wird zum Beispiel ein einziger Agent zur Funktionoptimierung eingesetzt ohne das mit andere Agenten zusammen gearbeitet wird. In den multiagenten Systemen befinden sich meistens mehrere autonome Agenten, die jeweils in der Lage sind mit anderen Agenten zu interagieren [Veloso97].
Das Lernen der Multiagenten ist der gezielte Umgang mit anderen Agenten. Dabei muß gelernt werden wie, wann und mit welchen Agent kooperiert werden kann bzw. muß.
In einigen Fällen kann das Grundverhalten des Agenten in einem multiagenten System individuell angelernt werden. Dies ist allerdings nur möglich, wenn andere Agenten vorhanden sind, die mit ihm kooperieren oder ihn behindern. Das dabei erlernte Grundverhalten kann dann für komplexere Interaktionen mit anderen Agenten genutzt werden.
Es ist möglich das Lernen solange durch ständiges hinzufügen von Verhaltensmuster zu erweitern bis das autonome Verhalten des Agenten zufriedenstellend ist (siehe Abschnitt: Entscheidung durch ein neuronales Netzwerk).

Man kann in der älteren Multiagenten Literatur Beispiele von Agenten finden, die in einer Multiagenten Umgebung lernen. In einer der ersten Multiagenten Berichte wird ein Agent beschrieben, der Informationen von einem andern Agenten erhält [Tan93].
Ein weiteres Beispiel erlernt ein Agent die Kommunikationstechnik eines anderen Agenten [Zeng96]. Hier kann man vom Lernen mit Multiagenten sprechen, da der lernende Agent eine Kooperation mit dem anderen Agent eingeht.
Als letztes Beispiel von Multiagenten Lernen ist ein Trainingsszenario, wobei ein unerfahrener Agent vom erfahrender Agent lernt [Clouse95]. Der unerfahrener Agent lernt in einer Rennsimulation von dem erfahrenden Agenten autofahren.
Alle diese Beispiel zeigen wie lernende Agent mit anderen Agenten interagieren, d.h. das Lernen ist also nur möglich, wenn andere Agenten vorhanden sind.

Adaptive Lernenverfahren

In den folgenden zwei Abschnitten werden zwei adaptive Lernverfahren vorgestellt, die in dem RoboCup Verwendung finden. Diese zwei Verfahren sind von Grund auf verschieden und eignen sich daher gut um die Stärken und Schwächen der beiden Verfahren gegenüberzustellen.

1. Verfahren: Entscheidungsfunktion mit einer Datenbank

Bei der in Abbildung 4 dargestellte Situation muß sich der Spieler entscheiden ob er den Ball abspielt oder in das Tor schießt.

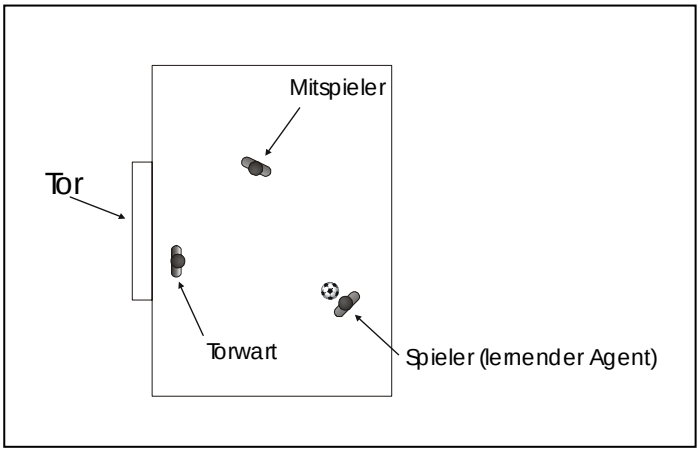

Abb. 4 mögliche Spielsituation

Es kann hilfreich sein Erfahrungen, die der lernende Agent bei einer ähnlichen Situation gemacht hat in die Entscheidung zu verwenden.
In diesem Verfahren werden Erfahrungen des Agenten in einer Datenbank gespeichert und bei Gebrauch wieder aus der Datenbank geholt. So kann er auf Erfahrungen zurückgreifen, die er früher gemacht hat.

Die zwei Funktionen $P_s(\phi)$ und $P_p(\phi)$ (s und p stehen für „Shoot“ und „Pass“) berechnen die Wahrscheinlichkeit ein Tor zu erzielen. Die Variable ϕ gibt dabei den Winkel vom Torwarts in seinem Kreis an, auf den er sich bewegt. Wenn diese Funktionen vollständig gelernt wurden, dann führen beide Funktionen ein binäre Abbildung aus, d.h. $P_s(\phi),P_p(\phi)$: $(0.0, 360.0)\rightarrow\{-1, 1\}$. Dies ist allerdings nur möglich, wenn sich der Torwart deterministisch verhält.
Nun kann der Agent für jeden Winkel ϕ die Wahrscheinlichkeit für Schießen bzw. Abspielen ausrechnen und entscheiden ob er schießt oder abspielt.

Speicher Modell

Damit der Agent die Funktionen $P_s(\phi)$und $P_p(\phi)$ verwenden kann, muß er seine Erfahrungen in seinem Speicher ablegen. Um alle Erfahrungen zu sichern würde der Speicher nach kürzester Zeit nicht mehr ausreichen. Durch eine Diskretisierung des Winkels ϕ in den Winkel Θ ist man in der Lage die Erfahrungen abhängig vom Index Θ zu speichern.
D.h. für eine Speichergröße von M (wobei M in 360 gleiche Bereiche aufgeteilt wird) erhalten wir Werte für $\Theta \in \{360n/M \mid 0 \leq n < 360\}$. Wir legen den Speicher als ein Array „MEM“ mit der Größe von M an, so das MEM[n] die Erfahrungen von den beiden Funktionen $P_s(\Theta)$ und $P_p(\Theta)$ speichern kann. Das Training wurden mit einer Speichergröße von M=360 bzw. M=18 durchgeführt. Wie wir bald sehen werden, hat die Speichergröße auf die Lerngeschwindigkeit einen großen Effekt.

Erfahrungen sichern

Mit der Diskretisierung kann es bei folgenden Trainingsversuchen Probleme beim Speichern kommen. Ein Training $E_{\phi,a,r}$ besteht aus einen Winkel ϕ, eine Aktion a und ein Resultat r, wobei Θ der relative Torwartwinkel ist, a für „s“ (shoot) oder „p“ (pass) und r für „1“ oder „-1“ (Tor oder kein Tor) steht. Zum Beispiel heißt $E_{72.345,p,1}$, daß ein Abspielen zum Tor führte als sich der Torwart bei der Position 72.345° auf seinem Kreis befand.

Ein einfaches Zuordnungsverfahren würde das Resultat unter dem Speicherindex Θ speichern, der am nächsten Winkel ϕ liegt. Zum Beispiel wäre Θ der nächstliegende Speicherindex vom Winkel ϕ, das MEM[Θ] definiert, dann wird durch die Zuweisung von $P_a(\Theta)=r$ das Resultat gespeichert.
Tritt der Fall auf das zwei Trainingseinheiten $E_{\phi 1,a,-1}$ und $E_{\phi 2,a,+1}$ existieren, so daß Winkel ϕ1 und Winkel ϕ2 auf den gleichen Speicherindex Θ gerundet wird, dann ist das Resultat nicht mehr eindeutig zuweisbar.
Um dieses Problem zu lösen wird der Wert von $P_a(\Theta)$ mit einem Faktor multipliziert. Der Faktor ist die Inverse vom Verhältnis zwischen der Winkeldifferenz Θ und ϕ, und der Speichergröße M. D.h. ist ein Resultat r das durch den Winkel ϕ gegeben wird multipliziert mit 1- (|ϕ-Θ| / (360/M)) bevor es im Array MEM[Θ] gespeichert wird. Dadurch werden die Winkel ϕ‘s, die näher am Speicherindex Θ liegen stärker gewichtet.
Wäre zum Beispiel die Speichergröße M=18 (d.h. MEM[Θ] ist dann definiert für den Speicherindex Θ = 20n, wobei n ∈ Natürlichen Zahlen) und $E_{116.5,p,-1}$ dann wird $P_p(120)$ ein Wert zugewiesen der sich aus –1*(1-3.5/20)=-0.825 errechnen läßt (siehe obige Formel).

Um ein Training $E_{\phi,a,r}$ in MEM[Θ] zu speichern wird:
1. r‘=r*(1- (|ϕ-Θ| / (360/M))) berechnet und dann
2. IF |r‘| > $|P_a(\Theta)|$ THEN $P_a(\Theta)$ =r‘

Durch dieses Verfahren wird $P_p(120)$ nur verändert, wenn $|P_p(120)| < 0.825$ ist. So werden nur die Resultate gespeichert, bei denen der Winkel ϕ am nächsten Speicherindex Θ steht. Da der Winkel 116.5 näher zum Speicherindex 120 als zum Speicherindex 100 ist macht es Sinn das der Erfahrungswert eine stärkere Auswirkung auf MEM[120] als auf MEM[100] hat.
Im unserem obigen Beispiel $E_{116.5,p,-1}$ wird nicht nur der Speicherplatz MEM[120] verändert. Ist |Pp(100)| kleiner ist als |r‘|=-1*(1-16.5/20)=-0.175, dann wird dieser Wert bei MEM[100] gespeichert.

Erfahrungen auslesen

Da jede einzelne Trainingseinheit in mehrere Speicherplätze gesichert wird, kann ein einfaches Ausleseverfahren angewendet werden, um die Erfahrungen aus dem Speicher zu lesen. Dazu wird der Winkel ϕ zum nächsten Winkel Θ gerundet, der einen Speicherplatz MEM[Θ] definiert und den Erfahrungswert von Pa(Θ) ausgelesen.

Entscheidung treffen

Die Entscheidungsmethode wählt die Aktion (shoot oder pass)aus, bei der die Wahrscheinlichkeit einen Treffer zu erzielen am höchsten ist. Ist zum Beispiel ϕ der Positionswinkel vom Torwart und liefern die Funktionen $P_s(\phi)$ und $P_p(\phi)$ die Erfahrungswerte, dann wird wie folgt entschieden:

```
IF Ps(ϕ)=Pp(ϕ) THEN
   bestimme SHOOT oder PASS zufällig (keine Erfahrungswerte vorhanden)
ELSE
   IF  Pp(ϕ)>0 UND Pp(ϕ)>Ps(ϕ) THEN
       PASS
   ELSE
      IF Ps(ϕ)>0 UND Ps(ϕ)>Pp(ϕ) THEN
         SHOOT
      ELSE
         IF Pp(ϕ)=0 THEN
             PASS (zum ersten Mal)
         ELSE
             IF Ps(ϕ)=0 THEN
                SHOOT (zum ersten Mal)
           ELSE
              bestimme SHOOT oder PASS zufällig (Pp(ϕ),Ps(ϕ)<0).
```

Eine Aktion wird Aufgrund der gespeicherten Erfahrungswerte ausgewählt. Wie oben ersichtlich bedeuten die Erfahrungswerten –1 zu 100% keinen Treffer, 0 zu 50% einen Treffer und +1 zu 100% einen Treffer. Somit wird die Aktion ausgeführt, bei der es am Wahrscheinlichsten ist einen Treffer zu erzielen. Ist kein positiver Erfahrungswert von der Funktion $P_p(\phi)$ und der Funktion $P_s(\phi)$ vorhanden, dann wird eine Aktion zufällig ausgesucht. Hat einer der beiden Funktionen einen Nullwert, dann wurde noch kein Erfahrung für diesen Fall gemacht. Durch einen Versuch wird getestet ein Tor zu erzielen. Das Resultat wird dann wie oben beschrieben

gespeichert. Somit lernt der Agent auch während dem Spiel dazu und kann seine fehlenden Erfahrungswerte ergänzen.

Experimente und Resultate

Die Erfolgsrate in der Praxis ein Tor zu erzielen liegt weit unter 100%, da es viele Torwartpositionen gibt, bei den weder ein direkter Schuß noch ein Abspielen zum Erfolg führte (siehe Abbildung 5).

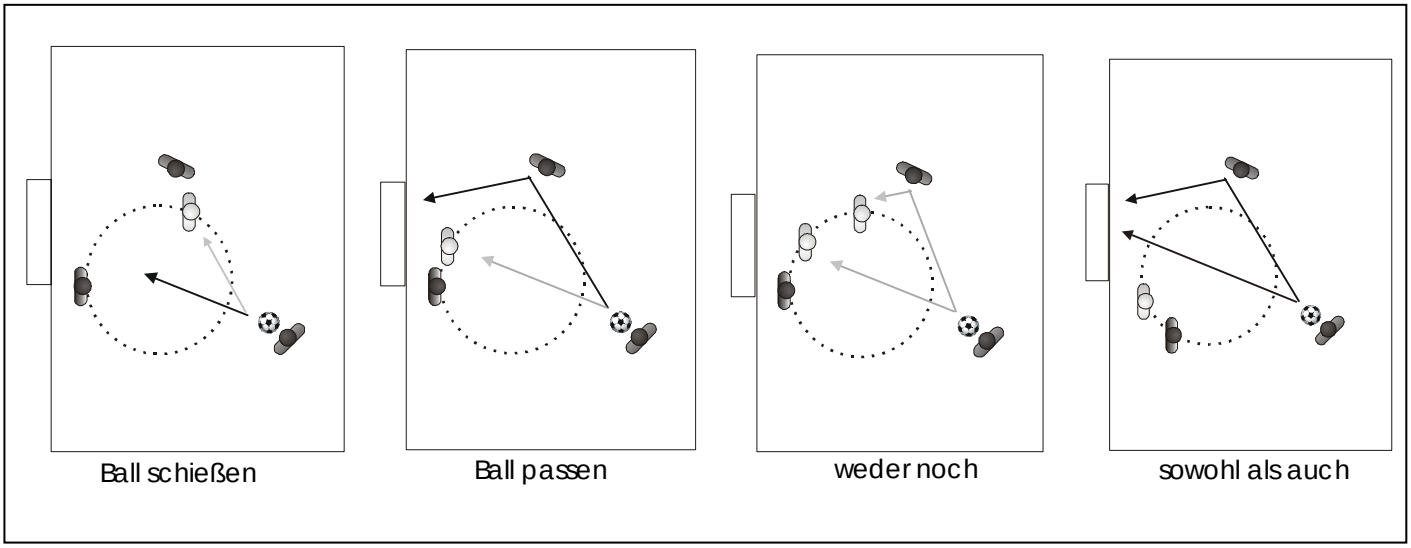

Abb. 5 Spielsituation

Wird der Torwart mit einer konstanten Geschwindigkeit von 50 Units/sek bewegt, dann ist die Wahrscheinlichkeit für ein Treffer 73.6%. Schießt oder spielt der Agent zufällig ab, ist die Wahrscheinlichkeit für ein Treffer 41.3%. Diese Meßwerte zeigen, daß der gelernte Agent den ungelernten Agent überlegen ist.
Um eine bessere Erfolgsrate zu erzielen verändert man das Verhalten des Agenten, indem der Agent bei $P_p(\phi),P_s(\phi)<0$ nicht reagiert. Dadurch steigt die Trefferrate auf 100%. Der Nachteil ist, daß der Agent immer solange wartete bis der Torwart außerhalb des Torbereich aufhält. Da aber beim Spiel nicht solange gewartet werden kann muß der Abspieler immer sofort schießen.

Speichergröße

Es wurden verschiede Speichergrößen ausgewählt, um eine optimale Speichergröße zu finden. Erfolgsrate und Effektivität wurden dabei als Entscheidungskriterium verwendet. Zusätzlich wurden zwei Geschwindigkeiten für den Torwart eingeführt.
In Tabelle 2 sind die entsprechenden Resultate zu entnehmen. In der Spalte „M“ steht die Speichergröße, die für die Tests verwendet wurde. In der Spalte „Success Rate“ ist die Wahrscheinlichkeit eines Tores angegeben. Dazu wurden 1000 Trainingssituationen simuliert und anschließend mit 1000 Testfällen überprüft. Die Spalten „Spd. 10“ und „Spd. 50“ gegeben dabei die Torwartgeschwindigkeiten an.
In der Spalte „Trials To Reach 70%“ steht die Anzahl der Versuche, um eine Erfolgsrate von 70% zu erreichen. Versuche, bei denen nicht 70% erreicht wurden steht ein „–„ Zeichen.
Je kleiner der Speicher, desto schneller (aber nicht so genau) wird gelernt. Je größer der Speicher, desto langsamer (aber dafür präziser) wird gelernt.
Wie man in der Tabelle 2 erkennen kann ist eine Speicheraufteilung von M=18 am besten geeignet, da der Lernaufwand im Verhältnis zur Erfolgsrate optimal ist.

M	Success Rate		Trials To Reach 70%	
	Spd. 10	Spd. 50	Spd. 10	Spd. 50
1	44.1	35.7	-	-
2	53.2	57.4	-	-
3	70.6	66.1	155	-
4	60.8	68.7	-	100
5	65.2	69.3	-	100
6	61.4	75.5	-	100
8	67.5	74.1	-	100
9	68.8	72.2	-	448
10	64.4	69.0	-	100
12	68.5	72.0	100	100

M	Success Rate		Trials To Reach 70%	
	Spd. 10	Spd. 50	Spd. 10	Spd. 50
18	69.7	74.9	126	100
20	71.9	71.1	297	139
24	67.3	74.7	219	100
30	72.9	74.5	166	319
36	73.7	75.1	153	189
40	74.4	74.4	319	315
45	72.8	76.0	465	362
60	71.4	76.2	461	100
180	75.8	74.3	655	821
360	73.5	74.0	1034	1080

Tabelle 2 optimale Speichergröße ermitteln

Entscheidung durch ein neuronales Netzwerk

Mit Hilfe eines neuronalen Netzwerkes (NN) versucht man ein adaptives Lernverfahren zu realisieren.
Im Training sind zwei Agenten vorhanden, um damit ein Low-Level Verhalten zu simulieren. Der Abspieler schießt den Ball zwischen Tor und Stürmer, der die Aufgabe hat im richtigen Moment loszulaufen, um den Ball direkt in das Tor zu schießen. Da der Ball in Bewegung ist wird es für den Stürmer schwierig den Ball zu treffen, da er abschätzen muß, wann er loslaufen muß. Um die Problematik etwas zu vereinfachen läuft der Stürmer mit einer konstanten Beschleunigung los. Hat er sich einmal zum Laufen entschieden, kann diese Entscheidung nicht mehr rückgängig gemacht werden. Durch die Ball- und Stürmerposition ist er in der Lage zu entscheiden, wann er loslaufen muß. Die Methode des Stürmers wie er seine Entscheidung fällt nennen wir „shooting policy“.
Abbildung 6 zeigt die Startposition der beiden Spieler. Die Position der Stürmer kann sich um 40 Units[1] (innerhalb des Aufenthaltsradius) abweichen. Ebenfalls ist die Ausrichtung nicht fest vorgegeben und kann bis zu 70 Grad abweichen. Die zwei Stürmer im Aufenthaltsradius in der Abbildung 6 zeigen eine Extremsituation

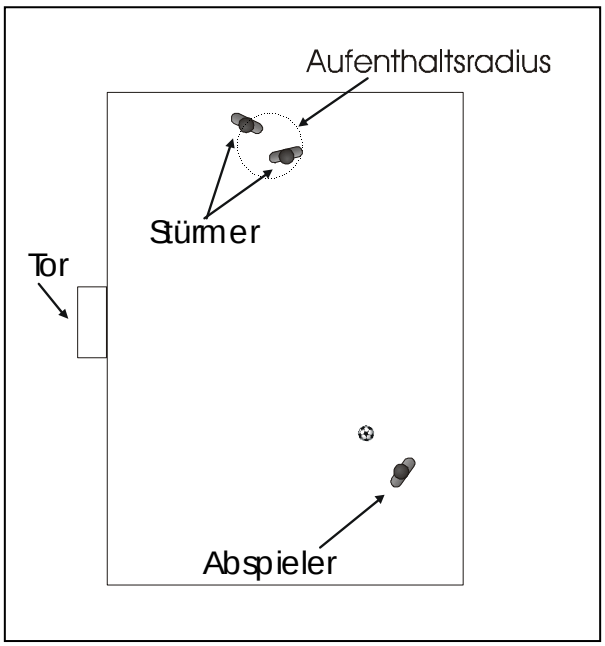

Abb. 6 Startzustand des Spielfeldes

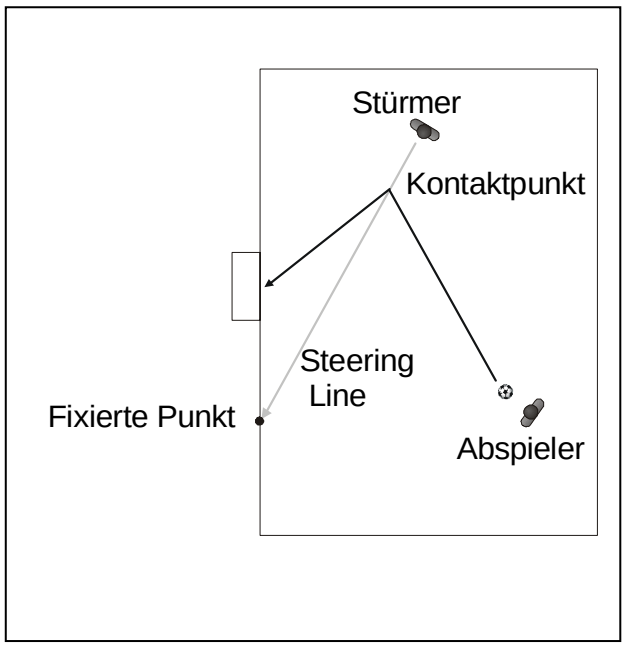

Abb. 7 Spielszenario

an.

Um ein Tor zu schießen muß der Stürmer den Weg des Ball korrigieren, wenn er gerade zwischen ihm und dem Tor befindet. Dazu fixiert der Stürmer einen imaginären Punkt an, der sich neben dem Tor befindet und beschleunigt dann auf der „steering line“ bis er auf den Ball stößt (Kontaktpunkt), den er dann ins Tor befördert (Abbildung 7). Die Methode des Stürmers mit der er den imaginären Punkt bestimmt nennen wir „aiming policy“.
Um die „shooting policy“ zu erlernen müssen mehre Faktoren berücksichtigt werden:

1. Die Ballgeschwindigkeit kann sich bei jedem zuspielen ändern.
2. Der Winkel mit dem der Ball zugespielt wird kann sich ändern.
3. Die Torposition kann sich bei jedem Zuspiel verändern, da der Torwart einige Bereiche vom Tor verdecken kann.
4. Die Informationen, die wir erhalten sind verrauscht.

Mit einer einfachen „shooting policy“, d.h. ist der Ball kleiner als 110 Units vom Stürmer entfernt, dann laufe los, ist die Trefferrate 100%. Dabei muß die Stürmerposition und Ausrichtung am Anfang exakt die gleiche sein. Wird aber die Position des Stürmers zufällig innerhalb des Aufenthaltsradius ausgewählt, dann liegt die Trefferrate nur noch 60.8% [Stone98].
Für eine komplexere „shooting policy“ müssen folgende Werte betrachtet werden (Abbildung 8):

1. Balldistanz: Entfernung zwischen dem Ball und dem Kontaktpunkt
2. Agentendistanz: Entfernung zwischen Kontaktpunkt und dem Agenten
3. Ausrichtungswinkel: Winkel zwischen der Agentenausrichtung und der „steering line“

[1] Einheit im Simulator

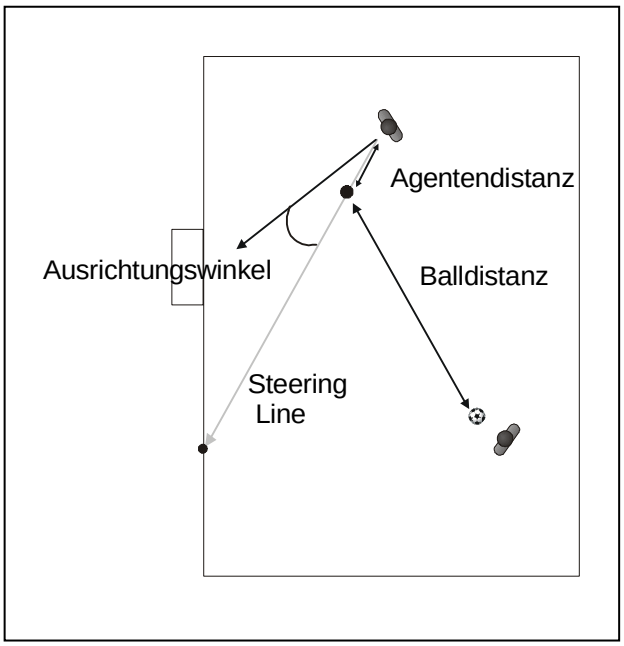

Abb. 8 Meßwerte

Low-Level Verhalten erlernen

In Abbildung 9 wird ein neuronales Netzwerk dargestellt, daß für die „shooting policy“ zuständig ist.
Das neuronale Netzwerk besteht aus 5 Eingängen, wobei 4 davon direkt an die unterste Schicht angeschlossen ist. In der mittleren befindet sich zwei unsichtbare Einheiten. Das neuronale Netzwerk gibt Werte zwischen 0.1 (bleiben) und 0.9 (starten) aus [Stone98].

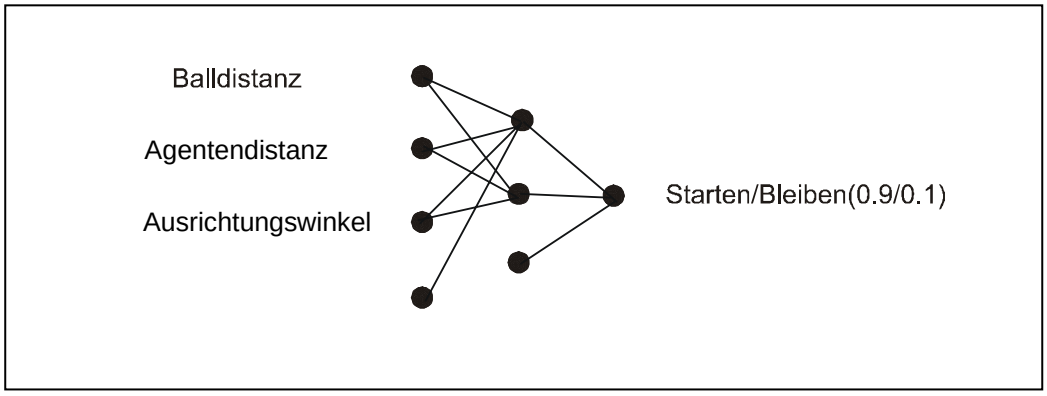

Abb. 9 neuronales Netzwerk

Testen des neuronalen Netzwerkes

Nachdem Training kann das neuronale Netzwerk benutzt werden. Die Balldistanz verringert sich, wenn der Ball in Richtung der „steering line“ geschossen wird. Das neuronale Netzwerk reagiert auf die Balldistanzverkürzung indem der Wert am Ausgang ansteigt. Ist die Ballposition optimal hat der Wert am Ausgang seinen Höchstpunkt erreicht. Rollt der Ball weiter zu der „steering line“ sinkt auch der Wert am Ausgang vom neuronalen Netzwerk. Der beste Zeitpunkt für den Stürmer loszulaufen wäre wenn der Wert am Ausgang am höchsten ist.
Ist die folgende Bedingung erfüllt, dann fängt der Stürmer an zu laufen.
IF (Wert am Ausgang > 0.6) UND (Wert am Ausgang < vorherige Wert – 0.01) THEN START
D.h. ist die Trefferrate größer als 60% und hat der Wert am Ausgang seinen höchsten Punkt erreicht, dann fange an zu laufen. Die 0.01 soll verhindern, daß durch die verrauschten Werte ein Fehlstart ausgelöst wird.
Durch dieses Verfahren erzielt der Stürmer eine Trefferrate um 96.5%.
In Tabelle 3 sind nochmals die einzelnen „shooting policy“ und ihre Erfolgsrate aufgelistet [Stone98].

Position des Stürmers	„shooting policy“	Erfolgsrate (in %)
variierend	zufällig	19.7
fest	einfach	100
variierend	einfach	60.8
variierend	3-Eingänge NN	96.5

Tabelle 3 Ergebnisse von verschiedenen „shooting policy“

Wird die Ballgeschwindigkeit zwischen 110-180 Units/sec variiert, dann beträgt die Trefferrate nur 49.1%. Fügt man nun einen weiteren Eingang an das neuronale Netzwerk hinzu, welche die Ballgeschwindigkeit angibt erhält man ein 4-Eingänge neuronales Netzwerk. Nach einem Training von ca. 4000 Versuchen kann die Ballgeschwindigkeit richtig abschätzen und der Stürmer läuft zum richtigen Zeitpunkt los [Stone98].
Tests haben gezeigt, das die Trefferrate des Stürmers bei 91.5% liegt. Tabelle 4 zeigt eine kurze Auflistung der verschiedenen „shooting policy“.

„shooting policy“	Erfolgsrate (in %)
zufällig	16.8
3-Eingänge NN	49.1
4-Eingänge NN	91.5

Tabelle 4 Ergebnisse bei variierenden Ballgeschwindigkeiten

Die Abschußposition des Balls ist selten die gleiche, deswegen verändert sich der Ausrichtungswinkel bei jeder neuen Situation (Abbildung 10). In den einzelnen Versuchen liegt der Ausrichtungswinkel zwischen 30-90 Grad. Die „shooting policy“ bleibt dabei unverändert, aber die „aiming policy“ muß modifiziert werden, da sonst der Stürmer das Tor verfehlt. Durch ein Verschieben des imaginären Punktes in vertikaler Richtung liegt die Trefferrate bei 96.3%, dabei muß das neuronale Netzwerk nicht erneut trainiert werden.
Mit einem zusätzlichen neuronalen Netzwerk kann man die „aiming policy“ so verändern, daß automatisch der

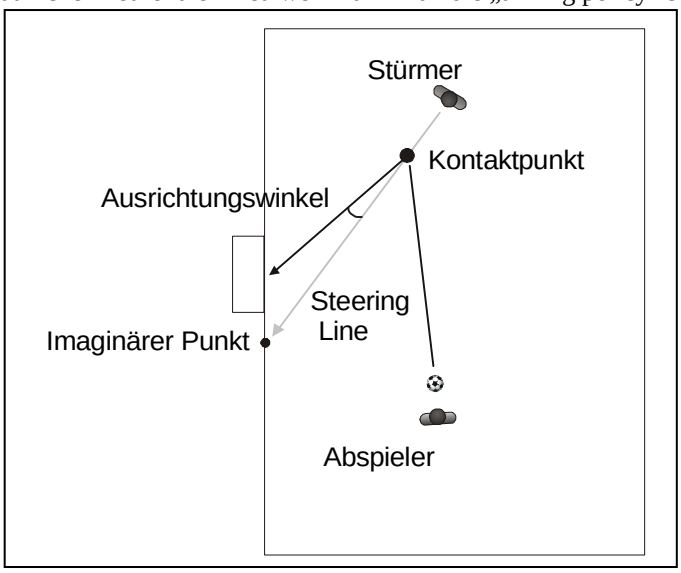

Abb. 10 Variierung des Ausrichtungswinkels

imaginäre Punkt berechnet wird. Dazu wird die Ballgeschwindigkeit und der Winkel zwischen Ball und Agent angegeben (Abbildung 11).

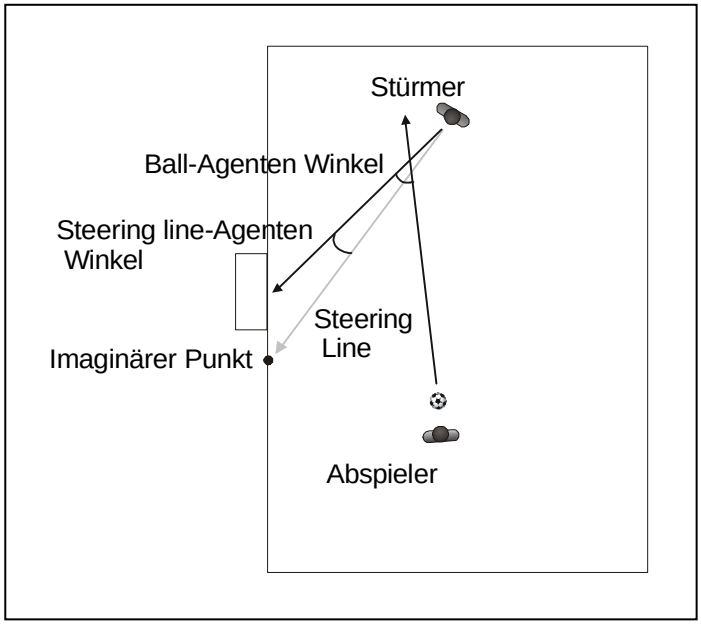

Abbildung 11 Bestimmung des imaginären Punktes

Mit dem neuronalen Netzwerk in Abbildung 12 ist man in der Lage den Winkel zwischen „steering line“ und den Agent zu berechnen und kann somit den imaginären Punkt bestimmen.
Mit der „aiming policy“ und der „shooting policy“ liegt die Erfolgsrate bei dem Stürmer bei 95.4%.

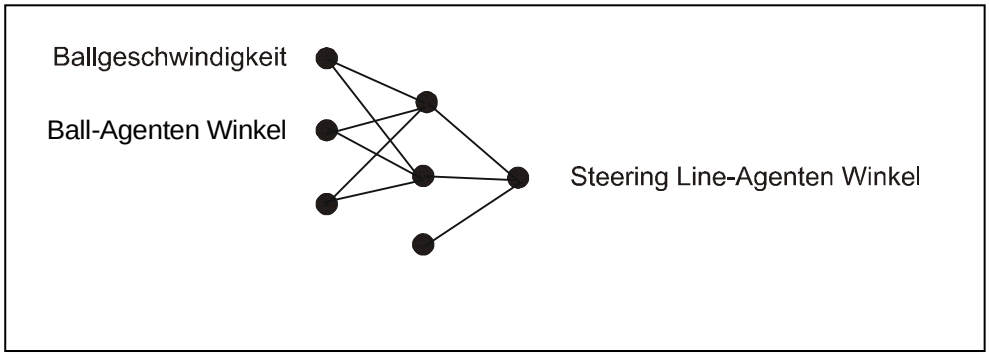

Abbildung 12 neuronale Netzwerk für die „aiming policy“

In Abbildung 13 muß der Stürmer mal auf das untere Tor und mal auf das obere Tor schießen. Diese Situation kann durch den Torwart entstehen, wenn sich dieser im Tor hin und her bewegt.
Die Trefferraten sind in Tabelle 5 zu entnehmen. Dabei fällt auf, daß die Trefferquote beim unteren Tor höher ist als beim oberen Tor. Das liegt vermutlich daran, daß es schwieriger ist den Ball in die Richtung zu schießen in der der Stürmer gerade läuft.

Torposition	Erfolgsrate (in %)
Mitte	95.4
Oben	74.9
Unten	84.1

Tabelle 5 Erfolgsrate bei verschiedenen Torpositionen

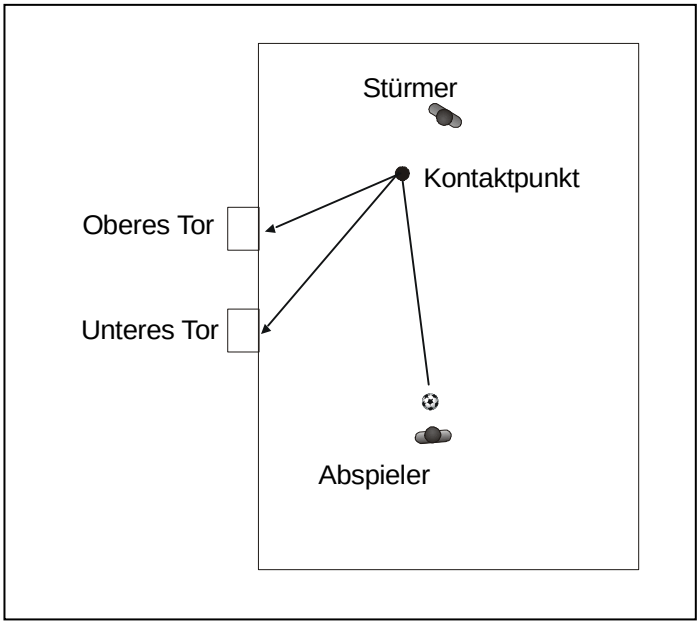

Abb. 13 Variierung der Torposition

Higher-Level Verhalten erlernen

Voraussetzung für ein higher-Level Verhalten der Agenten ist ein robustes low-Level Verhalten der Agenten. D.h. Agenten müssen sicher beim Abspielen und Zielen sein bzw. mit unterschiedlichen Ballgeschwindigkeiten und -winkel, unterschiedlichen Tor- und Spielerpositionen und simulierte Sensorverrauschung fertig werden. Durch das antrainierte Low-Level Verhalten der Agenten (siehe Oben) ist es nun möglich darauf ein higher-Level Verhalten aufzubauen. Ein solches kooperatives Verhalten ist in Abbildung 14 zu sehen.

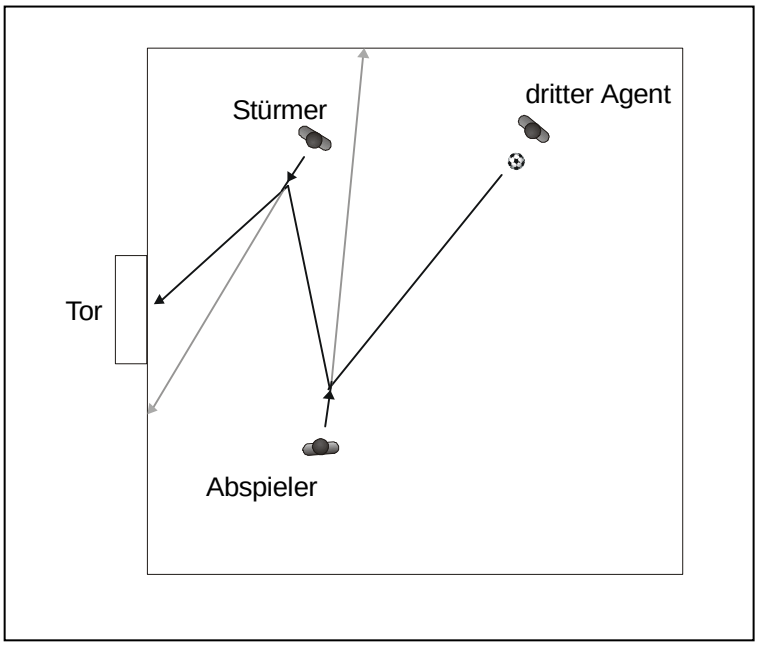

Abb. 14 kooperatives Verhalten

Hier kooperierte der Abspieler mit dem Stürmer, indem er den Ball zwischen Tor und Stürmer spielte. Nimmt man einen dritten Agenten ins Spielfeld wird die Kette der Abspieler etwas länger. Der dritte Agent spielt den Ball zum Abspieler, der wiederum den Ball zum Stürmer spielt, der dann den Ball ins Tor schießt. Theoretisch könnte man nun beliebig viele Agenten aufstellen, die den Ball gegenseitig zuspielen bis er dann schließlich im Tor landet.

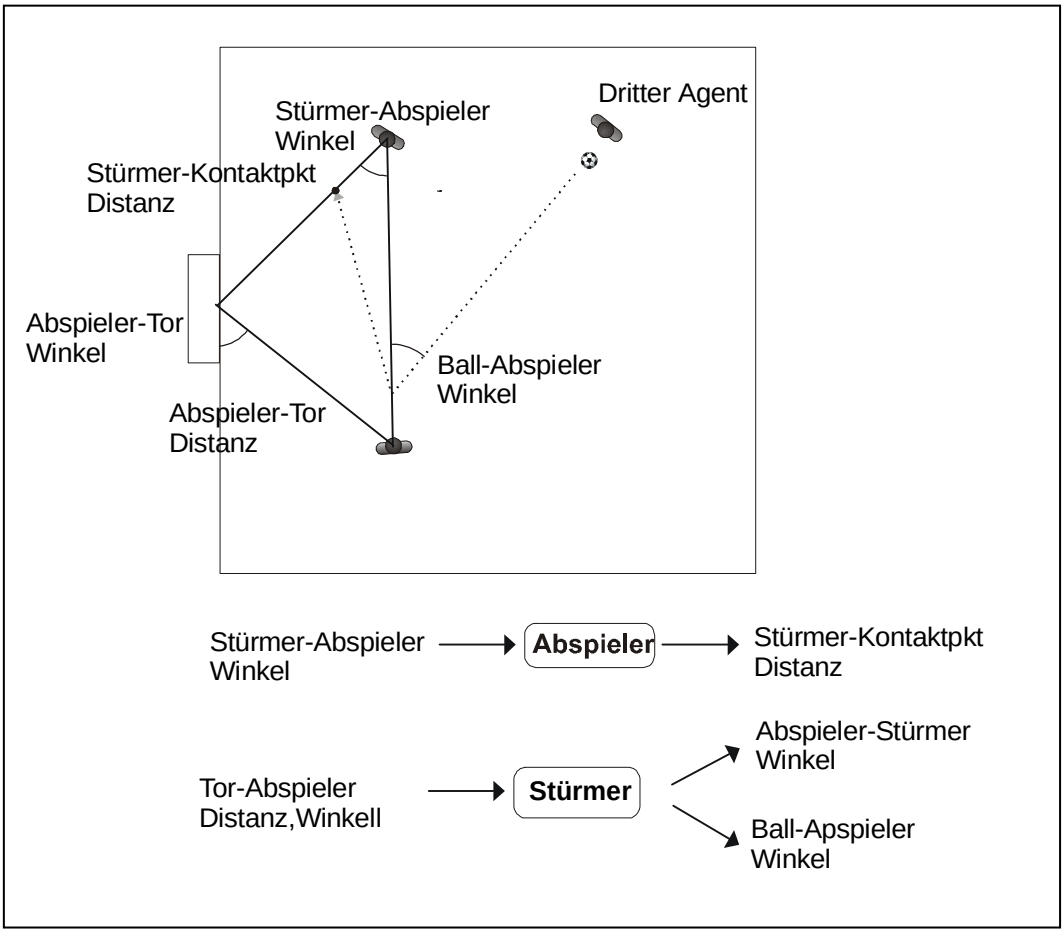

Abb. 15 Parameter für die Lernfunktion für den Stürmer und Abspieler

In Abbildung 15 werden die benötigten Parameter für ein Zusammenspiel zwischen Abspieler und Stürmer dargestellt. Aufgrund dem Stürmer-Abspieler Winkel kann der Abspieler den Stürmer-Kontaktpkt für den Stürmer bestimmen und den Ball in diese Richtung schießen. Der Stürmer kann bevor der Abspieler den Stürmer-Kontaktpkt ermittelt sich aufgrund der Tor-Abspieler Distanz bzw. Winkel in eine günstigere Position bringen,. Diese Funktionen (Abbildung 15) können durch neuronale Netzwerke ebenfalls realisiert werden.

Fazit

Beide Verfahren, die „Entscheidungsfunktion“ (Verfahren 1) und die Entscheidung mit dem neuronalen Netzwerk (Verfahren 2) lösen das Situationsproblem wie es in Abbildung 4 dargestellt wurde. Beide benötigen Trainingsversuche um mit eine optimale Lösung für die Situation zu finden. Das Verfahren 1 ist relativ einfach zu implemtieren und schon nach einer kurzen Trainingsphase (1000 Versuche) einsatzbereit. Das Verfahren 2 ist mit einem neuronalen Netzwerk aufwendig zu realisieren und die Trainigsphase (ca. 3000 Versuche) dauert länger.

Die Erfolgsrate liegt bei Verfahren 1 bei 74.9% und bei Verfahren 2 zwischen 74.9 und 95.4%.

In Verfahren 1 wird mit eine Torwart trainiert und deren Verhalten berücksichtigt. In Verfahren 2 wird die Präsenz eines Torwarts nur indirekt miteinbezogen, indem der Ball in die untere, in die obere Ecke bzw.in die Mitte geschossen wird. Das Verhalten des Torwarte bleibt dabei unberücksichtigt. Aber mit dem Verfahren 2 ist es möglich „higher-Level“ Verhalten zu realisieren.

In der Praxis kann Verfahren 1 realisiert werden, da die Torwartposition mit den Sensoren ermittelt werden kann. Verfahren 2 benötigt zudem noch die Ballrichtung und Ballgeschwindigkeit sowie die Ausrichtungen der einzelnen Spieler. Diese Informationen durch Sensoren, wie Scanner oder Kameras zu erhalten ist heute noch sehr aufwendig bzw. unmöglich. In der Simulation kann aber dieses Verfahren durchaus angewendet werden.

Anhang

Literaturverzeichnis:

Stone98	Torwards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer: Peter Stone and Manuela Velso, School of Computer Science. Carnegie Mellon University Pittsburgh, PA 1513
RoboCup92	Internet: http://www.robocup.org
Russel95	Russell, S. J. and Norvig, P. „Artificial Intelligence – A Modern Approach (AIMA).“ Englewood Cliffs, NJ, Prentice Hall 1995
Jenning96	Jennings, N. R. and Wooldridge, M. „Software Agents“ IEE Review 42(1), (Januar 1996), ff 17-21.
MSCI	Internet: http://www.msci.memphis.edu/franklin/AgentProg.html
Keil89	Keil, F.C. „Concepts, Kinds and Cognitive Development.“ Cambridge, MA, MIT Press, 1989.
Veloso97	Stone, P. & Veloso, M. (1997). Multiagent Systems: A survey from a machine learning perspective. Tech. rep. CMU-CS-97-193, Commputer Science Department, Carnegie Mellon University, Pittsburgh, P.A.
Tan93	Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In Proceedings of the Tenth International Conference on Machine Learning, ff. 330-337.
Zeng96	Zeng, D., & Sycara, K. (1996). Bayesian learning in negotiation. In Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium, ff. 99-104. Menlo Park, CA. AAAI Press. AAAI Technical Report SS-96-01.
Clouse95	Clouse, J.A. (1996). Learning from an automated training agent. In Weiß, G., & Sen, S.(Eds.), Adaptation and Learning in Multiagent Systems. Springer Verlag, Berlin.