Thema:

Implementierung einer Fußballmannschaft für den RoboCup SoccerServer

Ausarbeitung
im Rahmen des Seminars
Agent in simulierten Umgebungen

Betreuer: Dr. Dietmar Lammers

vorgelegt von:
Alexander Bernard
Wickenkamp 20
48161 Münster
0251/1356336
bernarda@uni-muenster.de

Johannes Tuchscherer
Gescherweg 60
48161 Münster
0251/83820480

Abgabetermin: 2004-01-20

Inhaltsverzeichnis

1 Motivation

Der Begriff des Agenten in der Informatik ist schillernd und viele verschiedene Definitionen lassen sich für ihn finden. Stuart Russel und Peter Norvig verstehen unter einem Software-Agenten eine Einheit, die in der Lage ist, ihre Umgebung durch Sensoren wahrzunehmen und mittels Aktoren in dieser Umgebung zu handeln.[1] Diese Definition lässt einige Aspekte der Agenten-Theorie außer Acht, die z. B. von Wooldridge in seiner Definition gefordert werden. Damit ist die Fähigkeit des Agenten gemeint, autonom zu handeln und mit anderen Agenten bzw. Elementen der Umwelt zu kommunizieren.[2] Eine für die vorliegende Arbeit nützliche Definition des Begriffes erhält man, wenn man beide vereint.

Eine konkrete Anwendung genau dieses Agenten-Verständnisses findet man in der Simulations-Liga der Roboterfußball-Organisation RoboCup. Diese Organisation veranstaltet jedes Jahr die Roboterfußball-Weltmeisterschaft und ermittelt in mehreren verschiedenen Disziplinen die Sieger. Eine dieser Disziplinen ist die Simulationsliga. Hier spielen keine realen Fußballroboter gegeneinander, sondern programmierte Agenten in einer simulierten Umgebung, die so weit wie möglich an die Regeln der FIFA[3] angepasst ist. Die Plattform, die dieses virtuelle Fußballstadion simuliert, ist der RoboCup Soccerserver. Dieser Server erlaubt es bis zu elf Agenten in zwei Mannschaften gegeneinander Fußball zu spielen. Umgesetzt ist er in einer Client-Server-Architektur, deren Kommunikation über UDP läuft. Die Agenten bekommen durch Sensoren, die Augen und Ohren simulieren, Informationen, können miteinander kommunizieren, die Umwelt verändern, und über ihre Handlungen selber entscheiden.

Ziel der Arbeit innerhalb des Seminars „Agenten in simulierten Umgebungen“ war es, für die Simulationsplattform Soccerserver eine Mannschaft bestehend aus elf Spielern zu entwickeln, die miteinander kommunizieren und interagieren. Das Verhalten der Mannschaft sollte zudem mittels aus der KI bekannter Techniken wie neuronaler Netze, maschinelles Lernen oder genetischer Programmierung optimiert werden.

In der vorliegenden Ausarbeitung wird die Programmierung der Mannschaft beschrieben. Im nächsten Kapitel wird zunächst unsere Herangehensweise näher erläutert und beschrieben, wie wir aufbauend auf einem simplen Spielerprogramm eine komplette Mannschaft entwickelten. Der nächste Abschnitt widmet sich der Architektur und dem Aufbau unseres Programms. Dies soll das Verständnis für die folgenden Kapitel erleichtern, in denen zunächst die Taktik der Mannschaft und das Verhalten einzelner Spieler und die Steuerung

[1] Vgl. Russel (2003), S. 32f.
[2] Vgl. Wooldrige (2002), S. XI.
[3] Fédération Internationale de Football Association, Weltfußballverband

2 Implementierung einer Fußballmannschaft für den RoboCup SoccerServer

2.1 Herangehensweise

Am Anfang der Entwicklung einer Fußball-Mannschaft für den SoccerServer stand die Entscheidung über die zu verwendende SoccerServer-Version, eine geeignete Programmiersprache und Entwicklungsumgebung. Unsere Implementierung einer Fußball-Mannschaft wurde für den SoccerServer in der Version 9.4.5 optimiert und zur visuellen Darstellung verwenden wir den SoccerMonitor und das SoccerWindow 1.3.1. Wir entschieden uns für Java als Implementierungssprache, da es sich hierbei um eine moderne objektorientierte Programmiersprache handelt und bei uns bereits Kenntnisse in dieser Programmiersprache vorhanden waren. Als Entwicklungsumgebung verwendeten wir Eclipse 3.1 und einen CVS-Server, um parallele und konsistente Programmierung zu ermöglichen.

Bei der Implementierung unserer Fußballmannschaft verzichteten wir auf eine komplette Neuentwicklung und verwendeten stattdessen den „Simple Client“ als solide Grundlage. Der „Simple Client“ ist ein rudimentärer Spieler-Client für den SoccerServer, der einen einfachen Algorithmus besitzt und im Gegensatz zu den meisten Agenten-Implementierungen nicht in C, sondern in Java programmiert wurde.

Der Algorithmus des „Simple Client“ ist sehr einfach und besteht lediglich aus vier Handlungsanweisungen, die sich in einer Schleife wiederholen:

1. Suche den Ball
2. Laufe zum Ball
3. Finde das Tor
4. Schiesse den Ball Richtung Tor

Aufgrund seiner Einfachheit hat der „Simple Client“ -Algorithmus zahlreiche Schwächen. So läuft der Agent manchmal aus dem Spielfeld und wertet viele wichtige Umwelt-Informationen nicht aus: Z. B. werden die hörbaren (`hear`-Nachrichten) und die fühlbaren (`senseBody`-Nachrichten) Informationen gar nicht ausgewertet und nur ein geringfügiger Teil der visuellen Mitteilungen (das gegnerische Tor und der Ball) werden für die Entscheidungsfindung verwendet.

Unser erstes Ziel war es nun, den „Simple Client“ so zu erweitern, dass er alle verfügbaren Informationen seiner Umgebung auswertet und sich mit ihrer Hilfe autonom in seiner Umwelt zurechtfindet. Zusätzlich sollten alle Server-Parameter eingelesen werden. D. h. der Agent liest bei seiner Initialisierung alle Server-Parameter (unter anderem Kick-Reichweite, Torgröße, …) ein, die in den Dateien server.conf und player.conf gesetzt und konfiguriert werden und speichert die Werte in internen Variablen. Wird später die Parameter-Konfiguration des Servers geändert, dann muss der Algorithmus nicht umständlich umgeschrieben werden, weil alle Werte dynamisch eingelesen werden. Parallel dazu entwickelten wir eine Client-GUI, die alle Informationen und Spielaktionen strukturiert und visuell darstellen sollte. Da der Spieler immer nur einen Teilausschnitt seiner Umwelt wahrnimmt, war es wichtig, dass der Spieler mithilfe weniger Spielfeldmarkierungen (Flags) seine Position auf dem Spielfeld lokalisiert. Dazu wurde die Orientierung als eine Kernkomponente unserer Fußballmannschaft implementiert. Durch diese konnte die Position des Spielers auf dem Spielfeld anhand zweier Flags eindeutig bestimmt werden.

Nachdem wir einen gut funktionierenden Einzelspieler hatten, haben wir komplexere Spielaktionen wie Dribbeln, Passen und Ball-Annahme eingebaut. Parallel dazu wurde die GUI weiterentwickelt und ein Torwart aufgestellt.

Im nächsten Schritt wurde versucht eine komplette Mannschaft bestehend aus 11 Agenten mit Mannschaftstaktik aufzustellen. Dazu wurde für jeden Spielertyp eine eigene Klasse angelegt, in der die individuellen Spieler-Fähigkeiten implementiert wurden, die aber alle von der Player-Klasse des Standard-Players erben. Hierbei spielte vor allem die Kommunikation und Abstimmung der Aktionen zwischen den Spielern eine wichtige Rolle. Erst als dies fertig war, wurden die Standardsituationen, wie z. B. Ball-Einwurf, Eckball, … eingebunden.

Am Ende des Entwicklungsprozesses hätte das fertige Team auf Fehler und mögliche Schwachstellen untersucht werden sollen, die bei der Entwicklung unentdeckt geblieben waren und versuchten den Algorithmus zu optimieren. Allerdings viel diese Phase wegen des hohen Termindrucks nur sehr spärlich aus.

2.2 Architektur

Die Architektur unserer Fußball-Mannschaft ist in Abbildung 2.1 in Form eines Klassendiagramms dargestellt und soll einen Überblick über die Struktur unserer Implementierung liefern.

Die Player-Klasse im Zentrum der Abbildung beinhaltet den Hauptalgorithmus und stellt somit die wichtigste Klasse dar. In ihr werden die verschiedenen Events[4] (siehe Abb. rechts oben) und die ObjectPerceptions (siehe Abb. links oben) ausgewertet, Entscheidungen getroffen und entsprechende Handlungsanweisungen (`turn`, `dash`, `dribble`, `kick`, …) in der Actor- und SpecialMoves-Klasse ausgeführt.

Jeder einzelne Spielertyp besitzt eine eigene Klasse, die von der Player-Klasse erbt. In diesen 11 Klassen stehen die individuellen Methoden und Attribute für den jeweiligen Spieler. So benötigt ein Torwart verständlicherweise einen anderen Spiel-Algorithmus als z.B. ein Stürmer.

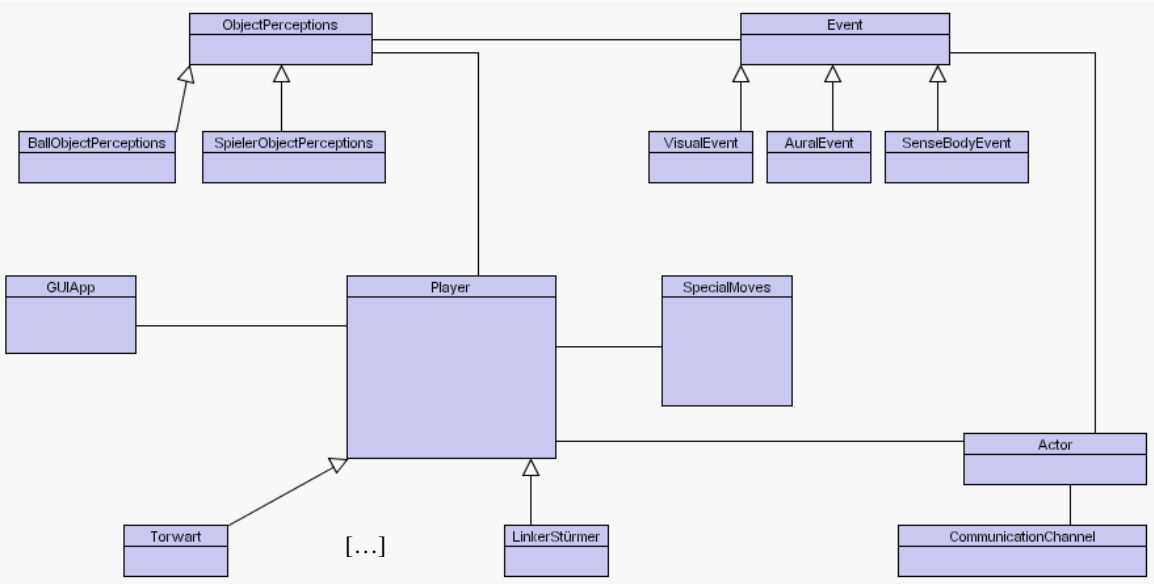

Abb. 2.1: Architekturübersicht

In der Actor-Klasse befinden sich die einfachen Basisaktionen, die jeder Spieler besitzt und die der SoccerServer standardmäßig unterstützt, z. B. `turn`, `dash`, `move`, `kick`, … Diese Methoden wandeln z. B. den `turn`-Befehl in einen für den Server verständlichen String um und rufen die `send`-Methode in der CommunicatorChannel-Klasse auf, die diesen Befehl an den Server weiterschickt. Die CommunicatorChannel-Klasse ist

[4] Ein Event entspricht einer Nachricht des Servers an den Agenten. Dies können Nachrichten über akustische oder visuelle Informationen sein, aber auch Statusinformationen des Spielers und Informationen zu den Parametern des Servers.

nur für die Kommunikation zwischen SoccerServer und Agent via UDP zuständig, d.h. sie sendet und empfängt Nachrichten und gibt sie an die Actor-Klasse weiter.

Komplexere Spielaktionen wie Dribbeln, Passen, Ball-Einwurf,… befinden sich in der SpecialMoves-Klasse, die über die Player-Klasse auf die Basis-Befehle der Actor-Klasse zugreifen. So benötigt z. B. die komplexe Aktion „Dribbeln“ neben der Dribble-Logik die Basis-Befehle `dash` und `kick`, die in der Actor-Klasse liegen.

Die Klassen VisualEvent, AuralEvent, SenseBodyEvent werden durch die Actor-Klasse instantiiert, je nachdem welche Art von Ereignis eingetreten ist. Tritt z. B. ein visuelles Ereignis ein, dann wird ein VisualEvent-Objekt erzeugt, das für jedes sichtbares Objekt (Flag, Mitspieler, …) ein eigenes ObjectPerceptions-Objekt erzeugt.

Die einzelnen Eigenschaften (unter anderem Zeitzyklus, Name, Richtung und Entfernung) der Spielobjekte (Ball, Flags, Mitspieler, Gegenspieler) werden in den ObjectPerceptions-Objekten gespeichert. Während für die statischen Objekte die ObjectPerceptions-Klasse verwendet wird, gibt es für die dynamischen Objekte (Ball und Spieler) die Klassen BallObjectPerceptions und PlayerObjectPerceptions, die von der ObjectPerceptions-Klasse erben.

Die Haupt-Gui-Klasse wird von der Player-Klasse aufgerufen. Sie bekommt von dort alle Informationen, die angezeigt werden sollen.

2.3 Spielerverhalten und Mannschaftstaktik

Bei unserer Fußball-Mannschaft wird folgende Mannschaftsaufstellung verwendet:

- Ein Torwart,
- drei Verteidiger,
- vier Mittelfeldspieler und
- drei Stürmer.

Das Spielfeld ist in Zuständigkeitsbereiche eingeteilt, so dass jeder Spieler einen eigenen Bereich hat für den er verantwortlich ist. Jeder Spieler besitzt eine Startposition und eine Standardposition. Die Standardposition gibt die Position im Zentrum seines Zuständigkeitsbereiches an. Der Spieler versucht sich immer an dieser Position aufzuhalten. Kommt der Ball in seinen Bereich, dann verlässt der Spieler seine Standardposition, um den Ball abzufangen und Aktionen auszuführen (z.B. passen, dribbeln oder Richtung Tor schießen).

Hat der Spieler den Ball abgegeben, dann versucht er wieder seine Standardposition einzunehmen.

Neben der Standardposition ist zusätzlich eine Startposition notwendig, da sich zu Beginn der ersten und zweiten Halbzeit alle Spieler in ihrer Spielhälfte aufhalten müssen. Dies ist vor allem bei den Stürmern problematisch, da ihre Standardpositionen in der gegnerischen Spielhälfte vor dem Tor des Gegners liegen und der Referee die Stürmer nach dem Anpfiff zufällig in der eigenen Spielhälfte positionieren würde. Um das zu verhindern gibt es eine Startposition, die für die Mittelfeldspieler und Stürmer direkt an der Mittelfeldlinie liegt. Nach dem Anpfiff suchen diese Spieler dann ihre Standardpositionen in der gegnerischen Spielhälfte auf. In der Abb. 2.2 ist das Spielfeld mit den Spielern auf ihren Standardpositionen abgebildet. Die farbigen Boxen geben die Zuständigkeitsbereiche der Spieler an: schwarz für den Torwart, blau für die Verteidigung, gelb für das Mittelfeld und rot für den Sturm.

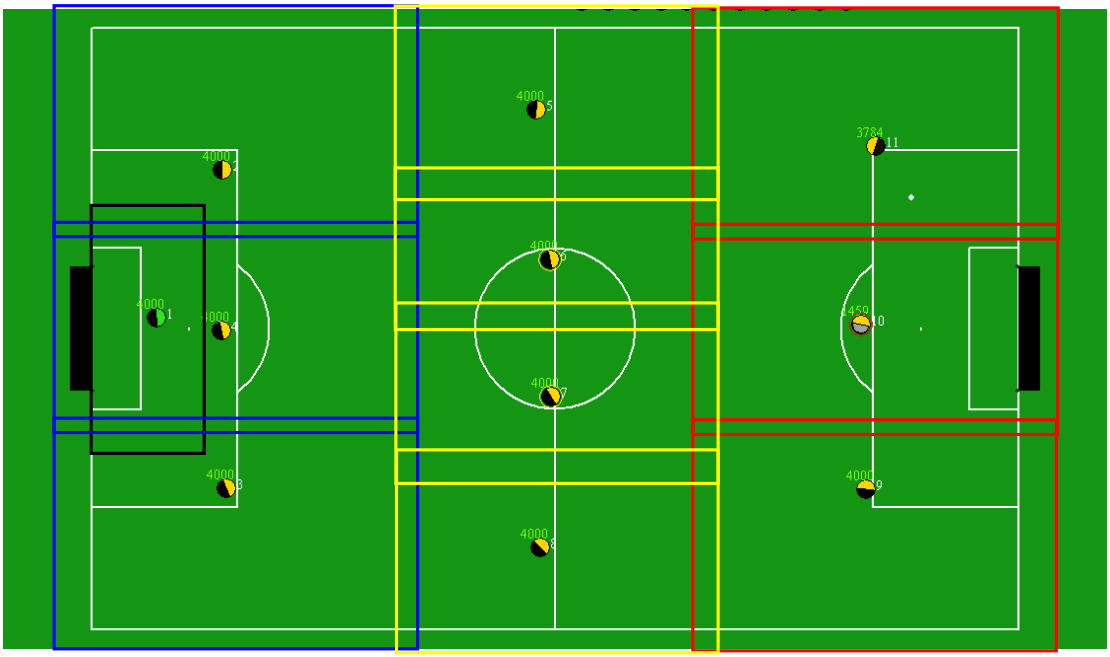

Abb. 2.2: Spielfeld mit Zuständigkeitsbereichen

Kommunikation

Die Kommunikation zwischen den Agenten stellt sich als schwierig dar, da der hörbare Bereich einer gesprochenen Nachricht auf 50 m beschränkt ist, was ungefähr der Länge einer Spielhälfte (52,5 m) entspricht. Somit werden nicht alle Spieler durch eine Mitteilung erreicht. Ein weiteres Problem besteht darin, dass in jedem Zeitzyklus nur genau eine Nachricht gesprochen werden darf. Überlagern sich zwei oder mehrere Nachrichten in ei-

nem Zeitzyklus im gleichen hörbaren Bereich, dann wird vom Server zufällig eine Nachricht ausgewählt, die der Spieler erhält. In unserer Implementierung sendet nur derjenige Spieler Nachrichten, der im Ballbesitz ist. Dadurch werden Kollisionen von Nachrichten ausgeschlossen und der Spieler kann in jedem Zyklus eine Sprach-Mitteilung (`say`-Message) abgeben. Im Ballbesitz ist ein Spieler genau dann, wenn der Ball innerhalb seiner Reichweite liegt (die Reichweite kann über Parameter festgelegt werden, siehe Kapitel 2.4). Nachteil dieser einfachen Kommunikation ist, dass dadurch z. B. die Abwehr nicht durch einen zentralen Spieler (z.B. Torwart) organisiert werden kann.

Pass- und Kick-Strategie

Jeder Spieler im Ballbesitz versucht eine freie Anspielstation zu finden. Eine freie Anspielstation ist ein Mitspieler, der durch keinen Gegenspieler gedeckt wird. Frei ist ein Mitspieler genau dann, wenn alle Gegenspieler, die sich in einer ähnlichen Richtung wie der betrachtete Mitspieler befinden, nicht zwischen diesem und dem Ball führenden Spieler stehen. Zusätzlich sind keine Rückpässe erlaubt, d. h. ein Spieler kann nur einen Mitspieler anpassen, der sich näher zum gegnerischen Tor befindet als er selbst.

Findet der Spieler keinen geeigneten Anspielpartner, dann schaltet er in den Dribble-Modus. Im Dribble-Modus versucht der Spieler den Ball zum gegnerischen Tor zu dribbeln. Zwei Ereignisse können dazu führen, dass der Spieler aus dem Dribble-Modus kommt:

1. Der Spieler findet einen freien Anspielpartner, dann wird der Ball zu diesem gepasst.

2. Der Spieler erreicht das Ende seines Zuständigkeitsbereiches. Da der Spieler seinen Bereich nicht verlassen darf, aber auch kein freier Anspielpartner zur Verfügung steht, wird der Ball mit maximaler Kraft in die Richtung des gegnerischen Tors geschossen.

Die Schwäche dieses Algorithmus' offenbart sich vor allem im letzten Punkt. Sinnvoller wäre wahrscheinlich die Strategie zu verfolgen, dass der Spieler weiter Richtung gegnerisches Tor dribbelt oder einfach einen riskanten Pass auf einen Spieler der eigentlich nicht frei steht spielt. Mit dem vorhandenen Algorithmus kommt es häufiger zu vermeidbaren Ballverlusten

Defensiv-Strategie

Die Defensiv-Strategie ist relativ einfach gestaltet. Es handelt sich hierbei um eine Raumdeckung. Jeder Spieler ist für seinen Spielzonenbereich verantwortlich. Kommt ein Ball in

seinen Bereich, dann geht der Spieler direkt zum Ball. Dabei ist es irrelevant, ob ein Gegenspieler in Ballbesitz ist oder ob der Ball alleine in seinen Bereich rollt. Hat der Spieler den Ball in seiner Kick-Reichweite, dann schaltet er in den Pass-Modus und sucht eine freie Anspielstation.

Alternativ-Mannschaft

Zusätzlich zu unserer offiziellen Mannschaft haben wir noch eine Alternativ-Mannschaft aufgestellt, die nur aus einem Spieler-Typen und dem Torwart besteht und einen sehr simplen Algorithmus besitzt. Die Kernidee zu dieser Alternativ-Mannschaft war es, zu testen, wie gut ein einfacher Algorithmus im Vergleich zu einem komplexen System abschneidet.

Das Verhalten dieser Alternativ-Mannschaft ist ähnlich zu der des „Simple Client“. Jeder Spieler dieser Mannschaft versucht den Ball zu finden, darauf zuzulaufen und den Ball Richtung gegnerisches Tor zu schießen. Da die Orientierungs-Funktion verwendet wird, muss allerdings im Gegensatz zum „Simple Client“ nicht immer wieder nach dem Tor gesucht werden. Eine Kommunikation bzw. ein Mannschaftsspiel mit Passen und Pass-Annahme findet nicht statt. Es hat sich gezeigt, dass diese einfachen Algorithmen im Vergleich zu komplexeren Spielmannschaften nicht schlecht abschneiden.

2.4 Steuerung des Verhaltens über Parameter

Die Steuerung des Mannschafts- und Einzelspielerverhaltens geschieht über einige Parameter, die zentral in der Player-Klasse als Instanzvariablen gesetzt werden. So lassen sich durch kleine Modifikationen im Quellcode große Änderungen in der Spiellogik erzielen.

Wozu Parameter gesteuertes Verhalten

Bei der Methode, die den Spieler dazu veranlasst zu einem Punkt zu rennen, fiel auf, dass er zu schnell auf den Punkt zu lief und über das Ziel hinausschoss bzw. nach einer Anpassung zu früh und stark abbremste und nur sehr langsam zum Zielort kam. Um das rechtzeitige Abbremsen zu verbessern führten wir eine Variable ein, die den Bereich in Metern angibt, ab der der Spieler nicht mehr mit maximaler Beschleunigung auf den Ball zu laufen sollte. So konnten wir diese Variable, die an einigen Stellen verwendet wird, global verändern und nach einigen Tests so anpassen, dass er sich einigermaßen sinnvoll auf den Ball zu bewegt. Nach diesen guten Erfahrungen, habe wir versucht alle Verhaltens steuernden Algorithmen durch globale Instanzvariablen zu optimieren, um in späteren Tests – die lei-

der aus Zeitmangel nicht stattfanden – eine geeignete Parameterkonfiguration zu finden. Im Folgenden sind einige beispielhafte Parameter aufgeführt.

Faktor zum Abbremsen vor Zielpunkt

Dieser Parameter steuert ähnlich wie der oben beschriebene das Zulaufen auf ein Ziel. Sobald der Bereich zum „Abbremsen“ vor dem Ziel erreicht ist, wird nur noch mit einer Kraft beschleunigt, die sich aus einem Faktor und der Entfernung zum Ziel ergeben. Dieser Faktor liegt in der momentanen Konfiguration bei 20, was wahrscheinlich ein zu niedriger Wert ist.

Radius des Zielpunktes

Bekommt ein Spieler die Anweisung auf seine Standardposition zu laufen, wird ihm die x- und y-Koordinate des Punktes angegeben. Problematisch erwies sich dieser Algorithmus im Spielverlauf, da die Spielerposition durch die Spielfeld-Orientierung nicht exakt berechnet werden kann und abhängig von den verwendeten Spielfeldmarkierungen schwanken kann. Es passierte ursprünglich oft, dass der Spieler auf dem angegebenen Punkt ankam, sich dann nach dem Ball drehte und anhand der neuen Spielfeldmarkierungen in seinem Sichtfeld seine Position neu berechnete. Die neue Positionsberechnung stimmte aber nicht mit dem Punkt überein, auf dem er sich befinden sollte. Er bekam daraufhin wieder die Anweisung auf seinen Punkt zu laufen, was ein weiteres `turn`-Kommando zur Folge hatte. In den schlimmsten Fällen drehte sich der Spieler unfähig andere Kommandos auszuführen ständig hin und her.

Durch eine Variable, haben wir den Zielpunkt zu einer Zielzone gemacht, die einen Kreis um den Zielpunkt mit einem variablen Radius bildet. Der Parameter für den Radius ist momentan 2 Meter, was bei den bisherigen Spielen zu sinnvollen Ergebnissen führte.

Start- und Standardpositionen, Zuständigkeitsbereiche

Die Positionen, auf denen sich der Spieler bei Start des Spiels und standardmäßig im Verlauf des Spiels befindet, lassen sich ebenso leicht und schnell verändern. Hier liegt noch enormes Potential der Mannschaft, denn weder die Positionen noch die Zonen sind auf Optimalität hin geprüft worden, sondern befinden sich noch im Zustand einer ersten Startkonfiguration. Sinnvoll wäre sicherlich eine größere Überlappung der Zonen und eine Verlagerung des Mittelfeldes und Angriffs nach hinten Richtung eigenes Tor.

Winkel für Ballsuche

Wenn der Spieler den Ball nicht sieht, bekommt er das Kommando, sich solange zu drehen, bis er ihn wieder in seinem Sichtfeld hat. Um wie viel grad er sich dabei dreht hängt[5] von seinem aktuellen Sichtwinkel ab. Bei der Festlegung des Suchwinkels muss zum einen bedacht werden, dass der Dreh-Winkel nie exakt erreicht wird und mit einer zufälligen Abweichung berechnet wird, aber auch so wenig `turn`-Kommandos wie möglich ausgeführt werden sollen. Bei einem normalen Sichtwinkel von 90° beträgt der optimale Suchwinkel wahrscheinlich zwischen 82° und 88°. Die aktuelle Parameterkonfiguration geht von 85° aus, müsste aber noch auf Optimalität getestet werden.

Torwartparameter

Der Torwart befindet sich auf einer gedachten Linie zwischen Tormittelpunkt und dem Ball vor seinem eigenen Tor. Wie groß dieser Abstand zwischen ihm und dem Tormittelpunkt ist, wird ebenfalls über einen Parameter gesteuert. Zurzeit beträgt dieser 5 Meter, es würde aber sicherlich Sinn machen, den Torwart zu testen, wenn er weiter oder näher vor dem Tor steht.

Auch die Zone, in der sich der Ball befinden muss, damit der Torwart auf ihn zuläuft und versucht ihn zu fangen, muss noch durch Tests optimiert werden. In der aktuellen Einstellung ist sie fast so groß wie der 16-Meter-Raum um das Tor. Tests mit kleineren Zonen könnten zu besseren Ergebnissen führen.

2.5 Erläuterung der Implementierung anhand von ausgewählten Beispielen

Im Folgenden sollen drei Beispiele aufgeführt werden, die unser Programmiervorgehen besonders gut veranschaulichen. Das erste Beispiel ist die Orientierung des Spielers auf dem Spielfeld, deren Perfektionierung viel Entwicklungszeit gekostet hat. Hierauf wird der Dribble-Algorithmus vorgestellt, der neben der Umsetzung der Orientierung und der Entwicklung der GUI im Mittelpunkt unserer Bemühungen stand. Das dritte Beispiel dient dazu, um Schwächen im Verhalten der Spieler – in diesem Fall des Torwartes – zu dokumentieren, die wegen des Zeitmangels nicht behoben werden konnten.

[5] Der Sichtwinkel kann über das `change_view`-Kommando eingeschränkt bzw. erweitert werden. Der Spieler bekommt dann öfters bzw. seltener visuelle Informationen. Vgl. Noda(2002), S. 75.

Orientierung

Ein Hauptbestandteil der gesamten Logik ist die Positionsberechnung des Spielers. Er benötigt sie z. B. dann, wenn er auf einen Punkt des Spielsfelds wie bspw. seine Standardposition laufen muss oder auf das Tor schießen muss, dass sich gerade nicht in seinem Sichtradius befindet.

Um die Lokalisierung möglich zu machen, wird ein Koordinatensystem über das Feld gelegt. Der Koordinatenursprung ist der Mittelpunkt. In Abbildung 2.3 wird veranschaulicht, wie das Feld in die Quadranten des Koordinatensystems eingeteilt wird. Für die Eckpunkte der Strafräume werden beispielhaft die Koordinaten angegeben. Alle statischen Punkte des Feldes werden mit ihren festen Koordinaten im Quellcode gespeichert.

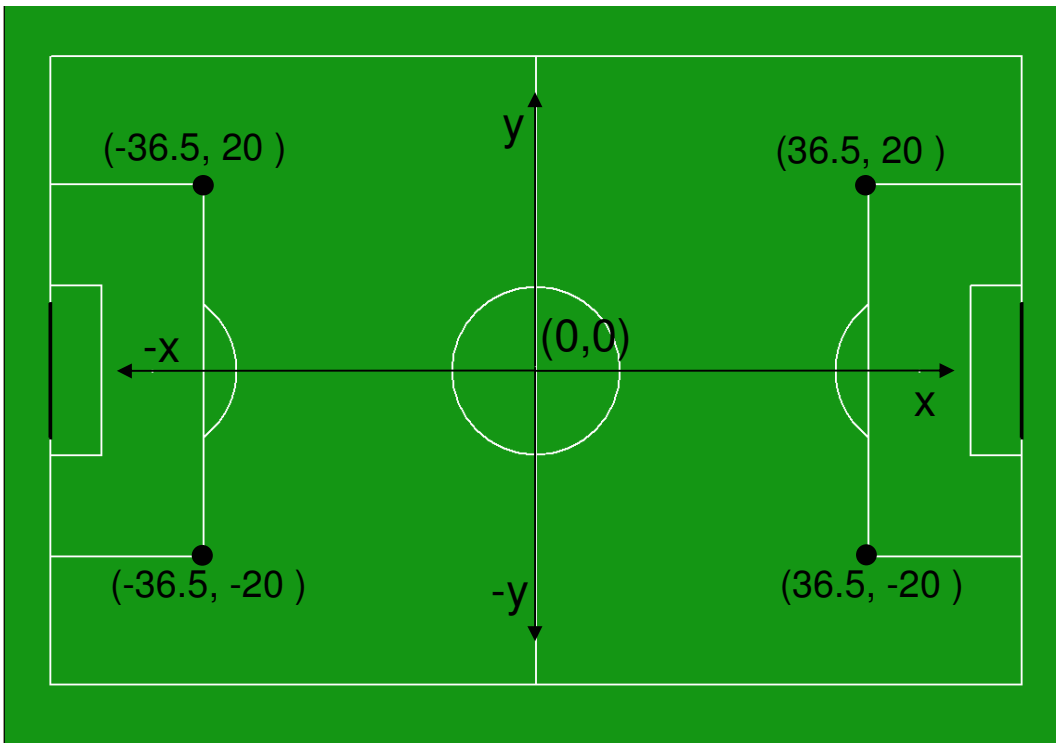

Abb. 2.3: Das Spielfeld als Koordinatensystem

Der Spieler bekommt vom Server eine Nachricht, in der alle Spielfeldmarkierungen, die sich in dem Sichtkegel des Spielers befinden, mit relativer Richtung im Verhältnis zur Körperausrichtung und Entfernung aufgeführt sind. Sieht der Spieler eine Spielfeldrandlinie, erhält er als Information die Entfernung zum Schnittpunkt seiner Sichtlinie und der Randlinie und den Winkel, der zwischen der Randlinie und der Sichtlinie liegt. Eine Nachricht, die der Spieler erhält, könnte wie folgt aussehen:

`(see 174 ((f r b) 44.7 1) ((f p r b) 22.9 1) ((f g r b) 33.8 -37) […] ((b) 36.6 1) ((l r) 44.3 50))`[6].

Aus diesen Informationen wird zuerst die Blickrichtung[7] des Spielers ermittelt. Wenn der Spieler eine Linie sieht, so ist diese Berechnung trivial. Die Ausrichtung der Linie muss von dem übergegebenen Winkel einfach addiert bzw. subtrahiert werden und der Spieler erhält seine Blickrichtung. Sieht der Spieler keine Linie, so wird der Winkel durch das Wissen über Entfernung, Richtung und absolute Position der zwei nächsten Punkte, die sich in seinem Sichtfeld befinden, berechnet. Ist die Blickrichtung bekannt, so genügt ein Punkt bzw. die Information über dessen relative Lage und absolute Position im Koordinatensystem, um die Koordinaten des Spielers zu bestimmen. Sobald die Koordinaten des Spielers berechnet sind, kann von diesen aus unter Berücksichtigung seiner Blickrichtung die Entfernung und die relative Richtung zu jeder anderen statischen Markierung oder zu jedem Punkt auf dem Koordinatensystem ermittelt werden.

Dribbeln

Die Herausforderung beim Programmieren eines guten Dribble-Algorithmus' ist die Berechnung, mit welcher Kraft der Spieler den Ball beschleunigt. Tut er dies zu stark, kann es passieren, dass der Abstand zwischen ihm und dem Ball zu groß wird und er den Ball an einen Gegner verliert. Ist die Kraft zu schwach, wird er sich insgesamt viel zu langsam bewegen und dem Gegner Zeit geben, seine Abwehr zu organisieren.

Grundsätzlich sollte das Dribbeln so gestaltet werden, dass abwechselnd in einem Zyklus geschossen und im nächsten mit maximaler Kraft gelaufen wird. Der Algorithmus ist so angelegt, dass der Spieler, wenn der Ball nicht in seiner Kick-Reichweite ist, zu diesem hinläuft. Im Idealfall ist er dann im nächsten Zyklus bei dem Ball und kann ihn weiter dribbeln. Damit dies geschehen kann, muss der Ball in einem Zyklus so beschleunigt werden, dass der Spieler anfangs der übernächsten Simulationsrunde den Ball wieder in Schuss-Reichweite vor sich hat.[8]

Um diese Kraft zu ermitteln, berechnet der Spieler ausgehend von seiner aktuellen Position und Geschwindigkeit die Position, auf der er sich nach dem Ausführen des `kick`-Kommandos in dieser und des `dash`-Kommandos in der nächsten Runde wahrscheinlich befindet. Zu dieser Position rechnet er noch den Abstand, in dem sich der Ball vor ihm

[6] Folgende Codierung liegt vor: `(see time ((ObjektName) Entfernung Richtung))`
[7] Als 0°-Blickrichtung wird der gerade Blick nach oben Richtung oberer Spielfeldrand definiert.
[8] Die auf den Schuss folgende Runde wird für das Laufen verwendet.

befinden soll[9] und hat damit die Stelle ermittelt, auf der der Ball idealer Weise nach zwei Zyklen liegen soll.

Daraufhin berechnet er den Weg, den der Ball von seiner aktuellen zu dieser gewünschten Position zurücklegen soll. Ist der Weg bekannt, kann daraus und aus den Daten über die aktuelle Ballgeschwindigkeit und unter Berücksichtigung der Trägheit des Balles die Beschleunigung hergeleitet werden, die der Ball erfahren muss, damit er den Weg in zwei Runden zurücklegt.

Für diese Beschleunigung wird im letzten Schritt des Algorithmus berechnet, wie groß die erforderliche Schusskraft des Spielers sein muss. Diese hängt von seiner relativen Position zum Ball, seiner verfügbaren Ausdauer und einiger Server-Parameter ab, die das Schießen beeinflussen. Diese Kraft wird letztendlich als Schusskraft an den Serverübermittelt.

Torwart-Verhalten

Im Verhalten des Torwarts werden einige Schwächen sichtbar, die durch mangelndes Testen und Anpassen zu erklären sind. Er befindet sich wie oben beschrieben immer auf einer gedachten Linie zwischen Ball und Tormitte. Von der Tormitte aus steht er immer 5 Meter vor dem Tor. Effektiv bewegt er sich also auf einer Halbkreisbahn mit dem Radius 5 Meter und der Tormitte als Mittelpunkt.

Tritt der Ball in eine festgelegte Zone vor dem Tor ein, rennt der Torwart darauf zu und versucht den Ball zu fangen. Nach einer erfolgreichen Parade wird er mittels des `move`-Kommandos an die untere Ecke seines 16-Meter-Raums bewegt und macht von dort aus den Abschlag Richtung Spielfeldmitte.

Schwächen hat der Algorithmus beim Bewegen auf der beschriebenen Linie und beim Versuch, den Ball zu fangen. Die erste Schwäche lässt sich durch einen zu häufigen Positionswechsel auf der beschriebenen Linie erklären. Denn damit ist verbunden, dass sich der Torwart zu oft vom Ball wegdreht und dabei keine aktuellen Informationen über den Ball erhält und nach dem Erreichen seiner Position, den Ball erst wieder suchen muss. Insgesamt sind dadurch zu viele `turn`-Kommandos nötig, die den Torwart in gefährlichen Situationen handlungsunfähig machen können.

Die zweite Schwäche entsteht durch das Laufen in Ballrichtung, während der Ball – in der Regel mit hoher Geschwindigkeit – dem Torwart entgegenkommt. So passiert es häufig, dass der Torwart über den Ball läuft und ihm dieser so entgeht.

[9] Es ist sinnvoll, denn Ball stets ungefähr 50 cm vor sich zu führen.

Beide Schwächen ließen sich aber durch mehr oder weniger kleine Anpassungen im Quellcode beheben und wären auch schon behoben worden, hätten wir mehr Zeit zum Testen und Evaluieren zur Verfügung gehabt.

2.6 Nicht umgesetzte Erweiterungen

Durch die Konzentration auf Kernfunktionen wie die Orientierung auf dem Spielfeld oder das Dribbeln, wurden einige Features und Funktionen nicht umgesetzt, die für das Verhalten des Spielers zum Teil sehr wichtig wären und konzeptionell auch schon durchdacht und eingeplant waren.

Mitteilung der Ballposition durch den Ball führenden Mitspieler

Eine Schwachstelle in der Logik der einzelnen Spieler ist die Abhängigkeit vom Wissen der aktuellen Ballposition. Die meisten Entscheidungen werden auf Basis des Abstandes und der relativen Richtung zum Ball, bzw. seiner aktuellen Position getroffen. Ein Beispiel hierfür ist das Verhalten des Torwarts, der, wie bereits beschrieben, seine aktuelle Position immer anhand der Ballposition ermittelt. Dementsprechend wichtig ist das Wissen über die Position des Balles. Wenn sich nun aber der Ball außerhalb des Sichtbereiches befindet, muss sich der Spieler so lange nach dem Ball umdrehen, bis er diesen wieder im Sichtfeld hat und so die aktuelle Ballposition berechnen kann. Diese Suche kann im worst-case[10] solange dauern, bis der Spieler durch `turn`-Kommandos fast eine vollständige Drehung vollführt hat.

Um dieses aufwendige Suchverhalten zu vermeiden, wäre es sinnvoll, wenn der Spieler, in dessen Zone der Ball ist, an alle Spieler die Position des Balles weitergäbe. Damit kann zumindest erreicht werden, dass alle Spieler, die sich in einem Radius von ca. 50 Metern um den Ball befinden, über dessen Position informiert sind. Konflikte in der Übertragung von Sprachmitteilungen[11] innerhalb unseres Teams durch zwei gleichzeitig gesendet Nachrichten treten nur dann auf, wenn sich der Ball im Grenzbereich der Zuständigkeitszonen zweier Spieler befindet. Dies stellt allerdings kein Problem dar, da der Inhalt der Nachrichten gleich ist[12]. Will ein Spieler passen, so setzt er einmal mit der Ballpositions-Mitteilung aus und sendet eine „Passen“-Nachricht an den gewünschten Passempfänger.

---

[10] Der worst-case liegt dann vor, wenn sich der Ball gerade nicht mehr im Sichtbereich des Spielers befindet und der Spieler sich aufgrund der Vergangenheitsdaten des Balles in die falsche Richtung dreht.

[11] Mit *Sprachmitteilungen* ist die Kommunikation zwischen den Spielern mit Hilfe des `say`-Kommandos gemeint.

[12] Werden zu einem Zeitpunkt von Spielern eines Teams zeitgleich `say`-Nachrichten gesendet, wählt der Server eine Nachricht aus und leitet diese an die anderen Spieler des Teams weiter

Befindet sich der Ball zudem noch in unmittelbarer Reichweite[13] des Spielers, wird in der Nachricht über die Ballposition auch mitgeteilt, dass sich der Ball in Besitz der Mannschaft befindet. Hierdurch wird zum einen im Fall, dass sich der Ball in der Überschneidung der Zuständigkeitszonen zweier Spieler befindet, dem einen Spieler signalisiert, dass der andere Spieler bereits am Ball ist, zum anderen lässt sich die gesamte Taktik durch die Information über den Ballbesitz beeinflussen.

Durch diese Mitteilungen kann die Effizienz der Spielzüge enorm gesteigert werden, da für die Spieler in der Nähe des Balles das Suchen nach diesem komplett entfällt. Dieses Feature wurde schon soweit eingebaut, dass der Ball besitzende Spieler eine Nachricht an seine Mitspieler sendet. Diese wird aber von den anderen noch nicht ausgewertet.

Freilaufen und Decken

Bei den durchgeführten Tests, vor allem bei solchen in denen die Mannschaft gegen sich selbst spielt, zeigt sich eine weitere Schwäche. Oft fanden die Spieler in Ballbesitz keine Anspielstelle für Pässe, da kein Mitspieler frei stand. Dieser Schwäche würde sich dadurch begegnen lassen, dass die Mitspieler versuchen sich freizulaufen. Diese Funktion würde das schnelle Spiel nach vorne positiv beeinflussen.

Im Gegenzug sollten natürlich die Verteidiger und Mittelfeldspieler stets versuchen, sich bei gegnerischem Ballbesitz zwischen den Ball und einen zu bewachenden Gegenspieler zu stellen oder zumindest so knapp dahinter zu stehen, dass sowohl Gegner als auch Ball im Sichtfeld sind, aber ein Pass gut abgefangen werden könnte. Ein Hinzufügen dieser „Decken"-Funktion wäre eine enorme Verstärkung des Abwehrverhaltens.

Angepasstes Angriffs-Verhalten

Die folgenden drei Funktionen sind für die Verbesserung des Verhaltens nicht elementar, wurden anfangs aber doch als fester Bestandteil unserer Mannschaftslogik bzw. graphischen Benutzeroberfläche eingeplant und daher auch ausführlich durchdacht und konzeptionell entwickelt.

[13] Denkbar wäre ein Wert von drei Metern, da der Spieler beim Dribbeln in Ausnahmefällen den Ball soweit vor sich her schießt.

Ein leicht einzubauendes Feature wäre die Möglichkeit gewesen, über einen Online-Coach[14] das Angriffsverhalten der Mannschaft zu steuern. Dieser kann alle 300 ms eine Nachricht an die Spieler senden. Da diese Nachricht jedoch den normalen Regeln der Kapazität von gehörten Nachrichten unterworfen ist[15], muss sichergestellt werden, dass in diesen Zyklen keine anderen Nachrichten ausgetauscht werden – eine kleine Einschränkung, die leicht in Kauf genommen werden kann.

Über die Nachrichten, die der Coach sendet, könnte er den Spieler entweder im Falle einer drohenden Niederlage eine offensivere Mannschaftsaufstellung oder, wenn sich ein Erfolg abzeichnet, eine defensivere Aufstellung vorgeben. Dies würde durch eine Verlagerung der Standardpositionen Richtung gegnerisches oder eigens Tor geschehen. Dadurch, dass die Mannschaft dann dichter zusammensteht, könnten Lauf- und Passwege verkürzt und optimiert, weniger Ausdauer verbraucht und die Pässe der Gegner besser abgefangen werden. Allerdings muss entweder das Risiko im defensiven Bereich in Kauf genommen werden, da hier im offensiven Modus die zu schützenden Räume größer sind, oder der Mangel an eigenen Torchancen im defensiven Modus.

Ausbau der GUI

Die entwickelte graphische Oberfläche hatte in der ursprünglichen Planung noch über die existierenden Funktionen hinaus eine viel mächtigere Funktionalität. Eigentlich war geplant eine Oberfläche für die Verwaltung der Mannschaft zu entwickeln, von der aus die Oberflächen mit den Informationen zu den einzelnen Spielern gestartet bzw. beendet werden kann. Außerdem hätte über diese Mannschafts-GUI die Aufstellung durch Festlegen der Standardpositionen möglich sein sollen[16]. So könnte man zum einen die oben beschriebenen defensiven bzw. offensiven Verhalten testen, zum anderen aber auch ganz andere Möglichkeiten der Mannschaftsaufstellung und Zonenverteilung zur Auswahl geben. Denkbar wären hier aus dem realen Fußball adaptierte Möglichkeiten wie eine Vierer-Abwehrkette oder eine Mittelfeld-Raute.

Außerdem wäre es schön gewesen, mit der Offline-Coach-Funktionalität die aktuellen Spielmodi des Servers zu beeinflussen und auf diese Art besondere Spielzüge bei Stan-

[14] Der Online-Coach ist ein weiterer Agent, der die Rolle des Trainers während des Spiels übernimmt. Er erhält vom Server alle exakten Positionen und Informationen über den Spielstand. Er kann Nachrichten senden, die alle Spieler zur gleichen Zeit erhalten und so die Mannschaftstaktik und Einzelaktionen bestimmen. (Vgl. Noda (2002), S. 85f.)

[15] Jeder Spieler kann pro Simulationsrunde nur eine Nachricht seiner Teammitglieder (Spieler und Coach) hören. Wenn mehrere Nachrichten von Mitgliedern seines Teams gesendet werden, wird vom Server zufällig eine ausgewählt, die er dann hört.

[16] Dieses Feature darf nur zu Trainingszwecken und nicht für eigentliche Spiele verwendet werden, da dies gegen das FairPlay-Commitment der Regeln verstoßen würde. Vgl. o. V. (2004), S. 4.

dardsituation wie Ecke, Anstoß, Abschlag oder Freistoß gezielt zu testen und zu optimieren.

Es ist bereits möglich den Spieler vom Server über einen Button in der Oberfläche abzumelden, ohne seine Daten in der GUI zu verlieren. Schön wäre die Ergänzung, ihn über die GUI auch wiederanmelden zu können

Komplexere Spielzüge

Einfache Spielzüge wie Passen und Dribbeln sind in der bisherigen Strategie bereits umgesetzt. Schön wäre eine Erweiterung um komplexe Spielzüge und Strategien wie das Doppelpassspiel oder Spielzüge über mehrere Anspielstationen. Außerdem sollten einige alternative, aber fest programmierte Spielzüge für Standardsituation wie Eckstoß, Abschlag, Freistoß oder Anstoß angelegt werden. So dass in solchen Situationen zufallsgesteuert aus einer Menge an Spielzügen gewählt werden kann. Dieses Vorgehen ist äquivalent zum echten Fußball, wo für Ecken und Freistöße Spielzüge einstudiert werden und die Spieler je nach Situation die Spielzüge auswählen.

Scoring-Modell

Im Zusammenhang mit den festgelegten Spielzügen sind unsere Überlegungen zu einem Scoring-Modell zu nennen. Mit der Verwendung dieses Modells sollte ermöglicht werden, während eines Spieles zu lernen und auf die Taktik des Gegners einzugehen. Dies betrifft in erster Linie zwar die oben erwähnten festen Spielzüge, ließe sich aber auch auf freie und sich zufällig ergebende Spielzüge adaptieren.

Erfolgreich Spielzüge werden mit einem positiven Faktor bewertet und Spielzüge, die nicht zum Torerfolg führen, dementsprechend mit einem negativen. Wenn nun eine Standardsituation ansteht und die Mannschaft einen der Spielzüge wählen muss, so wird der Spielzug mit der besten Bewertung gewählt. Freie Spielzüge, also solche, die keinem festen Schema folgen, werden ebenfalls nach Torerfolg bewertet und jeder Spieler, der Teil des Spielzuges war[17] merkt sich, wer der Empfänger seines Passes war. Je nach Erfolg wird dieser Empfänger dann mit einem Faktor bewertet und im weiteren Spielverlauf öfters mit Pässen bedient oder eher ignoriert.

Auch diese Strategie ist aus dem echten Fußball adaptiert. Gute Mannschaften stellen sich in der Regel auf den Gegner ein und versuchen ihre Spielzüge dynamisch anzupassen.

[17] Denkbar wäre, dass ein freier Spielzug als Reihe von Passstationen definiert wird und bei erfolgreicher oder verhinderter Torchance die letzten fünf Stationen zum Spielzug zählen.

3 Fazit

Die Entwicklung einer Agenten-Mannschaft für den SoccerServer kann einen idealen Einstieg bilden, um sich mit Künstlicher Intelligenz und Mulit-Agenten-Systemen auseinandersetzen. Hier kann auf spielerische Art Studiums-Wissen in der Praxis erprobt werden. Die Simulation ist sehr komplex und versucht ein reales Fußballspiel so gut wie möglich abzubilden. Die Regeln des echten Fußball wurden weitest gehend adaptiert. Daher kann man sich sehr schnell in die Regeln der RoboCup Simulationsliga zurechtfinden. Das Entwickeln eines Teams für den SoccerServer kann gerade in der Gruppe für viel Spaß sorgen, da Anpassungen der Spieltaktik und der Logik sofort und ohne Aufwand im realen System getestet werden können und man so im Idealfall durch Erfolgserlebnisse positiv motiviert wird.

Allerdings ist der Versuch, möglichst viel Komplexität in der Simulationsumgebung abzubilden, gerade für Einsteiger eine große Eintrittsbarriere. So musste für ein scheinbar einfaches Manöver wie das Dribbeln mit der Trägheitsformel für den Spieler, der Trägheitsformal für den Ball, der Berechnung für die Schusskraft unter Berücksichtigung vieler Parameter und der Berechnung für die Laufkraft gleich vier höchst fehleranfällige Gleichungen aufgestellt und gelöst werden. Das gleiche galt für die Ermittlung der absoluten Position des Spielers, die durch die wenigen visuellen Informationen sehr aufwendig und ebenso fehleranfällig ist. Wünschenswert wäre hier zum einen eine bessere Dokumentation des Servers und zum anderen eine Art Anfänger-Modus, in dem die Spieler ihre exakten Positionen direkt vom Server übermittelt bekommen und in dem von der Trägheit der Spieler und des Balles abstrahiert wird.

Wir haben uns bei der Entwicklung auf die wesentlichen Kernmodule wie die Orientierungs-Methode, die Spieler-GUI und das Dribbeln beschränkt. Das hatte den Vorteil, dass wenigstens diese Module mehr oder weniger gut funktionierten. Außerdem wurde das Minimal-Ziel, eine Mannschaft für den SoccerServer zu entwickeln, erreicht. Die Mannschaft kommuniziert miteinander und die Spieler verhalten sich in Ansätzen intelligent.

Allerdings ist das Verhalten noch von einige Schwächen und Fehlern durchzogen. Oft sind es nur kleine Änderungen, die im Quellcode vorgenommen werden müssen oder Parameter, die angepasst werden müssen. Die Zeit, diese Verbesserungen am Ende vorzunehmen, wurde allerdings in der der Anfangsphase der Entwicklung verschenkt, als wir orientierungslos und mit wenig Plan versuchten, einem einzigen Spieler intelligentes Verhalten beizubringen. Diesen Fehler würden wir bei einer Wiederholung des Projektes sicher nicht wiederholen.

Literaturverzeichnis

Noda, Isuki et al.: RoboCup Soccer Server for Soccer Server Version 7.07 and later. Users Manual. 2002. http://sserver.sourceforge.net/docs/manual.pdf. Abrufdatum: 19.01.2005

Russel, S.; Norvig, P.: Artificial Intelligence – A Modern Approach. New Jersey 2003.

Wooldridge, M.: An Introduction to Multi-Agent Systems. Chichester 2002.

O.V.: Rules 2D Simulator League RoboCup-2005 Osaka. 2004. http://staff.science.uva.nl/%7Ejellekok/robocup/rc05/rules_2d.pdf. Abrufdatum: 19.01.2005