Vorstellung einer eigenen Mannschaft für den Robocup Soccerserver

Ausarbeitung im Rahmen des Seminars
„Agenten in Simulierten Umgebungen“

Themensteller: Dr. Dietmar Lammers
vorgelegt von: Boris Beumers
Stefanie Filius
Tamer El-Hawari

Abgabetermin: 09.01.2005

Inhaltsverzeichnis

1. Einleitung

Der RoboCup Simulator ist ein Fußball Simulations System, welches es virtuellen Fußballspielern erlaubt, in einer dynamischen, komplexen, unsicheren Multi-Agenten-Umgebung gegeneinander anzutreten.

Aufbauend auf der Soccer Simulation Version 9.4.5, die die Darstellung und die Umgebung der Agenten übernimmt, haben wir ein Team erstellt. Ziel dieser Arbeit ist es, unser Team die WWUnderdogs vorzustellen. Um den Rahmen dieser Arbeit nicht zu sprengen, werden grundlegende Kenntnisse über die Simulationsumgebung zum Verstehen dieser Arbeit vorausgesetzt. Dazu gehören Kenntnisse sowohl über das Aktionsmodell und das Sensormodell als auch über das Zusammenspiel von Soccer Server, Monitor und Agenten.

Bei der Programmierung unserer Mannschaft orientierten wir uns an realen Fußball-Mannschaften. Sie dienten als Vorbild, um Spieltechniken und Spieltaktiken zu entwickeln. Das ursprünglich geplante Top-Down-Vorgehen konnten wir jedoch nicht beibehalten. Schon bei der Entwicklung eines Konzepts für unsere Mannschaft stellten wir fest, dass unser Kenntnisstand über den Server nicht ausreichte, um die Realisierbarkeit bestimmter Taktiken und Techniken einschätzen zu können. Daher verfolgten wir zunächst eine Bottom-Up-Strategie, um eine geeignete Testumgebung für unsere Ideen zu besitzen. Auf dieser Grundlage implementierten wir, so realitätsnah wie möglich, Spieltechniken, die wir anschließend zu einer Spieltaktik zusammenführten.

Entlang unseres Vorgehens ist der Aufbau dieser Arbeit strukturiert. Das zweite Kapitel beschreibt die Basis eines Agenten, d.h. den Teil, der für die Kommunikation mit dem Server und die Speicherung der Daten zuständig ist. Hier wird zunächst die Architektur eines Agenten vorgestellt. Daraufhin wird beschrieben welche Daten wie lange und in welcher Form gespeichert und welche zusätzlich berechnet werden. Das dritte Kapitel beschreibt die Kommunikation zwischen den Agenten. Im vierten Kapitel werden die realisierten Spieltechniken vorgestellt. Anhand von Pseudo-Code werden die Algorithmen dargestellt. Das fünfte Kapitel beschreibt das Verhalten der Agenten während eines Spiels.

Unsere Mannschaft ist in Java realisiert. Der Quellcode befindet sich im Anhang.

2. Die Basis eines Agenten

2.1. Architektur

Ein Agent muss während eines Spiels Aufgaben auf unterschiedlichen Abstraktionsniveaus wahrnehmen, um erfolgreich agieren zu können. Diese reichen von der Übertragung von Zeichenketten über UDP bis hin zur Beherrschung einer Spieltaktik. Daher bot es sich an, eine Mehrschichtenarchitektur zu wählen, um die Abstraktionsniveaus sauber voneinander trennen zu können. Wir entschieden uns für vier Schichten:

(1) Eine Zeichenkettenübertragungsschicht, welche die Übertragung von Zeichenketten über UDP mit dem Server leistet.

(2) Eine Nachrichtenübertragungsschicht, welche Nachrichten an den Server codiert und decodiert.

(3) Eine Technikschicht, die Grundtechniken bereitstellt, die im Spiel eingesetzt werden sollen.

(4) Eine Taktikschicht, welche die Reaktionen auf bestimmte Spielsituationen festlegt.

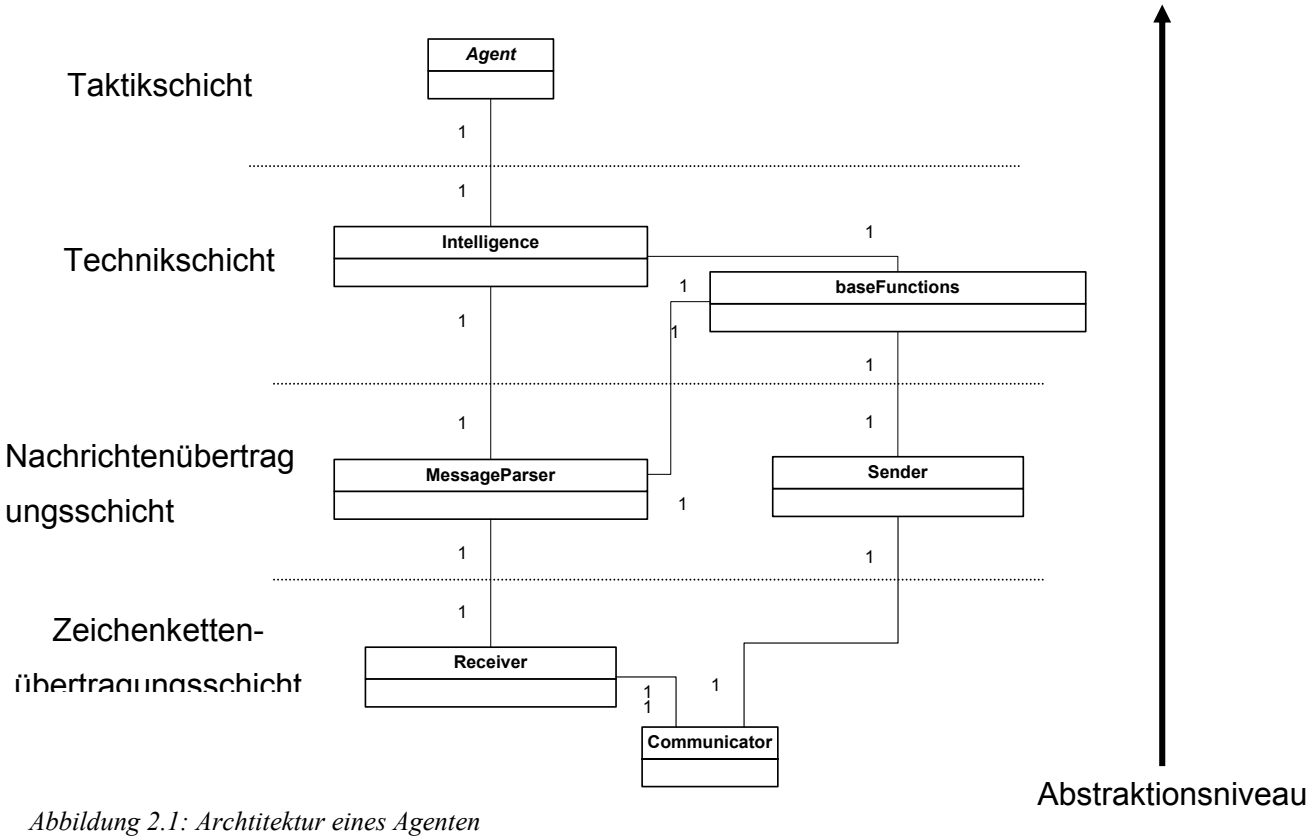

Abbildung 2.1: Archtitektur eines Agenten

Abbildung 2.1 zeigt die Schichtenarchitektur mit den zugehörigen Klassen.
Ursprünglich versuchten wir zunächst die Taktik, also die Schicht mit dem höchsten Abstraktionsniveau zu planen. In Schicht (1) realisierten wir zwei Klassen: Den Communicator, der Methoden zur Übertragung von Zeichenketten vom und zum Server bereitstellt, und den Receiver, welcher in einer Dauerschleife die Nachrichten vom Server abfragt.
Schicht (2) enthält die Klassen Sender und MessageParser. Der Sender stellt Methoden bereit, die logische Nachrichten in eine vom Server verstandene Zeichenkette verwandeln. Der MessageParser, decodiert mit Hilfe von Regular Expressions die Informationen aus eingehenden Servernachrichten und veranlasst die Berechnung von verwertbaren Informationen. Methoden aus der Klasse Sender, welche eine Änderung der Position bzw. Spielerrichtung mit sich bringen und nur einmal pro Runde ausgeführt werden können, rufen außerdem die Methode Sender.waitAfterSending() auf. Diese wartet auf die jeweils nächste SenseBody- und See-Nachricht vom Server, um dem Agenten aktuelle Informationen für seine nächste Entscheidung bereitstellen zu können.

```
public void kick(int power, double direction){
    com.send("(kick "+power+" "+direction+")");
    waitAfterSending();
}
```

Codebeispiel 2.1: Sender.kick(power, direction)

Codebeispiel 2.1 zeigt die kick-Methode aus der Klasse Sender, welche z.B. bei einem Aufruf von kick(100, -20) die Zeichenkette “(kick 100 -20)“ an den Communicator übergeben würde und danach mit Sender.waitAfterSending() auf aktuelle Informationen wartet.
In Schicht (3) realisierten wir die Klassen Intelligence und BaseFunctions. Intelligence beinhaltet Methoden zur Berechnung von Spieldaten und stellt komplexe Techniken bereit. BaseFuncitions enthält, einfache Techniken, die normalerweise nicht länger als eine bis zwei Runden dauern.
Schicht (4) schließlich enthält nur die abstrakte Klasse Agent. Durch Erben aus dieser Klasse lassen sich Spieler erstellen, welche unterschiedlich auf verschiedene Spielsituationen reagieren, wodurch die Realisierung einer Mannschaftstaktik ermöglicht wird.

2.2. Relevante Umweltinformationen

Eine schwierige Entscheidung zu Beginn des Entwurfes besteht darin, festzulegen, welches Wissen relevant ist, und wie lange gespeichert werden soll. Des Weiteren muss überlegt werden, wie man aus den eingehenden Sensordaten sinnvolle Informationen über die Umgebung des Agenten gewinnen kann.

Durch Sensoren gewonnene Informationen lassen sich grob in zwei Klassen einteilen: Solche, die nur im aktuellen Zyklus verwendbar sind, und solche, die evtl. auch länger relevant sind. Die Entscheidung, zu welcher Klasse eine Information gehört, ist nicht immer einfach. Dabei hängt auch viel davon ab, ob sich der Aufwand, bestimmte Informationen auszuwerten, überhaupt lohnt. So ist es zwar z.B. evtl. von Nutzen, sich den Weg eines gesehenen Spielers zu merken, um ihn auch noch einem Team zuordnen zu können, wenn er sich weit entfernt, jedoch sind die dazu notwendigen Algorithmen so komplex, dass wir uns dagegen entschieden haben, sie zu realisieren.

Zu den Informationen, die sich unser Team nur im aktuellen Zyklus merkt, gehören die Position des eigenen Spielers, Positionen von Spielern, deren Nummer unbekannt ist, Winkel und Entfernungen von Spielobjekten, sowie der Geschwindigkeitsvektor des Balls. Länger relevant sind hingegen die Ballposition und Positionen von Spielern mit bekannter Nummer. Die Ballposition ist deshalb länger verwertbar, da man alte Ballpositionen dazu verwenden kann, die Suche nach dem Ball zu beschleunigen. Alte Spielerpositionen können wichtig sein, wenn man einen Spieler sucht, den man anspielen kann. Da diese Daten mit zunehmendem Alter fehlerhafter werden, erhalten sie einen Zeitstempel.

Da man häufig die in Form von Winkeln und Entfernungen eingehenden Sensordaten nicht direkt zu Entscheidungsfindung verwenden kann, müssen sie miteinander kombiniert werden, um verwertbare Informationen zu erhalten. Im Folgenden soll gezeigt werden, wie aus den eingehenden Sensordaten verwertbare Umweltinformationen berechnet werden.

2.3. Berechnungen von Spieldaten

Damit ein Spieler sinnvoll auf eine Spielsituation reagieren kann, muss er die zur Verfügung stehenden Daten interpretieren. Es ist dazu häufig nötig, die Sensordaten zu transformieren, um ein verwertbares Bild der Spielsituation zu erhalten.

Die absolute eigene Position

Damit ein Spieler eine ihm zugewiesene Position auf dem Spielfeld einnehmen kann, ist es nötig, die absolute eigene Position überhaupt erst zu kennen. Da sie nicht direkt vom Server übergeben wird, muss sie aus denjenigen gesehenen Spielfeldobjekten berechnet werden, die eine unveränderliche Position bzw. Ausrichtung besitzen. Es kommen daher nur Flaggen und Linien für diese Berechnung in Frage. Wir haben uns für eine Berechnung der Position aus zwei gesehenen Flaggen entschieden. Dazu muss ein Spieler sowohl die Entfernung als auch die Winkel zwischen Blickrichtung und Flaggen kennen.[1]

Als Nullpunkt des Koordinatensystems haben wir den Anstoßpunkt gewählt. In der gegnerischen Hälfte ist die x-Koordinate positiv, in der eigenen ist sie negativ. Wir haben das Koordinatensystem als Linkssystem definiert, da die vom Server zurückgegebenen Winkelmaße im mathematisch negativen Drehsinn zunehmen. Auf diese Weise bleiben Geometrische zusammenhänge gültig.

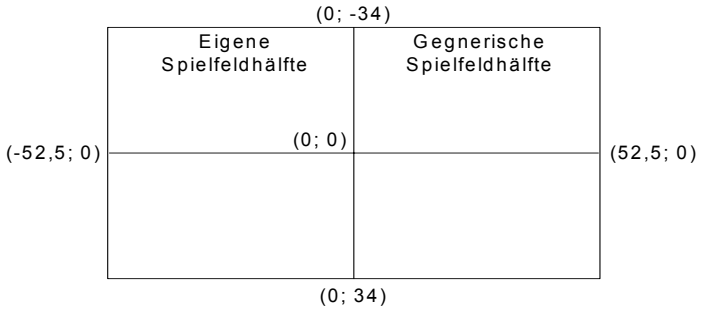

Abbildung 2.2: Koordinatensystem

[1] Es ist daher notwendig, die Sichtqualität auf „high“ zu belassen, da ansonsten ausschließlich die Winkel bekannt wären.

Es sei

F1: die erste Flagge

F2: die zweite Flagge

P: der Spieler

O: Der Ursprung des Koordinatensystems

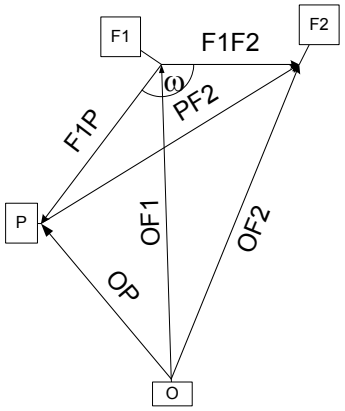

Abbildung 2.3: Berechnung der Absoluten eigenen Position

Vektoren werden hier der Einfachheit halber durch ihren Anfangs- und Endpunkt bezeichnet. F1P ist also z.B. der Vektor von F1nach P

Bekannt sind:

OF1: Der Ursprungsvektor auf Flagge F1, also ihre absolute Position

OF2: Der Ursprungsvektor auf Flagge F2

|F1P|: Der Abstand zwischen Spieler und erster Flagge

|F2P|: Der Abstand zwischen Spieler und zweiter Flagge

α: Der Winkel zwischen Flagge 1 und Sichtrichtung des Spielers

β: Der Winkel zwischen Flagge 1 und Sichtrichtung des Spielers

Die Flaggen werden zunächst so sortiert, dass F1 die linke Flagge im Sichtfeld ist, so dass gilt:

$\alpha < \beta$

Berechnung von F1F2:

F1F2 = OF2 – OF1

Dann Berechnung von ω (siehe Abbildung 2.3):

$$\omega = \arccos\left(\frac{|\,\mathrm{F2P}\,|^2}{|\,\mathrm{F1P}\,|^2 + |\,\mathrm{F1F2}\,|^2 + 2\,|\,\mathrm{F1P}\,||\,\mathrm{F1F2}\,|}\right)$$

Durch die Sortierung der Flaggen ist die Richtung von ω eindeutig.

Berechnung von F1P:

$$F1P = F1F2 \cdot \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos\alpha \end{bmatrix} \cdot \frac{|F1P|}{|F1F2|}$$

Dies entspricht einer Drehung des Vektors F1F2 um den Winkel ω und einer anschließenden Angleichung auf die bekannte Länge |F1P|

Damit errechnet sich die absolute eigene Position über

OP = OF1 + F1P

Die absolute Blick- und Körperrichtung

Um Aktionen wie Drehungen in Richtung des gegnerischen Tors auch dann durchführen zu können, wenn keine zum Tor gehörende Flagge gesehen wurde, ist es nötig, die absolute Blickrichtung und die absolute Körperrichtung zu kennen.

Um überhaupt eine „absolute“ Richtung festlegen zu können, legten wir eine Nullrichtung fest. Diese ist parallel zu den Seitenauslinien, und zeigt in Richtung der gegnerischen Hälfte. Die anderen Winkel sind im Uhrzeigersinn angeordnet:

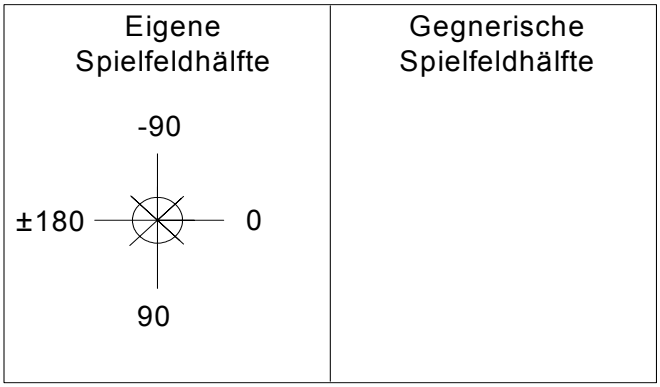

Abbildung 2.4: Die absoluten Richtungen

Wir setzen bei der von uns gewählten Methode zur Berechnung der Richtung das Bekanntsein der absoluten Spielfeldposition voraus. Darüber hinaus wird nur noch die Richtung und Entfernung einer Flagge im Sichtfeld relativ zur Blickrichtung benötigt, wobei ohne Probleme eine der Flaggen verwendet werden kann, die auch schon zur Positionsberechnung benutzt wurde.

Zunächst wird die absolute Blickrichtung berechnet.

Es sei

X_F, Y_F: Die Koordinaten der Flagge

dist(F): Die Entfernung zur Flagge

X_P, Y_P: Die Koordinaten des Spielers

Zunächst haben wir den Hilfswinkel α eingeführt. Er wird berechnet durch:

$$\alpha = \arccos\left(\frac{|\,XF - XP\,|}{dist(F)}\right)$$

Die Absolute Blickrichtung viewDirection ist nun

viewDirection = α, falls ($X_F \geq X_P$) und ($Y_F \geq Y_P$)

$-\alpha$. falls ($X_F \geq X_P$) und ($Y_F \leq Y_P$)

α – 180, falls ($X_F \leq X_P$) und ($Y_F \leq Y_P$)

180 – α, falls ($X_F \leq X_P$) und ($Y_F \geq Y_P$)

Da der Winkel zwischen Körper und Kopf als headDirection per SenseBody – Nachricht vom Server übergeben wird, kann nun die absolute Körperrichtung berechnet werden:

bodyDirection = viewDirection – headDirection

Die absoluten Positionen anderer Objekte

Die absoluten Positionen anderer Spielobjekte sind wichtig, wenn man Fragen folgender Art beantworten möchte:

- Befindet sich ein Gegenspieler im Zuständigkeitsbereich des Spielers?
- Muss der Spieler den Freistoß ausführen, da der Ball in seinem Zuständigkeitsbereich liegt?
- Wie groß ist der Abstand zwischen zwei gesehenen Objekten?

Zur Berechnung wird die absolute eigene Position, die absolute Blickrichtung, die Entfernung zum Objekt sowie der Winkel zwischen Blickrichtung und Objekt benötigt.

Es sei

X_O, Y_O: Koordinaten des Objekts

X_P, Y_P: Koordinaten des Spielers

dist(O): Die Distanz zum Objekt

dir(O): Der Winkel zwischen Blickrichtung und Objekt

α = viewDirection + dir(O)

Dann ist die Position des Objekts

$X_O = \cos(\alpha) * dist(O) + X_P$

$Y_O = \sin(\alpha) * dist(O) + Y_P$

2.4. Speicherung von Objektdaten

Da in den Spielfeldobjekten Ball, Flagge und Spieler ähnliche Daten gespeichert werden, erben sie aus einer gemeinsamen Oberklasse GameObject. Dadurch wird die Speicherung von Daten vereinfacht, da Methoden die Sensordaten oder

Informationen schreiben, auf dieselbe Weise auf diese Objekte zugreifen können. Da für Ball und Spieler außerdem noch die absolute Position schreibbar sein musste, für Flaggen jedoch nicht, da sie konstant ist, haben wir die Get- und Setmethoden hierfür im Interface AbsPosSettable zusammengefasst. Abbildung 2.5 zeigt das zugehörige Klassendiagramm.

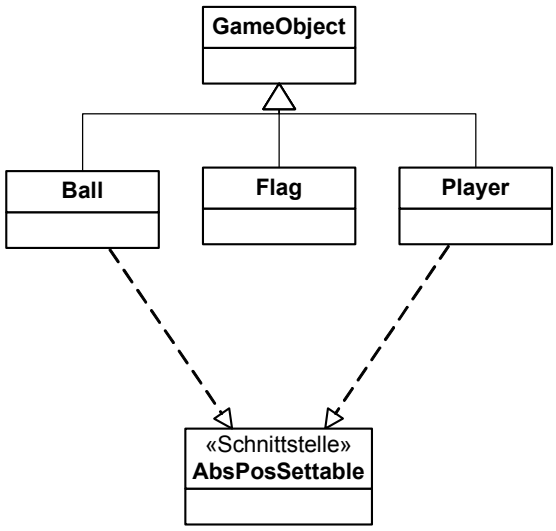

Abbildung 2.5: Klassendiagramm einiger Spielobjekte

2.5. Threadmodell

Der Empfang von Sensordaten und Aktionen müssen parallel ablaufen. Die Entkopplung von Nachrichtenempfang und –versendung ermöglicht es, komplexe Algorithmen in den Schichten höherer Abstraktion zu implementieren, ohne sich dabei um Details des Nachrichtenempfangs kümmern zu müssen.

Wir entschieden uns für eine Aufteilung des Programms in einen Empfangsthread und einen Entscheidungs- und Sendethread. Beide werden bis zur Beendigung des Programms ausgeführt.

Im Folgenden werden anhand zweier Beispiele die Vorgänge innerhalb der Threads verdeutlicht.

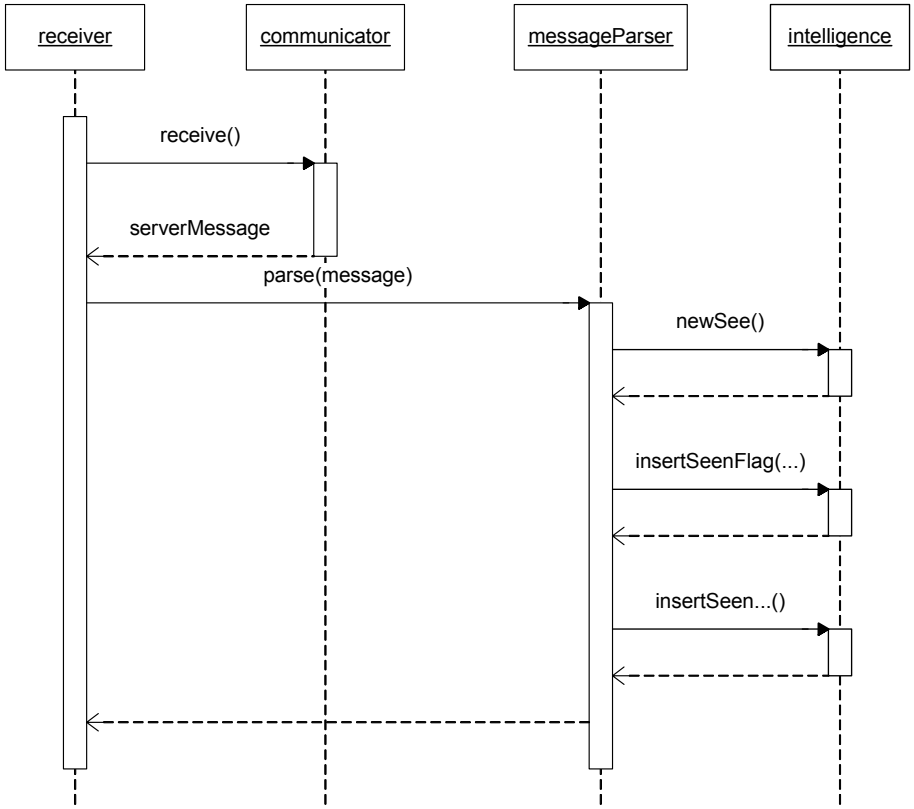

Abbildung 2.6: Empfang einer (see ...)-Nachricht

Abbildung 2.6 zeigt ein Sequenzdiagramm, welches die Vorgänge beim *Empfang* einer See-Nachricht darstellt. Der Receiver ruft zunächst die Methode Communicator.receive() auf, die eine Servernachricht holt (und ggf. auch auf sie wartet). Daraufhin wird diese mit MessageParser.parse(message) vom MessageParser dekodiert. Dieser vergleicht die Nachricht dabei mit unterschiedlichen Mustern und erkennt, dass es sich um eine (see ...) Nachricht handelt. Daraufhin wird die Methode Intelligence.newSee() aufgerufen. Diese bereitet das intelligence-Objekt auf die Aufnahme von Sichtdaten vor, indem sie flüchtige Spielfelddaten, löscht.

Dem so vorbereiteten Intelligence-Objekt werden anschließend aktuelle Daten von gesehenen Spielobjekten übergeben, also Daten über Flaggen, Linien, andere Spieler und Ball.

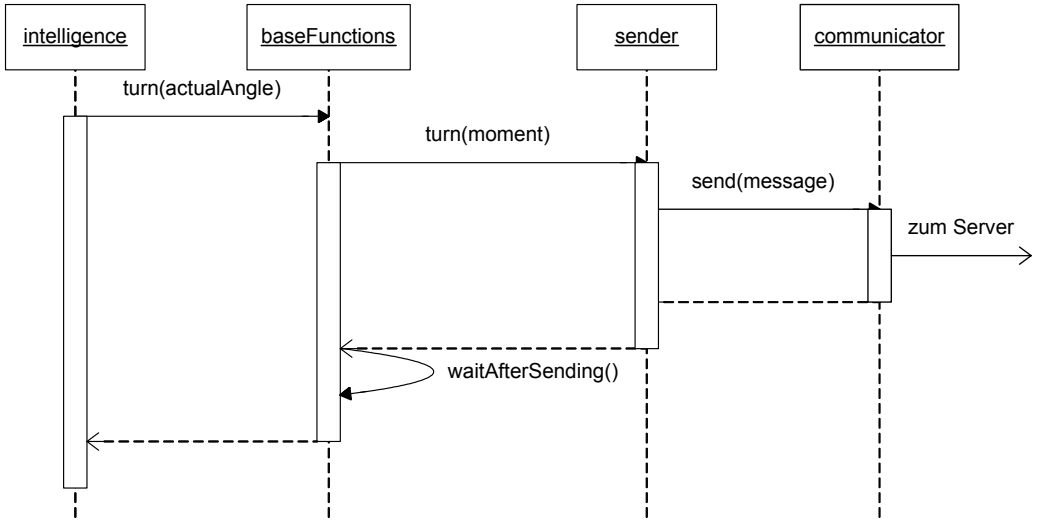

Abbildung 2.7: Ausführung der Aktion turn(actualAngle)

Das intelligence-Object kümmert sich anschließend um die Berechnung auswertbarer Daten.

Abbildung 2.7 zeigt, wie eine Drehung ausgeführt wird. Das intelligence-Objekt benachrichtigt das baseFunctions Objekt, dass es den Spieler um den Winkel actualAngle drehen soll. Dieses berechnet mit Hilfe der aktuellen Spielergeschwindigkeit das dazu benötigte „Drehmoment“. Der Sender setzt mit diesen Informationen dann die Nachricht an den Server zusammen, die der Communicator daraufhin verschickt. Abschließend wird mit waitAfterSending() auf neue Informationen gewartet.

3. Die Kommunikation zwischen den Spielern

Um Aktionen der einzelnen Spieler besser aufeinander abstimmen zu können, ist es nötig, sie miteinander kommunizieren zu lassen. Der Server bietet in der Version 9.4.5 hierzu die Möglichkeit, zehn Zeichen lange Nachrichten zu verschicken. Nummer und Team des Absenders, sowie die Richtung, aus der die Nachricht kam, werden hierbei mit übermittelt.

Unser Team kennt drei verschiedene Nachrichtentypen: die Passnachricht, die einen Spieler darauf aufmerksam macht, dass er einen Ball bekommt, die Decken-Anweisung, mit welcher der Torwart einem Abwehrspieler einen zu deckenden Spieler zuweist und die Ballbesitz-Nachricht, mit welcher der ballführende Spieler seinen Mitspielern den Ballbesitz mitteilt.

3.1. Gerichtete Kommunikation

Da der Server keine Methode zur genauen Adresseierung bereithält ist es nötig, innerhalb der Nachrichten einen Empfänger zu benennen. Wir haben daher die ersten beiden Zeichen einer Nachricht für die Nummer des adressierten Spielers reserviert. Möchte man eine Nachricht an alle Spieler verschicken (Broadcast), so muss die Nummer 88 eingesetzt werden. Die Methoden BaseFunctions.sayTo(int receiverUnum, String message) und MessageParser.parseSay(int time, String data) kapseln diese Funktion. Nachrichten, die nicht an die eigene Spielernummer oder 88 gerichtet sind, werden ignoriert.

3.2. Ballbesitz

Der Begriff Ballbesitz ist weder einheitlich definiert, noch gibt es eine einzig „richtige“ Methode, um den ballführenden Spieler bestimmen zu können. Ein erster Gedanke war, dass derjenige Spieler den Ball besitzt, der diesem am nächsten ist. Ein sich schnell bewegender Ball kann jedoch häufig nicht von den Spielern kontrolliert werden, an denen er sich vorbeibewegt, auch wenn sie ihm kurzzeitig am nächsten sind. Außerdem muss ein Spieler den Ball und den entsprechenden Spieler sehen, um sagen zu können, wer im Ballbesitz ist. Da viele taktische Entscheidungen zumindest davon abhängen, welches Team im Ballbesitz ist, können aus unvollständigen Sichtinformationen Fehlentscheidungen resultieren.
Wir realisierten diese Art der Ballbesitzbestimmung daher nicht und entschieden uns stattdessen für eine Kommunikationslösung, bei der ein ballführender Spieler aus dem eigenen Team seine Mitspieler informieren soll. Ein eigener Spieler wird dabei als ballführend definiert, wenn er den Ball innerhalb einer bestimmten Zeitspanne berührt hat. Ein Ballkontakt ohne vorherigen Ballbesitz wird als Ballgewinn bezeichnet. Verstreicht die Zeitspanne ohne einen Kick, wird dies als Ballverlust bezeichnet. Wir realisierten daher eine einfache, aber dennoch recht effektive Variante: Ein ballführender Spieler meldet *fortlaufend*, dass er noch im Ballbesitz ist. Ein Mitspieler ist nur dann der Meinung, dass das eigene Team den Ball besitzt, wenn er gerade einen Spieler gehört hat, der ihm dies mitteilte. Diese Implementierung hat folgende Eigenschaften: Ein Spieler außerhalb der Hörweite des ballführenden Spielers ist der Meinung, dass der Gegner den Ball hat, handelt also eher defensiv. Fehler bei der Nachrichtenübertragung führen ebenfalls ausschließlich zu defensivem Verhalten. Ein Spieler greift keinen eigenen Spieler an,

um ihm den Ball abzunehmen, da er vorher in Hörweite ist. Deutlich wird, dass defensives Verhalten begünstigt wird, wodurch die Implementierung als „sicher“ angesehen werden kann.
Das Verschicken der Ballbesitznachrichten ist mit den Runden des Servers synchronisiert und wird vom Empfangsthread initialisiert, da dieser genau einmal pro Runde eine SenseBody-Nachricht erhält. Sollte einmal der seltene Fall eintreten, dass zwei Spieler gleichzeitig den Ball kicken, gibt derjenige Spieler nach der die kleinere Spielernummer hat.

3.3. Kollisionsvermeidung

Würde ein Ball führender Spieler jede Runde eine Nachricht verschicken, würden diese Nachrichten gelegentlich mit einer Passnachricht aus dem eigenen Sendethread oder einer Torwartnachricht zusammenfallen, wodurch die Mitspieler eine der Nachrichten überhören würden. Es muss daher vermieden werden, dass zwei Spieler gleichzeitig reden. Dies ließ sich relativ einfach realisieren, indem wir die Ballbesitznachrichten nur in geraden Runden verschicken während andere Nachrichten in ungeraden Runden erlaubt sind. Dies ist möglich, da jede SenseBody-Nachricht eine Rundenzahl enthält. Auf diese Weise sind Kollisionen nur noch zwischen Torwart- und Passnachrichten möglich. Dies wird jedoch ignoriert, da beide Nachrichten vergleichsweise selten sind und Kollisionen daher de Ausnahmefall sind.

4. Spieltechnik

Spieltechniken eines Spielers sind Grundfähigkeiten, die ein Fußballspiel erst ermöglichen. Wir haben die Spieltechniken dribbeln, passen, Ball-abnehmen, aufs-Tor-schießen und decken implementiert. Jeder Agent kann auf diese Spieltechniken zugreifen.

4.1. Dribbeln

Die Dribbelfunktion ermöglicht es einem Spieler, den Ball zu einer übergebenen Zielposition zu dribbeln. Die Aufgabe beim Dribbeln ist es, sich möglichst schnell mit dem Ball fortzubewegen ohne die Kontrolle über den Ball zu verlieren. Dazu muss der Spieler fortwährend den Ball beobachten. Solange die Zielposition nicht erreicht ist, verfolgt der Spieler den Ball bis er sich in Schussnähe befindet und schießt ihn anschließend in Richtung Ziel. Dabei muss die Schusskraft so bestimmt werden, dass der Ball in wenigen Zyklen eingeholt werden kann ohne erheblich an Laufgeschwindigkeit zu verlieren.
Da die Spieler der gegnerischen Mannschaft wahrscheinlich versuchen werden, dem dribbelnden Spieler den Ball abzunehmen, beobachtet dieser Spieler nicht nur den Ball sondern auch die Gegenspieler in der Nähe des Balls. Falls zu viele Gegenspieler den Ballbesitz bedrohen, sucht der Spieler einen freistehenden Mitspieler, um ihm den Ball zuzupassen. Findet er keinen passenden Mitspieler, so schießt er in Richtung des gegnerischen Tores. Eine weitere Möglichkeit auf die Bedrohung der Gegenspieler zu reagieren wäre der Versuch, den Gegenspieler zu umspielen. Da der Ball jedoch durch einen erfolgreichen Schuss eine größere Distanz pro Zyklus zurücklegen kann als ein Spieler mit maximaler Laufgeschwindigkeit, ist ein Pass sicherer, als das Umspielen eines Gegenspielers.

Algorithmus zum Dribbeln

```
while (     Du noch nicht am Ziel angekommen bist
&&      Ball unter Deiner Kontrolle ist
   &&  Gegner Dich nicht bedrohen) {

        while (Distanz zum Ball zu groß ist, um ihn zu schießen)  {
        Schaue, wo der Ball gerade ist;
        Renne einen Schritt in die Richtung des Balls;
   }
   schieße mit angemessener Kraft in Richtung Ziel;
```

```
}
if (am Ziel angekommen) {
        Stoppe den Ball;
        return;
}
if (Ball nicht mehr unter Kontrolle) {
   return;
}
if (Bedrohung durch Gegner) {
   if   (Du gerade einen freien Mitspieler siehst) {
        passe;
        return; } else {
        schieße in Richtung Tor;
        return;
   }
}
```

4.2. Passen

Ein Pass kann sowohl aus taktischen Motiven als auch aus Notsituationen, in denen der Ballbesitz bedroht wird, sinnvoll erscheinen. Aus diesem Grund wird das Passen sowohl von den Agentenklassen als auch von anderen Techniken aufgerufen.

Die zentralen Fragen beim Passen sind es zu entscheiden wer wann zu wem und wohin passt? Uns erschien es am sinnvollsten, den ballführenden Spieler diese Entscheidungen treffen zu lassen. Ein großer Vorteil dieser Variante gegenüber der Verlagerung der Entscheidungsgewalt an freistehende Mitspieler besteht zum einen darin, dass der ballführende Spieler am besten entscheiden kann, wann ein Pass gespielt werden sollte. Zum anderen können die Mitspieler ihre Konzentration auf etwas anderes legen.

Zur Auswahl eines geeigneten Passkandidaten kommen ausschließlich Spieler in Frage, die mit Sicherheit zum eigenen Team gehören und deren Trikotnummer sichtbar ist. Letzteres ist wichtig, um den Spieler direkt ansprechen zu können. Für den Erfolg eines Passes sind zwei Kriterien ausschlaggebend: die Distanz, über die der Ball gespielt werden muss und die umliegenden Gegenspieler, die auf dieser Distanz den Ball abfangen können. Um den besten Spieler auszuwählen, werden alle im letzten Zyklus gesehenen Spieler anhand dieser Kriterien bewertet. Der Spieler, der auf kürzeste Distanz angespielt werden kann und in dessen Gefahrenzone kein Gegenspieler steht, wird angepasst. Ein Gegenspieler steht in der Gefahrenzone, wenn er in einem ähnlichen Winkel zum Passkandidaten, jedoch in einer kürzeren Distanz zum passspielenden Spieler steht.

Algorithmus zum Wählen eines geeigneten Passkandidaten **while** (Du Spieler siehst, die Du noch nicht untersucht hast) {
if (Spieler von unserem Team ist
&& Du seine Trikotnummer erkennen kannst) {

Passkandidat = Spieler;
}
if (**!istGedeckt(Passkandidat)**
&& Passkandidat kann auf kürzester Distanz angespielt werden) {
gewählterSpieler = Passkandidat;
}
}
return gewählterSpieler

istGedeckt(Passkandidat)
while (Du Spieler siehst, die noch nicht als ungefährlich identi. wurden
&& Passkandidat noch als nicht gedeckt gilt){
if (Spieler nicht von unserem Team ist
&& er näher an Dir steht, als der Passkandidat
&& er im ähnlichen Winkel wie der Passkandidat zu Dir steht){
return true;
}
return false;
}

Pässe werden immer direkt auf den Mann gespielt. Eine Verbesserungsmöglichkeit besteht darin, den Pass seitlich versetzt zum Passkandidaten zu spielen, sodass dieser im Laufen den Ball erreichen kann, Gegenspieler in seiner Nähe jedoch nicht. Auf diese Weise könnten auch jene Mitspieler angespielt werden, die im Stand nicht freistehen.

Wurde ein geeigneter Passkandidat ausgewählt, wird er sogleich über den bevorstehenden Pass informiert. Zusätzlich zu den vom Server bereitgestellten Daten wird die Position, auf die der Ball gespielt werden soll, übergeben. Obwohl in unserer Implementierung nur direkt auf den Mann gespielt wird, ist diese Information von großem Nutzen, da oft erhebliche Abweichungen zwischen den berechneten, absoluten Positionen der unterschiedlichen Spieler bestehen. Diese Unterschiede sind auf die in den Servermodellen enthaltenen Unschärfen zurückzuführen.

Im gleichen Zyklus, in dem die Nachricht versendet wurde, passt der Spieler. Dabei wird die Schusskraft anhand der zu überwindenden Distanz bestimmt. Der Parameter, mit dem die Schusskraft an die zu spielende Distanz angepasst wird, muss so gewählt werden, dass der Pass möglichst sicher ankommt. Ist die Schusskraft zu niedrig, kann der Ball leicht vom Gegner abgefangen werden. Ist sie

jedoch zu hoch, bleibt dem angespielten Spieler nicht mehr genügend Zeit, den Ball zu erreichen. Eine Verbesserung unserer Mannschaft könnte erzielt werden, wenn nicht nur die Distanz, über die gespielt werden muss, sondern auch die Anzahl der Gegner entlang des Passes in die Berechnung der Schusskraft mit einbezogen würden.

Erhält ein Spieler eine Passnachricht, hat das Annehmen des Passes höchste Priorität. Die Zeit, die ein Spieler benötigt, um sich in eine Position zu bringen, aus der er den Pass annehmen kann, ist sehr knapp. Um keine Zyklen zur Orientierung zu verbrauchen, reagiert der Agent blind auf die Informationen der Nachricht. Er dreht seinen Körper in die Richtung der Position, die ihm übergeben wurde, und dreht gleichzeitig seinen Kopf in die Richtung, aus der er die Nachricht empfangen hat. Aus dieser Position sollte er in der Lage sein, sowohl den Ball zu beobachten, als auch seine Position verbessern zu können. Sieht er den Ball nicht, bricht der Spieler das Passannehmen ab. Da Unschärfen einen großen Einfluss in der Soccersimulation haben und daher Bälle nie exakt zu einer beabsichtigten Position geschossen werden können, verlässt sich der passannehmende Spieler nicht auf die Position, die ihm übergeben wurde. Solange die Distanz zum Stoppen des Balls zu groß ist, korrigiert er seine Position. Die Position, an der der Ball abgefangen werden soll, wird durch das Lot seiner derzeitigen Position auf die Schusslinie des Balls bestimmt.

Um reaktionsfähig zu bleiben, muss der Spieler frühzeitig erkennen, dass ein Pass nicht mehr ankommen wird. Indikatoren für einen missglückten Pass sind die Zeit, die der Spieler bereits schon auf den Pass wartet, und die Differenz der Distanzen der zuletzt gesehenen Bälle. Ist das Timeout überschritten oder verringert sich die Distanz zum Ball nicht mehr, bricht der Spieler das Passannehmen ab.

Algorithmus zum Pass annehmen

```
Drehe Deinen Körper in die Richtung, in die der Pass gehen soll
Drehe Deinen Kopf in die Richtung, aus der die Nachricht kam
while  ( noch genug Zeit zum Ball stoppen ist ) {
   if    (     Ball aus den Augen verloren
         ||    Du denkst, dass der Pass nicht mehr ankommt) {
         return;  }
   renne einen Schritt zu dem bestimmten Ziel;
   drehe den Kopf in Richtung des Balls;
}
stoppe den Ball;
```

4.3. Ball abfangen

Um dem Gegner den Ballbesitz abnehmen zu können, müssen die Spieler die Technik des Ballabfangens beherrschen. Das Abfangen des Balls kann in zwei Unteraufgaben zerlegt werden: das Einholen und das Abnehmen des Balls.
Mit der Geschwindigkeit und der Reaktionsfähigkeit eines Spielers steht und fällt der Erfolg, den Ball einzuholen. Nach einigen missglückten Versuchen, den Weg zum Ball durch Berechnung zukünftiger Ballpositionen zu verkürzen[2], erschien es uns am sinnvollsten, gerade auf den Ball zu rennen.
Sobald die Distanz zum Ball klein genug ist, um ihn schießen zu können, analysiert der Spieler seine Umgebung. Befindet sich kein Gegner in der Nähe des Balls, besteht keine Notwendigkeit zu weiteren Aktionen. Bedrohen jedoch Gegner den Ballbesitz, ist ein Pass zu einem freien Mitspieler am Erfolg versprechendsten. Sind keine freien Spieler in Sicht, spielt er den Ball in den Rücken des Spielers, der sich am nächsten zum Ball befindet und beginnt mit dem Algorithmus wieder von vorne.

Algorithmus zum Ball abfangen

```
while  ( Du den Ball noch nicht geschossen hast ) {
   while    ( Distanz zum Ball zu groß, um ihn zu schießen ) {
         schaue, wo der Ball sich gerade befindet;
         renne einen Schritt in die Richtung des Balls;
   }
   if    ( keine Gegner Dich bedrohen ) {
         Du hast den Ball. Schreie es ruhig herum;
         return;
   } else   if    (Du gerade einen freien Mitspieler siehst) {
         passe;
         return; } else   {
         schieße den Ball in den Rücken des nächsten Spielers
   }
}
```

[2] Durch Unschärfen und schnelle Ballrichtungsänderungen lagen die gestreckten Bewegungsvektoren des Balls so weit auseinander, dass die benötigte Zeit, der daraus resultierenden Drehungen, den Zeitgewinn durch die Abkürzung weit überstieg. Das Glätten der Vektoren durch das Einbeziehen von historischen Daten beeinträchtigte erheblich die Reaktionsfähigkeit.

4.4. Aufs Tor schießen

Eine Fußballmannschaft kann nur gewinnen, wenn sie auch Tore erzielen kann. Der Algorithmus, den wir in unserem Code realisiert haben, sucht für einen Torschuss den größten freien Winkel, auf den geschossen werden kann. Torpfosten und Gegenspieler begrenzen freie Winkel. Wurde der größte freie Winkel ermittelt, schießt der Spieler auf die Winkelhalbierende dieses Winkels.
Eine Verbesserung dieses Algorithmus könnte erzielt werden, indem der Spieler die Aktionsfähigkeit der einzelnen Objekte mit einbezieht. In unserer Implementierung werden die Torpfosten auf die gleiche Weise in die Bestimmung des optimalen Schusswinkels mit einbezogen wie die gegnerische Feldspieler und der gegnerische Torwart. Jedoch ist der Aktionsradius eines Torwarts um ein dreifaches höher als der eines Feldspielers. Von einem Torpfosten hingegen gehen keine Aktionen aus. Er stellt lediglich die Begrenzung des Tors dar.

4.5. Decken

Da Pässe des Gegners im Strafraum den Torwart leicht ausspielen können, werden die Spieler, die sich am nächsten zum Tor befinden, gedeckt. Welche Gegenspieler von wem gedeckt werden, organisiert der Torwart.
Bekommt ein Spieler die Anweisung, einen Gegner zu decken, stellt er sich auf die Gerade zwischen Gegner und Ball. Diese Position birgt das Problem, sowohl den Ball als auch den Gegner im Auge behalten zu müssen. Um diesen Konflikt zu lösen, rennt der Spieler erst so nah an den Gegner heran, bis er ihn fühlen kann und dreht sich anschließend in Richtung Ball. Auf diese Weise kann er die Bewegungen des Gegners und des Balls gleichzeitig verfolgen und seine Position entsprechend korrigieren.

```
while  (     Du diesen Spieler decken sollst )  {
   while     (     Distanz zum Spieler zu groß ist )  {
         Schaue, wo der Spieler gerade ist;
         Renne einen Schritt in die Richtung des Spielers;
   }
   Bestimme eine Position zwischen Spieler und Ball
   Renne zu dieser Position
   Drehe Dich anschließend zum Ball
   while     (     Du keine andere Anweisung bekommen hast
               &&  Spieler hinter Dir ist
               &&  Ball vor Dir ist )  {
         doNothing;
```

```
    }
}
```

5. Spieltaktik

Die Spieltaktik wird im Fußball vom Trainer bestimmt und erklärt, wie sich ein Spieler im Spiel verhalten soll. Dies zeigt sich in folgender Definition:
„Die Taktik ist ein geplantes und im Vorhinein abgesprochenes Spielverhalten unter Berücksichtigung […] der eigenen Möglichkeiten. Oftmals bringt die spielerische und kämpferische Leistung einer Mannschaft ohne die richtige Taktik nicht den gewünschten Erfolg.“[3]

5.1. Aufstellung, Zuständigkeitsbereiche

Die Vorgaben, die in der Programmierung zum Tragen kommen, beziehen sich auf die Aufstellung der Mannschaft, auf die Zuständigkeitsbereiche der Spieler und auf das Verhalten der Spieler im Spiel. Die Aufstellung die hier gewählt wurde, entspricht einem klassischen 3-4-3-er Spielsystem und gibt somit die Startposition der Spieler vor.
Die Zuständigkeitsbereiche der Spieler differieren in Abhängigkeit von der Spielsituation, in welcher der Spieler sich befindet.

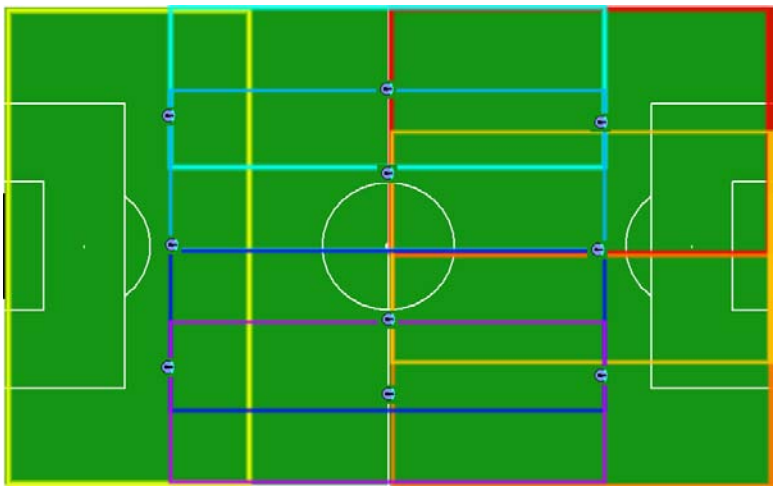

Abbildung 5.1: Zuständigkeitsbereiche im normalen Spielverlauf

Befindet sich der Spieler im normalen Spielverlauf, haben die Agenten überlappende Zuständigkeitsbereiche, d.h. ein Agent kann in den Bereich eines anderen Agenten hineinlaufen. Die Ausgangsposition befindet sich in diesem Verlauf in der Mitte des

[3] http://www.fd21.de/15859.asp

Zuständigkeitsbereiches. Der Grund für die Unterteilung in Zuständigkeitsbereiche ist anhand zweier Extrema gut zu erklären. Ist zum einen der Spieler für das gesamte Spielfeld zuständig, so führt das dazu, dass alle Spieler zum Ball rennen und somit große Teile des Spielfeldes frei bleiben. Dadurch öffnet sich das Feld für gegnerische Angriffe. Sind im anderen Extrem die Zuständigkeitsbereiche der Spieler strikt voneinander getrennt, so kommt es zum so genannten Standfußball. In diesem Fall kann es dazu führen, dass z.B. Spieler in ihren Bereichen dem Gegner zahlenmäßig unterlegen sind oder zu selten in den Spielverlauf eingreifen. Eine erfolgreiche Spieltaktik wird daher durch überlappende Zuständigkeiten erreicht, wobei die Qualität der Teamarbeit von dem Grad der Überlappung abhängt.

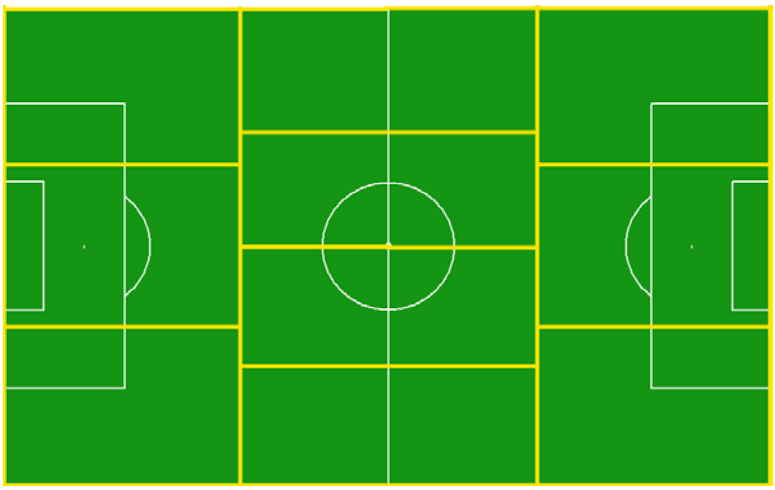

Abbildung 5.2: Zuständigkeitsbereiche in Sonderspielsituationen

In Sonderspielsituationen, zu denen Eckball-, Freistoß- und Einwurfsituationen zählen, ist es hingegen sinnvoll, die Zuständigkeiten eindeutig aufzuteilen. Die Aufteilung stellt sicher, dass nur ein Spieler einen Eckball, Freistoß oder Einwurf ausführt. Dies führt zwar nicht zu einer optimalen Ausnutzung der Spielsituation, jedoch wäre die Zuordnung der Spieler zu Sonderspielsituationen ansonsten nur über say-Nachrichten möglich und dies gestaltet sich schwierig und vor allem zeitaufwendig.

5.2. Taktik-Kriterien

Das Spielerverhalten wird durch den Spielertyp beeinflusst. Die hier implementierten Spielertypen sind Torwart, Verteidigung, Mittelfeld und Sturm.
Um das Spielverhalten eines Spielers genau beschreiben zu können, sind Kriterien notwendig, die entscheiden, welche Aktion ausgeführt wird. Zu den im Programm

verwendeten Kriterien gehören die Position des Spielers, die Position des Balls, die Zuständigkeitsbereiche und der Ballbesitz.
Die eigenen Möglichkeiten bzw. die Aktionen, die dann ausgeführt werden können, wurden in dem Kapitel Spieltechnik bereits erklärt und sollen hier nur noch einmal kurz erwähnt werden. Die Fähigkeiten, die die Agenten beherrschen, sind dribbeln, passen, Pässe annehmen, Ball abfangen, aufs Tor schießen und decken - neben den Grundfunktionen, wie z.B. zu einer Position laufen.

5.3. Das Agents-Package

Die oben genannten Faktoren, Spielverhalten und eigene Möglichkeiten, wurden in dem Agents-Package realisiert.

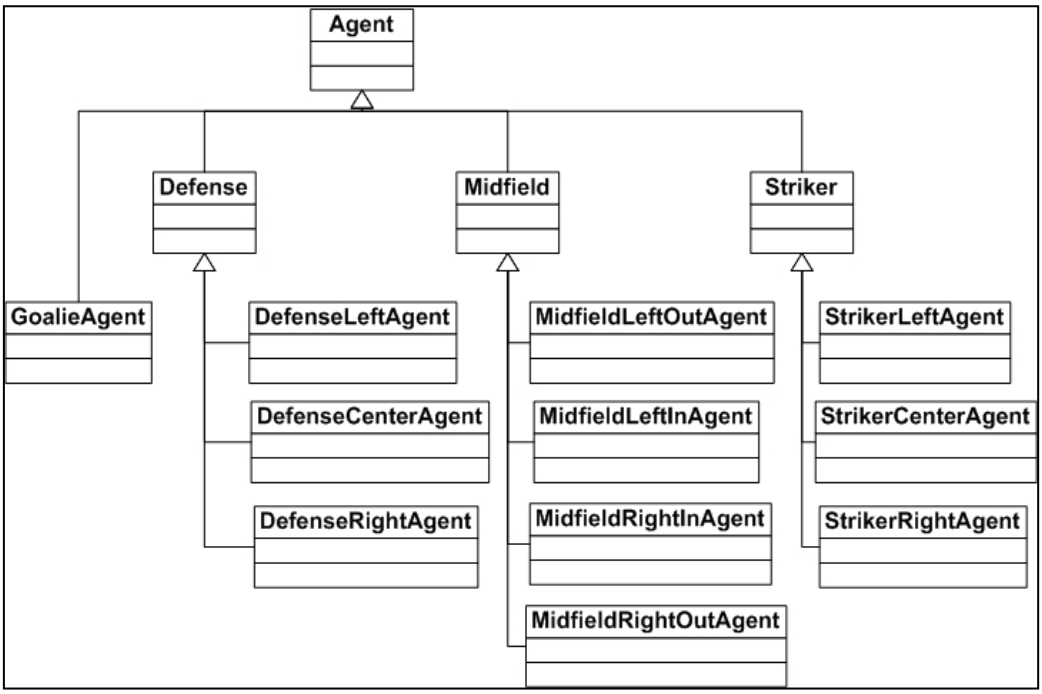

Abbildung 5.3: Klassendiagramm des Agents-Package

Das Agents-Package ist dreistufig aufgebaut. Auf der obersten Ebene, in der Agent-Klasse, werden die Schnittstellen zu den übrigen Klassen aufgebaut und sind die Algorithmen implementiert, die das Spielverhalten der Spieler in Sonderspielsituationen beschreiben.
Auf der mittleren Ebene befinden sich die Klassen Defense, Midfield und Striker, die von der Agent-Klasse erben. Diese Klassen stellen die Methoden bereit, die für die

Spielertypen Verteidigung, Mittelfeld und Stürmer von Bedeutung sind. Dies ist insbesondere das Spielverhalten eines Spielertyps im normalen Spielverlauf.
Von diesen Klassen erben dann wiederum die Klassen DefenseLeftAgent, DefenseCenterAgent, etc. In diesen Klassen sind dann spielerindividuelle Einstellungen enthalten, zu denen die Spielposition, Startposition und die Zuständigkeitsbereiche zählen.
Eine Sondersituation nimmt dabei die Klasse Goalie an. Diese Klasse stellt die Funktionen für den Torwart bereit. Da jede Mannschaft nur ein Torwart hat, ist eine Unterteilung der Klasse in Spielertyp und spielerindividuelle Einstellungen nicht sinnvoll.

5.4. Agent-Lebenszyklus

Ein Agent muss die Möglichkeit haben während des gesamten Spielverlaufes durchgehend zu entscheiden, welche Handlung oder Aktion aufgrund der ihm vorliegenden Umwelt durchzuführen ist. Die Umweltveränderungen können in veränderte Spielmodi oder einer neuen Spielsituation auf dem Spielfeld bestehen.
Nach dem Start eines Agenten wird die Methode decideAction ausgeführt. Diese Methode enthält eine While-Schleife, in der fortlaufend anhand des Spielmodus entschieden wird, welche Handlung auszuführen ist. Wird durch den Spielmodus ein Ende oder Abbruch des Spiels signalisiert, wird die Schleife abgebrochen und das Programm beendet.
Ist der Spielmodus auf Eckball, Freistoß oder Einwurf eingestellt, werden entsprechende Methoden zur Behandlung dieser Sonderspielsituation ausgeführt, die bei allen Agenten gleich sind.
Zeigt der Spielmodus einen normalen Spielverlauf an, werden auch hier die entsprechenden Methoden zur Behandlung aufgerufen, die abhängig von dem Spieltyp eines Agenten sind.
In den Methoden zur Behandlung von Eckball, Freistoß, Einwurf oder normalem Spielverlauf, wird entschieden welche Aktion aufgrund der Spielsituation auf dem Spielfeld auszuführen ist. Die Aktionen die dabei aufgeführt werden sind zum einen die Grundfunktionen, wie zum Ball laufen, etc. oder Spieltechniken. Da diese in der Regel länger als einen Zyklus dauern, muss auch hier jede Runde eine Überprüfung des Spielmodus stattfinden und die Aktion bei Veränderung abgebrochen werden.

Danach findet eine neue Validierung in der Methode decideAction statt und die entsprechende Methode zum Spielmodus wird ausgeführt.

5.5. Sonderspielsituation – Algorithmus

Befindet sich der Agent in einer Sonderspielsituation, muss ein Agent einen Abstoß ausführen. Das bedeutet, dass nach der ersten Berührung des Balls ein anderer Spieler den Ball berühren muss, um kein Faul zu provozieren. Dabei führt derjenige Agent den Eckball, Freistoß oder Einwurf aus, in dessen Zuständigkeitsbereich der Ball liegt. In diesen Algorithmen läuft der Agent zum Ball und passt diesen zu einem frei stehenden Mitspieler. Steht kein Spieler frei, wird der Ball zum Tor geschossen.
Die Idee der Algorithmen, die die Spielertypen im normalen Spielverlauf verwenden, werden im Folgenden erklärt und findet eine genauere Beschreibung in Pseudocode.

5.6. Torwart – Algorithmus

Im normalen Spielverlauf organisiert der Torwart die Abwehr, solange der Ball außerhalb eines Gefahrenbereiches ist. Der Gefahrenbereich umfasst einen Umkreis von 20m um den Torwart. Die Organisation verläuft dann über Say-Nachrichten, die fortlaufend an die Verteidigung gesendet werden. Dabei erhalten zwei Verteidiger die Anweisung, jeweils einen der beiden Gegenspieler zu decken, die am nächsten am Tor stehen. Dadurch, dass lediglich zwei Verteidiger decken, wird sichergestellt, dass kein Gegenspieler ungestört nach vorne dribbelt und aufs Tor schießt.
Befindet sich der Ball innerhalb des Gefahrenbereiches, wird die Organisation der Verteidigung unterbrochen, und der Torwart bewegt sich auf seine Position. Zur Berechnung der Position dient eine Linie, die sich zwei Meter parallel vor der Torlinie befindet. Der Schnittpunkt aus dieser Linie und der Linie, die vom Tormittelpunkt zum Ball führt, bestimmt die Position, auf die sich der Torwart dann stellt. Kommt nun der Ball in seinen Fangbereich, versucht der Torwart diesen zu fangen.
Erfolgt ein Abstoß, bewegt sich der Torwart innerhalb seines Strafraums zu einem nahe liegenden Spieler und passt den Ball ab. Dies geschieht um weite Pässe zu vermeiden und damit dem Gegner die Möglichkeit, den Ball abzufangen, zu verringern.

Spielmodus: normaler Spielverlauf

if (Der Ball ist >20m von mir entfernt){

Koordination der Verteidigung
}
else{
Lauf auf die optimale Position
}

Spielmodus: Goal-Kick

If (ein Spieler frei steht && du hast den Ball)
„beame“ zum nächsten Spieler
passe
else{
schieße aufs gegnerische Tor
}

5.7. Verteidigung - Algorithmus

Die Verteidigung soll dem Gegner den Ball abnehmen und Torschüsse des Gegners verhindern. Da Pässe des Gegners im Strafraum den Torwart leicht ausspielen können, sollen diese unterbunden werden. Dazu beherrscht die Verteidigung die Spieltechnik des Deckens. Wie bereits erwähnt, wird das Decken vom Torwart organisiert. Erhält ein Verteidiger eine Anweisung zum Decken, muss sichergestellt werden, dass diese durchgängig ausgeführt wird. Um dies zu gewährleisten, erstrecken sich die Aktionsbereiche für alle Verteidiger im Gegensatz zu anderen Spielern über die komplette Breite des Spielfeldes, damit das Decken eines Gegenspielers nicht an einer vorgegebenen Grenze aufhört und ab da ein eventuell gefährlicher Spieler nicht mehr gedeckt wird.
Befindet sich die Verteidigung im Ballbesitz, ist es ihre Aufgabe, den Ball schnell an das Mittelfeld abzugeben und somit aus dem Gefahrenbereich vor dem Tor herauszubefördern.

Spielmodus: normaler Spielverlauf

If (Du eine Anweisung zur Ballannahme hast)
führe Ballannahme durch
If (der Ball in deinem Zuständigkeitsbereich ist){
if (Du den Ball hast)
drehe dich nach vorne
passe, wenn Spieler frei steht, ansonsten dribble
if (unser Mitspieler den Ball hat)
ruhe dich aus
if (der Gegner den Ball hat) {
if (Du keine Anweisung zum decken hast) fange den Ball ab

```
            if (Du eine Anweisung zum decken hast &&
                der Ball weit weg ist) decke
            if (Du eine Anweisung zum decken hast &&
                der Ball nah dran ist ) fange den Ball ab
        }
    }
    If (der Ball nicht in deinem Zuständigkeitsbereich ist){
        if (unser Mitspieler den Ball hat)  gehe auf deine Spielposition
        if (der Gegner den Ball hat &&
                Du eine Anweisung zum decken hast) decke
        if (der Gegner den Ball hat && keiner gedeckt werden soll)
                gehe auf deine Spielposition
    }
```

5.8. Mittelfeld – Algorithmus

Dem Mittelfeld kommen zwei Hauptaufgaben zu. Zum einen dem Gegner frühzeitig den Ball abzunehmen und unter die eigene Kontrolle zu bringen und zum anderen den Ball zum Sturm zu befördern.

Hat ein Mittelfeldspieler den Ball, so wählt dieser einen zufälligen Punkt am Rand seines Zuständigkeitsbereiches, der dem gegnerischen Tor am nächsten ist, und dribbelt solange in diese Richtung, bis die Situation einen Pass erfordert. Ist der Mittelfeldspieler dort angelangt, erfolgt ein Pass zu einem Mitspieler, der vor ihm steht und durch die Zuständigkeitsbereiche in vielen Fällen ein Stürmer ist.

Da das Mittelfeld durch diese beiden Aufgaben ein sehr hohes Laufpensum absolviert, fiel die Entscheidung auf eine 4-er Kette im Mittelfeld.

Spielmodus: normaler Spielverlauf

```
If (Du eine Anweisung zur Ballannahme hast)
    führe die Ballannahme durch
If (der Ball in deinem Zuständigkeitsbereich ist){
    if (Du hast den Ball) dribble nach vorne und passe,
        wenn frei ist, ansonsten schieß aufs Tor
    if (unser Mitspieler den Ball hat) gehe auf deine Spielposition
    if (Gegner den Ball hat) fange den Ball ab
}
If (der Ball nicht in deinem Zuständigkeitsbereich ist){
    if (Du dich im Zuständigkeitsbereich befindest) laufe Richtung Ball
    if (Du dich nicht im Zuständigkeitsbereich befindest)
            gehe auf deine Spielposition
}
```

5.9. Sturm – Algorithmus

Der Sturm hat die Aufgabe, den Ball vom Mittelfeld abzunehmen und ihn dann ins Tor zu befördern. Bei Ballbesitz des Stürmers ist die Distanz zum Tor und die Lage vor dem Tor, also ob das Tor frei steht oder nicht, von zusätzlicher Entscheidungsrelevanz. Ist der Stürmer zu weit von dem Tor entfernt, vervollständigt er die Aufgabe des Mittelfeldes und dribbelt den Ball selbst in eine geeignete Distanz. Ist das Tor so zugestellt, dass ein Schuss nicht zum gewünschten Ziel führen würde, wird ein Pass ausgeführt. Dies geschieht, da der Winkel eines anderen Mitspielers zum Tor eventuell eine bessere Situation für den Schuss ergibt und dieser damit aussichtsreicher für ein Tor ist.

Spielmodus: normaler Spielverlauf

If (Du eine Anweisung zur Ballannahme hast)
  führe die Ballannahme durch
If (der Ball in deinem Zuständigkeitsbereich ist){
  if (Du hast den Ball) {
    if (Das Tor nahe ist && das Tor frei steht) schieße aufs Tor
    if (Das Tor nahe ist && das Tor nicht frei steht) passe
    if (Das Tor weit weg ist) dribbel nach vorne
}
if (unser Mitspieler den Ball hat) gehe auf deine Spielposition
if (Gegner den Ball hat) fange den Ball ab und schieße aufs Tor

6. Fazit

Der Soccerserver ist eine Simulationsumgebung, die ein reales Fußballspiel gut abbildet. Da es sich jedoch um eine Simulation handelt und somit ein Modell ist, ist die entscheidende Frage, ob es möglich ist mit Ansätzen aus der realen Welt gute Ergebnisse zu erzielen. Das Ergebnis unserer Arbeit hat gezeigt, das dies in weiten Teilen gut möglich ist.

Da jedoch deutliche Unterschiede zwischen Realität und Soccerserver existieren, allen voran die fehlende dritte Dimension, ist es wahrscheinlich, dass mit anderen Ansätzen bessere Ergebnisse erzielbar sind. Besonders häufig findet man Mannschaften, die mit Hilfe von neuronalen Netzen lernfähig sind. Beispielsweise kann man ein neuronales Netz mit verschiedenen Spielsituationen darauf trainieren, den „besten“ Winkel für einen Torschuss zu berechnen.

Der Soccerserver bietet eine gute Umgebung zum Testen solcher Ansätze aus dem Gebiet der künstlichen Intelligenz. Aufgrund der Client Server Architektur ist man bei der Entwicklung programmiersprachenunabhängig, wodurch die Zahl möglicher Ansätze nahezu unbegrenzt ist.