HAW HAMBURG

BACHELORTHESIS

Lucas Neitsch

Entwicklung eines Multiagentensystems für die Simulation des Positionsangriffs im Handball unter Verwendung des MARS-Frameworks

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Lucas Neitsch

Entwicklung eines Multiagentensystems für die Simulation des Positionsangriffs im Handball unter Verwendung des MARS-Frameworks

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Prof. Dr. Christian Lins

Eingereicht am: 30. August 2024

Lucas Neitsch

Thema der Arbeit

Entwicklung eines Multiagentensystems für die Simulation des Positionsangriffs im Handball unter Verwendung des MARS-Frameworks

Stichworte

Mulitagentensystem (MAS), Agentenbasierte Modellierung (ABM), MARS-Framework, Handball Simulation, Team Sport Simulation, Goal-Oriented Action Planning (GOAP)

Kurzzusammenfassung

Diese Bachelorarbeit beschreibt die Entwicklung eines Multiagentensystems zur Simulation des Positionsangriffs im Handball unter Verwendung des MARS (Multi-Agent Research and Simulation)-Frameworks. Das System soll es Trainern ermöglicht, vorgegebene Auftakthandlungen in bestimmten Spielsituationen zu testen und als Werkzeug zur Verbesserung der taktischen Planung und Entscheidungsfindung dienen. Der Fokus liegt auf der Simulation von Positionsangriffen, bei denen jeder Agent sich an das Verhalten der Mit- und Gegenspieler anpassen muss. Das System modelliert die Handballumgebung, einschließlich Spielerbewegungen, Balldynamik und Kommunikation zwischen den Agenten. Durch die Implementierung von Goal-Oriented Action Planning (GOAP) können die Agenten kontextabhängige Entscheidungen während der Simulation treffen. Die Arbeit diskutiert die Systemarchitektur, das Design, die Tests und die Resultate der ausgeführten Experimente. Die Experimente werden in Hinsicht auf die Nutzbarkeit des Systems durch Trainer und Trainerinnen, sowie der Eignung als Darstellung des Handballsports ausgewertet.

Nach Auswertung der Ergebnisse wird festgestellt, dass die Eignung als Werkzeug für Handballtrainer noch nicht gegeben ist. Der Grund hierfür ist in den noch nicht ausgereiften Verhaltensweisen der Spieler-Agenten zu sehen. Durch diese ist eine Simulation des Handballsports und insbesondere der eingegebenen Auftakthandlungen nicht gegeben. Die Gründe hierfür liegen in dem entwickelten GOAP-Modell, dessen Pläne nicht detailliert genug sind und bei denen der Fokus nicht die gewünschten Verhaltensweisen priorisiert.

Lucas Neitsch

Title of Thesis

Development of a Multi-Agent System for simulating the positional attack in Handball using the MARS framework.

Keywords

Multi-Agent System (MAS), Agent-based Modelling (ABM), MARS framework, Handball Simulation, Team Sport Simulation, Goal-Oriented Action Planning (GOAP).

Abstract

This bachelor thesis describes the development of a multi-agent system for the simulation of positional attacks in handball using the MARS (Multi-Agent Rese-arch and Simulation) framework. The system enables coaches to test their attacking plays in specific game situations and serves as a tool to improve tactical planning and decision-making. The focus is on the simulation of positional attacks, where each agent must adapt to the behaviour of teammates and opponents. The system models the handball environment, including player movements, ball dynamics and communication between the agents. By implementing Goal-Oriented Action Planning (GOAP), the agents can make context-dependent decisions during the simulation. The paper discusses the system architecture, the design and testing and the results of the experiments performed. The experiments are evaluated regarding the usability of the system by coaches and its suitability as a representation of handball.

After analysing the results, it is concluded that the suitability as a tool for handball coaches is not yet given. The reason for this is to be seen in the not yet fully developed behaviour of the player agents. This means that it is not possible to properly simulate handball, and particularly the plays entered. The reasons for this lie in the developed GOAP model, whose plans are not detailed enough and where the focus does not prioritise the desired behaviours.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ABM	Agentenbasiertes Modell
FPS	Frames per Second
GNU	General Public License
GOAP	Goal-Oriented Action Planning
MARS	Multi Agent Research and Simulation
MAS	Multiagenten Simulation
STRIPS	Stanford Research Institute Problem Solver
STS2	Simple Team Sport Simulator 2
XGoals	Expected Goals

Glossar

Auftakthandlung Ein Ablauf von Lauf- und Passwegen und weiteren Anweisungen, durch die eine angreifende Mannschaft versucht Tormöglichkeiten zu erspielen. Das Befolgen der Anweisungen kann dabei jederzeit zugunsten einer besseren Alternative abgebrochen werden.

Expected Goals (XGoals) Ein Wert, der eine Wahrscheinlichkeit angibt, dass ein bestimmter Torwurf zu einem Tor führt, basierend auf verschiedenen Faktoren.

Goal-Oriented Action Planning (GOAP) Ein KI-Planungssystem, bei dem Agenten auf Grundlage von Zielen Entscheidungen treffen, um Aktionen auszuführen, welche das Erreichen des Ziels unterstützen.

Handball Eine Mannschaftssportart, in der zwei Teams aus je 7 Spielen gegeneinander antreten und versuchen, den Ball in das Tor des Gegners zu werfen.

MARS-Framework Ein C# Framework für Multiagentensysteme, das zur Entwicklung und Simulation von Szenarien verwendet wird.

Multiagentensystem (MAS) Ein System, das aus mehreren Agenten besteht, die autonom handeln und miteinander interagieren, um Aufgaben zu lösen oder bestimmte Ziele zu erreichen.

Positionsangriff Eine bestimmte Situation eines Handballspiels. In dieser wird ein organisierter Angriff gespielt, in dem die Spieler ihre Positionen basierend auf ihrer Rolle einnehmen und durch Auftakthandlungen versuchen, zum Torerfolg zu kommen.

1 Einleitung

Diese Arbeit beschreibt ein im Rahmen der Bachelorthesis entworfenes Multiagentensystem. Dieses System simuliert den Handballsport, indem Agenten, die die Spieler repräsentieren, in isolierten Spielsituationen gegeneinander antreten. Ein wichtiger Aspekt des entworfenen Systems ist die Möglichkeit für Trainer und Trainerinnen, die das System verwenden wollen, Auftakthandlungen für die angreifende Mannschaft zu konfigurieren, die dann von den Agenten ausgeführt werden. Nachfolgend beschreibt dieses Kapitel die Motivation für die Arbeit, definiert die Zielsetzung sowie die gesetzte Eingrenzung. Abschließend wird der Aufbau der restlichen Arbeit ausgeführt.

1.1 Hintergrund und Motivation der Arbeit

Im Handballsport ist der Positionsangriff ein essenzieller Faktor für eine Mannschaft. Schafft sie es, hier die richtigen Mittel zu finden, um die Stärken ihrer Spieler in Szene zu setzen oder die Schwächen der gegnerischen Abwehr auszunutzen, bestehen gute Möglichkeiten, erfolgreich zu sein. Trainer erarbeiten aus diesem Grund Auftakthandlungen für die Angriffe ihrer Mannschaft, um eben diese Faktoren für den Erfolg zu schaffen. Insbesondere im Bereich des professionellen Handballs planen Trainer für jeden Gegner spezifische Auftakthandlungen und geben ihrer Mannschaft oftmals in konkreten Spielsituationen einen bestimmten Ablauf vor.

Üblicherweise verwenden Trainer zum Vermitteln ihrer Anweisungen eine sogenannte Taktiktafel oder Applikationen, die das simple Animieren der Abläufe ermöglichen. Die Applikationen bieten dabei jedoch nur die Möglichkeit, eine strikte Abfolge von Bewegungen der Spieler abzubilden. Im Handball beinhalten die Auftakthandlungen allerdings regelmäßig die Möglichkeit, an bestimmten Punkten zwischen verschiedenen Varianten auszuwählen, je nachdem, was in der Situation am vielversprechendsten ist. Ebenso können Spieler ungeplant die

vorgegebenen Abläufe abbrechen, wenn sich in der Situation unvorhergesehen eine andere Möglichkeit für einen guten Torwurf bietet.

Darüber hinaus ist es den Trainern mit diesen Methoden nicht möglich, ihre Ideen für den Positionsangriff zu testen, beispielsweise gegen bestimmte Abwehrformationen und Gegenspieler. Für solche Analysen sind Tests der Abläufe im Training oder erst in einem tatsächlichen Spiel notwendig. Für einen solchen Fall können Multiagentensysteme eine wichtige Rolle einnehmen, um die Abläufe für konkrete Spielsituationen direkt bei der Konzeptionierung zu simulieren und Analysen anstellen zu können. Ein solches System wurde im Rahmen dieser Arbeit entwickelt, um Trainer und Funktionäre des Handballsports in der Spielvorbereitung zu unterstützen. Hierbei ist für die Umsetzung des Systems wichtig, dass Trainer ihre Anweisungen für verschiedene Auftakthandlungen an die Agenten vermitteln können. Die Agenten müssen innerhalb des Systems zusammenarbeiten und individuell die Spielsituationen bewerten, um so zu handeln wie es die Spieler in einem Handballspiel tun würden.

1.2 Zielsetzung und Abgrenzung der Arbeit

Wie zuvor in Kapitel 1.1 beschrieben, soll das in dieser Arbeit entworfene System ein Analyseprogramm für Trainer bieten, um von ihnen überlegte Abläufe im Positionsangriff gegen bestimmte Konstellationen von Gegenspielern zu testen. Zudem bietet es die Funktionalität einer Art digitaler Taktiktafel, da durch die Visualisierungskomponente des Systems die Abläufe an reale Spieler übermittelt werden können.

Spezifisch wird im Rahmen dieser Arbeit ein Multiagentensystem für den Sport entwickelt und mithilfe des MARS-Frameworks implementiert. In diesem System liegt der primäre Anteil darin, die Umgebung für die Simulation umzusetzen. Der zweite wichtige Anteil liegt in der Umsetzung der Agenten der angreifenden Mannschaft und insbesondere der Möglichkeit, konkrete Abläufe in das System einzugeben, welche von den Agenten ausgeführt werden. Da es im Handballsport bei solchen Abläufen für den Ballführer häufig mehrere Möglichkeiten zum Agieren gibt, müssen auch die Agenten basierend auf ihrer Wahrnehmung der Spielsituation aus mehreren Optionen eine Entscheidung fällen.

Die Simulation von einem vollständigen Handballspiel wird mit dem System nicht möglich sein, da dies den Rahmen einer Bachelorarbeit überschreiten würde. Aus demselben Grund wird das Agentenverhalten des verteidigenden Teams sowie der Torhüter nicht im Detail behandelt, allerdings ist zumindest eine simple Implementierung für eine sinnvolle Analyse des Systems und die Verwendung durch Trainer notwendig. Die detaillierte Überprüfung und Umsetzung des Handballregelwerks ist für den hier behandelten Anwendungsfall nebensächlich und wird daher nicht Teil des Systems sein. Selbst wenn diese Dinge in der aktuellen Version nicht oder nur vereinfacht umgesetzt werden, soll das System in Hinsicht auf solche Ergänzungen möglichst erweiterbar gebaut werden. So soll es möglich sein, dass das System zukünftig einen größeren Umfang an Funktionen für Trainer und Funktionäre bieten kann. Zudem könnten die Erweiterungen dazu führen, auch im Gebiet der Forschung mit Multiagentensystemen ein interessantes Framework zu bieten.

1.3 Aufbau der Arbeit

In dieser Arbeit wird zunächst in Kapitel 2 auf die Grundlagen für das weitere Verständnis und die Methodik eingegangen, wie die Beschreibung von Multiagentensystemen, dem MARS-Framework, wichtigen Aspekten des Handballsports und verwandten Arbeiten. In dem folgenden Kapitel 3 werden die an das System gestellten Anforderungen erörtert. Kapitel 4 beschreibt die Entwurfsdetails des Systems und geht dabei auf die wichtigen Aspekte der Umgebung sowie der Agenten ein. In Kapitel 5 werden die Implementierungsdetails beschrieben. Kapitel 6 stellt durchgeführte Experimente zur Evaluierung des Systems dar. Diese werden in Kapitel 7 dahingehend diskutiert, was sie für das Erreichen der Ziele und die Nutzbarkeit des Systems bedeuten. Das Kapitel 8 führt die Schlussfolgerungen aus und bietet Empfehlungen für weitere Verbesserungen und Erweiterungen des entwickelten Systems, die in weiterer Arbeit umgesetzt werden sollten.

2 Grundlagen

Dieses Kapitel betrachtet die Grundlagen, die für die Entwicklung dieser Arbeit relevant sind. Sie stellen die Erkenntnisse aus einer Literaturrecherche dar, welche durchgeführt wurde, um eine fundierte Basis für die Entwicklung des Systems zu schaffen.

2.1 Multiagentensysteme

Multiagentensysteme (MAS), oft auch bekannt als *Agentenbasierte Modelle (ABM)*, sind softwarebasierte Repräsentationen von bestimmten Systemen und Situationen der realen Welt. In diesen Simulationen agiert eine Menge A von autonomen, adaptiven Entscheidungsträgern (Agenten) in einer Simulationsumgebung, welche das zu modellierenden Systems darstellt. Die Agenten sind Individuen, die bestimmte Ziele verfolgen, welche sie durch die Interaktion mit der Umgebung und untereinander erreichen (Saeed Rahimi et al., 2022). Eine wichtige Grundlage von MAS ist die Autonomie der Agenten, dass sie also eigenständig handeln und Entscheidungen treffen und nicht durch deterministische Muster geleitet werden. Ihre Handlungsentschlüsse fällen sie basierend auf ihrem aktuellen internen Zustand, ihrer Wahrnehmung, ihrer Umgebung und interner Ressourcen, die ihnen zur Verfügung stehen (Clemen et al., 2022). Durch ihr Verhalten wirken die Agenten auf ihr Umfeld ein und verändern dessen Zustand, was zu neuen Verhaltensweisen aller sich in der Menge A befindlichen Agenten A_i führen kann. Auch wenn alle Agenten des Systems eigenständig sind und individuelle Entscheidungen treffen, haben sie oftmals übergeordnete Ziele. Für das Erreichen dieser Ziele, müssen sie mit anderen Agenten zusammenarbeiten. Hierfür müssen die Agenten kommunizieren und sich untereinander koordinieren (Abdollah Amirkhani & Amir Hossein Barshooi, 2021).

Agenten können auf unterschiedliche Weise umgesetzt werden. Gemeinsam ist dabei, dass jeder Agent eine individuelle Programmeinheit ist. Häufig handelt es sich um sogenannte *Simple Reflex Agenten*, welche durch ein regelbasiertes Verhalten definiert sind. Diese sind in der

Regel einfach nachzuvollziehen, da den definierten „wenn-dann“-Fällen leicht zu folgen ist. Jedoch können Agenten auch komplexere Umsetzungen haben, so beispielsweise lernende Agenten, die ihr Verhalten durch Training mit maschinellen Lernalgorithmen erlernen (Clemen et al., 2022).

MAS haben ihren Ursprung in der Biologie, wo sie auch heute für die Modellierung und Untersuchung von Bewegungs- und Migrationsmustern von Tieren verwendet werden (Abdollah Amirkhani & Amir Hossein Barshooi, 2021). Darüber hinaus kommen sie auch in vielen anderen Domänen zum Einsatz, wie beispielsweise dem Modellieren von Bewegungsmustern und Modalitätsverhalten der Einwohner von Städten. Hierbei kommen die Systeme als digitale Zwillinge oder digitale Schatten zum Einsatz, um die Verhaltensweisen in der Region nachzuempfinden und analysieren zu können (Saeed Rahimi et al., 2022; Tolk et al., 2022).

2.2 Das MARS-Framework

Das *Multi-Agent Research and Simulation* (*MARS*) Framework ist eine Software-Bibliothek zum Entwickeln und Durchführen von Multiagentensystemen. MARS wird an der HAW Hamburg in der Programmiersprache C# entwickelt und wird unter der *General Public License* (GNU) zur freien Verwendung bereitgestellt. Das Framework ist gleichermaßen für Anfänger und erfahrene Modellierer von Multiagentensystem gedacht. Es soll mit der Verfügbarkeit als *NuGet*-Paket einen leichten Einstieg bieten. Beispielsweise durch die Integration verschiedener spatio-temporaler Daten in die Modelle schafft es zudem die Grundlage für komplexe Simulationen (Clemen et al., 2021; Lenfers et al., 2021).

Der Aufbau von MARS basiert auf zwei Kernkonzepten, den Layers (Schichten) und den Agents (Agenten). Unter Verwendung dieser Konzepte können die gewünschten realen Modelle in konzeptionelle Modelle überführt und in einem Programm umgesetzt werden.

Wie für MAS üblich werden Simulationsläufe in MARS in Simulationsschritte (Ticks) von fester Größe Δt aufgeteilt. Innerhalb jedes Simulationsschritts aktiviert das Framework die aktiven Komponenten des Systems. Ähnlich wie in vergleichbaren Frameworks findet diese Aktivierung über den Aufruf der `Tick`-Methode statt. Diese wird von allen Agenten sowie von einigen Layer-Typen in MARS-Systemen implementiert (Clemen et al., 2022).

2.2.1 Layers

In MARS spielen die Layers eine wesentliche Rolle, da sie dazu dienen, die Umgebung des betrachteten Simulationsszenarios zu modellieren. Diese Welt wird im Framework durch eine Menge von (spatio-temporalen) Layers definiert. Jede dieser Layers spiegelt spezifische Teile des Modells wider und enthält beispielsweise Agenten oder Daten (Clemen et al., 2022).

Layers sind für die Erzeugung der Agenten und Entitäten zuständig und bieten diesen die Schnittstelle, um Informationen über ihre Umwelt zu erfahren. Außerdem ermöglichen sie den Agenten, mit ihrer Umwelt zu interagieren (MARS Group, o. J.-b, o. J.-e).

2.2.2 Agenten

Die Agenten sind der zweite fundamentale Teil einer Simulation in MARS. Sie sind die aktiven Komponenten eines MAS, die ihre Umgebung eigenständig und individuell wahrnehmen. Basierend darauf treffen sie Entscheidungen, wie sie sich verhalten und mit anderen Agenten, Entitäten und der Umwelt interagieren wollen. Jeder Agent ist ein autonom laufendes Programm, welches seine Verhaltensroutine in der zuvor erwähnten `Tick`-Methode definiert. Das Verhalten der Agenten basiert in der Regel auf ihrer Wahrnehmung des Systems, auf ihren individuellen und gemeinschaftlichen Zielen und weiteren Aspekten ihres internen Zustands (Clemen et al., 2021; MARS Group, o. J.-b, o. J.-a).

2.2.3 Entitäten

Entitäten modellieren nicht-aktive Komponenten der Simulation. Das bedeutet, dass sie nicht eigenständig agieren und daher auch über keine `Tick`-Methode verfügen. Sie stellen Objekte oder Gegenstände dar, mit denen die Agenten interagieren können. Durch diese Interaktionen sind Änderungen des Zustands der Entität möglich, beispielsweise wenn sie durch einen Agenten bewegt werden (MARS Group, o. J.-b, o. J.-c).

2.2.4 Environments

Die Environments sind Datenstrukturen, welche für das Positionieren, das Bewegen in der Umgebung und das Erkunden der Umwelt dienen. Sie werden als Teil der Layers verwendet und

bieten unterschiedliche Möglichkeiten, die Bewegung umzusetzen. Ein Beispiel ist das `SpatialHashEnvironment`, welches die Umwelt in Kacheln einteilt und direkte Bewegung von einer Kachel zur anderen ermöglicht. Ein anderes verfügbares Environment ist das `CollisionEnvironment`, welches eine freiere Bewegung im Raum über Vektoren bietet und dabei auf Kollisionen zwischen den enthaltenen Komponenten prüft. Sie helfen zudem, Aspekte des Umfelds wahrnehmbar zu machen, sodass Agenten Erkenntnisse über den Zustand der Umgebung erlangen können (MARS Group, o. J.-d).

2.3 Goal-Oriented Action Planning

Goal-Oriented Action Planning (GOAP) ist die zielorientierte Aktionsplanung. Diese Methode kommt aus dem Bereich der künstlichen Intelligenz und wurde in der Spieleentwicklung als eine Erweiterung des *Stanford Research Institute Problem Solver (STRIPS)*-Modells entwickelt. Diese Entscheidungsfindungs-Architektur kombiniert den Ansatz von adaptivem Verhalten mit Multiagentensystemen.

In der GOAP-Architektur kommen drei essenzielle Bestandteile zum Einsatz: Der Zustand (State), Aktionen (Actions) und Ziele (Goals). Die Agenten verfügen über einen inneren Zustand, welcher ihre Wahrnehmung des Umfelds beschreibt. Dieser Zustand besteht aus einer Menge von Zustandsattributen (Properties), welche zumeist als boolesche Werte dargestellt werden. Durch die Auswertung eines auf internen und externen Sensordaten basierenden Prädikats (Predicate) wird die Ausprägung jedes dieser Zustandsattribute in jedem Simulationsschritt ermittelt. Im Kontext von Teamsportarten könnte ein solches Zustandsattribut beschreiben, ob der Agent in Ballbesitz ist. Hat der Spieler den Ball, ist das entsprechende Attribut `HasBall` mit dem booleschen Wert `true` ausgeprägt.

Weiterhin stehen den Agenten eine Menge von Aktionen zur Verfügung, welche bestimmte Verhaltensweisen beschreiben und den Agenten von einem Zustand in einen anderen überführen. Für jede Aktion ist jeweils eine Menge an Vorbedingungen definiert. Diese beschreiben, welche konkreten Ausprägungen bestimmter Zustandsattribute gegeben sein müssen, damit die Aktion ausgeführt werden kann. Beispielsweise könnte für eine Aktion `PassBall` die Vorbedingung definiert sein, dass der Agent in Ballbesitz, also `HasBall == true` erfüllt, ist. Zudem ist für jede Aktion eine Menge von Nachbedingungen oder Effekten definiert, welche

beschreibt, welche Ausprägungen bestimmter Zustandsattribute nach einer erfolgreichen Ausführung der Aktion gegeben sind. Eine Nachbedingung der `PassBall`-Aktion könnte dementsprechend sein, dass der Agent nach Ausführung nicht mehr im Ballbesitz, also `HasBall == false` ist. Außerdem ist es möglich, Kosten für die einzelnen Aktionen zu definieren. Zuletzt haben Agenten eine Menge von Zielen, welche jeweils durch eine Menge von Paaren von Zustandsattributen und deren Ausprägung definiert werden. Diese beschreiben den zu erreichenden Zustand, um das jeweilige Ziel zu erfüllen. Ein Ziel verfügt über einen Relevanz-Wert und kann eigenständig durch Auswertung eines Prädikats seine aktuelle Relevanz bestimmen. Dieses Prädikat bewertet den gegebenen Zustand, um festzustellen, ob das Ziel aktiviert werden soll, beziehungsweise aktuell eine höhere Priorität hat. Auf diese Weise können indirekt auch für die Ziele Vorbedingungen definiert werden. Des Weiteren können Ziele selbst auswerten, ob sie durch einen Zustand erfüllt wurden (Lenfers et al., 2018; Orkin, Jeff, 2008).

Basierend auf diesen Bestandteilen wird in GOAP ein System namens Planner verwendet, welches die Aufgabe hat, den gegebenen Aktionsraum zu durchsuchen, um valide Sequenzen von Aktionen zu finden. Eine solche Sequenz von Aktionen wird Plan genannt. Valide Sequenzen von Aktionen sind solche, deren Aktionen den Agenten von einem Ausgangszustand in den gewünschten Zielzustand bringen. Der Planner agiert dabei nach dem *Best-First*-Ansatz und wählt in jedem Fall den Plan mit den geringsten Kosten. Dieser Prozess wird als Formulieren eines Plans bezeichnet. Sobald ein Agent einen Plan formuliert hat, verfolgt er diesen, indem er die Aktionen sequenziell ausführt. Dies tut er so lange, bis der Plan abgeschlossen oder invalidiert wurde oder ein anderes Ziel eine höhere Relevanz als das aktuell ausgewählte erlangt. Ein Plan wird invalidiert, wenn für die kommenden Aktionen festgestellt wird, dass der Zustand zum Zeitpunkt der voraussichtlichen Ausführung nicht mehr die Vorbedingungen erfüllt (Lenfers et al., 2018; Orkin, Jeff, 2008). Abbildung 1 zeigt eine beispielhafte Formulierung eines Plans von einem Startzustand zu einem Zielzustand innerhalb eines gegebenen Aktionsraum.

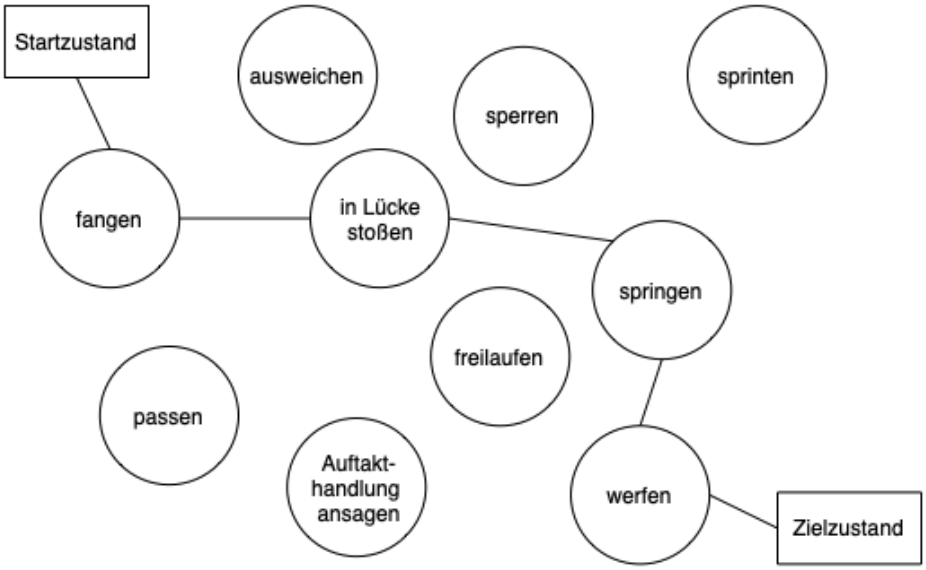

Abbildung 1: Der Prozess des Formulierens eines Plans mit beispielhaften Aktionen des Handballs. Basierend auf (Orkin, Jeff, 2008).

Die Verwendung von GOAP bringt einige Vorteile mit sich, denn es erleichtert das Implementieren von Agenten, welche variables, komplexes und interessantes Verhalten zeigen. Dies geschieht durch das adaptive Planen zur Laufzeit, welches auf die Änderungen des Umfelds reagiert und neue Pläne formuliert, wenn dies nötig ist. Zudem bietet GOAP einen Ansatz, welcher auf natürliche Weise zum Schreiben von strukturiertem Code anleitet, der zudem einfach zu warten und wiederverwendbar ist (Orkin, Jeff, 2008).

2.4 Handball als Sportart und Forschungsobjekt

Der Handballsport ist eine Mannschaftssportart, in der zwei Mannschaften gegeneinander antreten. In der Regel sind je Team 7 Spieler auf dem Spielfeld, die in 6 Feldspieler und einen Torwart eingeteilt sind. Die Anzahl und Aufteilung können während eines Spiels variieren, da Spieler durch Zeitstrafen für eine Dauer von zwei Minuten dem Spielfeld verwiesen werden können. In dieser Zeit darf das Team den suspendierten Spieler nicht ersetzen. Der Torhüter darf durch einen zusätzlichen Feldspieler substituiert werden, was allerdings normalerweise nur im Angriff genutzt wird.

Die Mannschaften treten gegeneinander an, um durch geschicktes Zusammenspiel und strategische Manöver (wie beispielsweise Auftakthandlungen) den Ball in das gegnerische Tor zu werfen. Das Spiel erfordert eine Kombination aus körperlicher Ausdauer, Schnelligkeit, Kraft und taktischem Verständnis. Handball ist bekannt für sein schnelles Tempo, häufige Torwürfe und intensive Zweikämpfe. Ziel des Spiels ist, wie oftmals in Mannschaftssportarten, mehr Punkte zu erzielen als das gegnerische Team.

2.4.1 Regelwerk

Das Spielfeld im Handball ist in Abbildung 2 dargestellt und verfügt über einige Bereiche, für die besondere Regeln gelten. In der Abbildung sind diese mit Ziffern markiert, die in der nachfolgenden Beschreibung zur Nachvollziehbarkeit referenziert werden. Das Spielfeld ist 20 Meter breit und 40 Meter lang. In der Mitte der beiden Spielfeldenden befindet sich je ein Tor (1), welches wie zuvor erklärt den Mittelpunkt des Spiels darstellt. Die Tore sind jeweils drei Meter breit und zwei Meter hoch. Ein Tor wird erzielt, sobald der Ball die Torlinie mit vollem Umfang überquert hat. Um das Tor herum befinden sich zwei halbe ovalförmige Bereiche. Der kleinere Bereich (2a) ist der Torraum, auch „6-Meter-Kreis“ genannt. Diesen Bereich darf nur der Torhüter des Teams, dem die Seite zugeordnet ist, dauerhaft betreten. Sollte ein Feldspieler den Torraum betreten ist es ihm während des Betretens nicht erlaubt in das Spiel einzugreifen, oder einen Vorteil aus dem Betreten zu erlangen. Dies gilt genauso für die „6-Meter-Linie“ (2b), die den Torraum begrenzt. Betritt ein Feldspieler diesen Bereich, während er aktiv am Spiel teilnimmt, indem er zum Beispiel den Ball führt, den Ballführer verteidigt oder zum Ort des aktiven Geschehens läuft, resultiert dies in Ballbesitz für die andere Mannschaft. Bei einem solchen Regelverstoß durch das angreifende Team nennt man dies „Übertritt“, bei Verstoß durch die Abwehr wird es „Abwehr durch den Kreis“ genannt. Ein wesentlicher Teil des Spiels ist, dass die Spieler, insbesondere der angreifenden Mannschaft, in den Torraum hineinspringen, um bei einem Torwurf näher zum Tor zu kommen. Während des Sprungs darf sich der Spieler im Torraum aufhalten.

Das größere Oval (3a) ist der „9-Meter-Kreis“. Dieser Bereich darf, anders als der Torraum, im normalen Spielverlauf von den Feldspielern betreten werden. Im Falle eines Freiwurfs für

die angreifende Mannschaft wird dieser von der umgebenen Freiwurflinie (3b) ausgeführt. Nur in dieser Situation ist es den Angreifern verboten, den „9-Meter-Kreis“ zu betreten.

Für schwerwiegende Fouls oder Verhinderungen klarer Torchancen wird eine besondere Art des Freiwurfs zugesprochen, der „7-Meter“ genannt wird. Dieser wird von der Linie (5), die sich sieben Meter vom Tor entfernt im 9-Meter-Kreis befindet, ausgeführt. Während dieses Wurfs darf lediglich der Schütze innerhalb des 9-Meter-Kreises sein. Der Wurf ist dementsprechend ein freier Wurf gegen den Torwart. Der Torwart darf in dieser Situation vor seinem Tor stehen, jedoch nicht weiter heraustreten als vier Meter, was durch die kleine Linie (4) im Torkreis markiert wird.

Zu Beginn einer Halbzeit und nach einem Torerfolg wird das Spiel durch einen Mittelanwurf gestartet. Bei der Ausführung muss sich der anwerfende Spieler stehend in dem Mittelkreis (6b) befinden. Alle Mitspieler müssen sich in der Hälfte ihres Teams hinter der Mittellinie (6a) aufhalten.

An den langen Seiten des Spielfelds befinden sich die Seitenauslinien (7). Überquert der Ball diese, oder tritt ein Spieler, der den Ball berührt, auf oder über diese Linien, resultiert dies in einem Einwurf für die Mannschaft, die nicht zuletzt den Ball berührt hat.

Wird die Torauslinie an den kurzen Enden des Feldes vom Ball oder Spieler überquert, führt dies zu einem Abwurf durch den Torwart, wenn zuletzt ein Angreifer oder der verteidigende Torwart den Ball berührt haben. Dieser wird vom Torhüter aus dem Torraum heraus ausgeführt. War zuletzt ein Feldspieler der verteidigenden Mannschaft am Ball, gibt es eine Ecke für die angreifende Mannschaft. Diese wird von der Schnittstelle der Toraus- und der Seitenauslinie ausgeführt (Kolodziej, 2013).

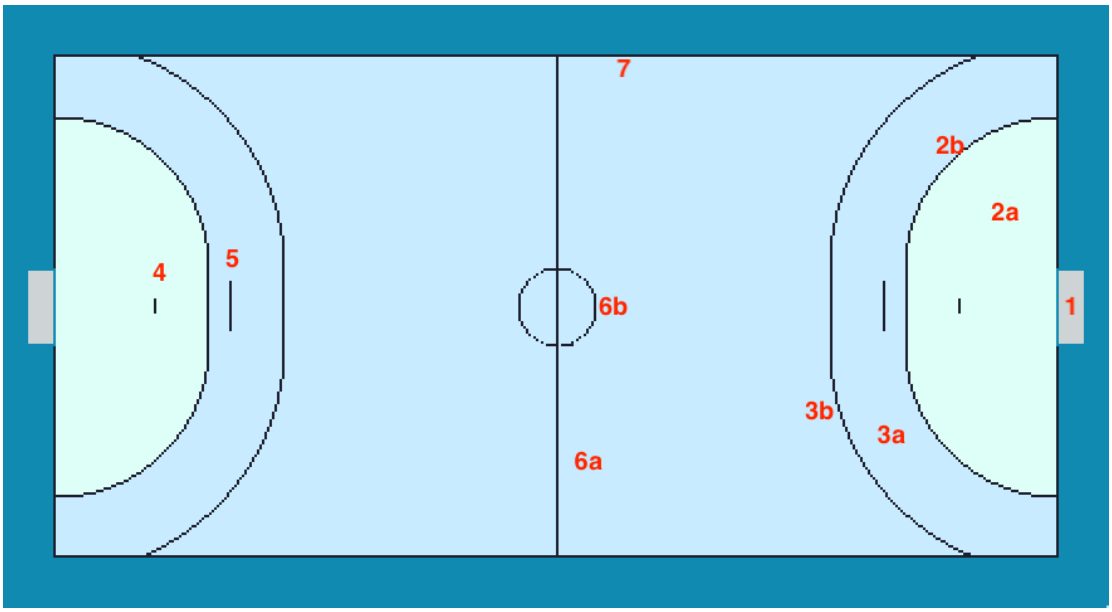

Abbildung 2: Das Spielfeld im Handball mit Markierungen bestimmter Bereiche.
(Darstellung innerhalb des entwickelten Visualisierungsprogramms).

Das Regelwerk von Handball besagt zudem, dass Feldspieler den Ball nicht mit ihrem Fuß berühren dürfen, andernfalls wird ein Freiwurf für das gegnerische Team ausgesprochen. Der Torhüter hingegen darf zum Abwehren des Balls seinen ganzen Körper verwenden, sofern dies innerhalb des Torkreises geschieht. Außerhalb gelten für ihn die Regeln wie für Feldspieler. Hält ein Feldspieler den Ball in seinen Händen, darf er maximal drei Schritte mit dem Ball machen, sonst wird ein Freiwurf für das andere Team gegeben. Durch Prellen – das fortlaufende zu Boden tippen des Balls mit der Hand – sind dem Spieler jedoch weitere Schritte möglich. Wird nach Beginn des Prellens der Ball wieder aufgenommen, darf nicht erneut begonnen werden zu Prellen. Auch hier führt ein Verstoß zu einem Freiwurf für die Gegner. Ein Spieler darf den Ball zudem nur maximal drei Sekunden in den Händen halten, ohne dass er sich bewegt oder prellt. Eine angreifende Mannschaft muss durch ihre Bemühungen Torgefahr ausstrahlen, sonst kann der Schiedsrichter durch das Heben des Arms passives Spiel signalisieren. Ab dem Zeitpunkt des Signals dürfen noch maximal vier Pässe erfolgen, bis ein Torwurf geschehen muss. Werden mehr Pässe gespielt oder erkennt der Schiedsrichter weiterhin eine übermäßige Passivität, wird der verteidigenden Mannschaft der Ballbesitz zugesprochen. Sind bereits vier Pässe gespielt und die Angreifer erhalten einen Freiwurf, so darf dieser immer

ausgeführt werden, ohne als fünfter Pass gezählt zu werden. Wann das Signal für passives Spiel gezeigt wird, liegt im Ermessen der Schiedsrichter.

Für bestimmte Arten von Fouls gibt es im Handball die progressive Bestrafung. Hier wird zunächst mit einer gelben Karte verwarnt, danach Zweiminutenstrafen ausgesprochen, die zuvor bereits beschrieben wurden. Erhält ein Spieler seine dritte Zweiminutenstrafe, resultiert dies in einer roten Karte. Diese hat für die Mannschaft die gleichen Konsequenzen wie die Zweiminutenstrafe, nämlich eine zweiminütige Unterzahl. Für den Spieler bedeutet sie den permanenten Ausschluss vom Rest des Spiels. Hat ein Team insgesamt bereits drei gelbe Karten erhalten, so beginnt die Progression nachfolgend direkt mit Zweiminutenstrafen. Härtere Strafen als die nächste in der Progressionsstufe können für besonders harte Fouls immer ausgesprochen werden (International Handball Federation, 2024).

2.4.2 Handball als Multiagentensystem

Mannschaftssportarten bieten ein interessantes Anwendungsgebiet für Multiagentensysteme, denn sie bieten ein klar definiertes Umfeld, welches auf einen kleinen Raum beschränkt ist. Die Regeln für Interaktionen innerhalb einer Simulation sind eindeutig festgelegt. Die Agenten stellen Spieler dar, die in dem Spiel eigenständig agieren. Ihre Entscheidungen werden durch übergeordnete Ziele für ihre Mannschaft und dem Wettbewerb mit den Gegnern beeinflusst. So müssen die Agenten im Sinne des Mannschaftserfolgs zusammenarbeiten (Prokopenko & Wang, 2016). Die Spieler im Handball bewegen sich teils gleichzeitig, was ein dynamisches und komplexes System erzeugt, in dem sie ihre Entscheidungen treffen müssen. Um das Verhalten der Agenten zu modellieren und das Modell zu bewerten, besteht zudem in Sportarten ein gutes Verständnis und eine Wissensbasis, um dies zu tun. Darüber hinaus gibt es Beispiele aus der realen Welt, wie Aufnahmen der Spiele, um Vergleiche anzustellen (Saeed Rahimi et al., 2022).

Im Handball kommt hinzu, dass das Verhalten von bestimmten taktischen Anweisungen gesteuert wird, die der Trainer einer Mannschaft vorgibt. Diese Anweisungen bestehen einerseits aus der Position oder Rolle, die einem Spieler zugeordnet ist. Die Rollen im Angriff sind in Abbildung 3 im linken Bild zu sehen. Die Spieler an den beiden Seiten sind der Rechtsaußen (RA) und Linksaußen (LA). Der Spieler, der sich an der 6-Meter-Linie befindet, wird

Kreisspieler (KS) oder Kreisläufer genannt, während die drei Spieler, die weiter vom Tor entfernt sind, im sogenannten Rückraum positioniert sind. Dementsprechend heißen ihre Rollen Rückraum Links (RL), Rückraum Mitte (RM) und Rückraum Rechts (RR). In der Verteidigung werden die äußeren Rollen mit Außen Links (AL) und Außen Rechts (AR) sowie Halb Links (HL) und Halb Rechts (HR) benannt. Befinden sich die inneren beiden Spieler hintereinander, so nennt man die Rollen Hinten Mitte (HM) und Vorne Mitte (VM). Sind sie nebeneinander aufgestellt, so handelt es sich um den Innen Links (IL) und Innen Rechts (IR).

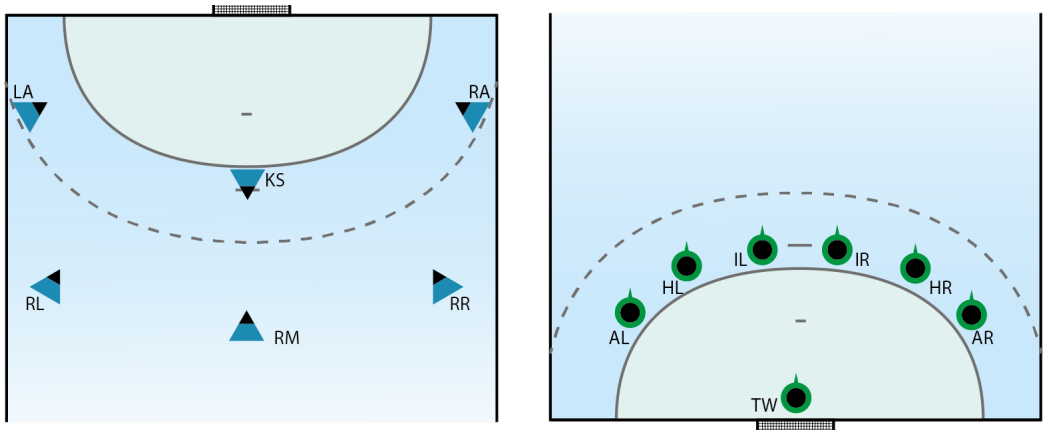

Abbildung 3: Die Rollen der Spieler in einer 3:3-Angriffsformation (links) und einer 6:0-Abwehformation (rechts). Basierend auf (Kolodziej, 2013).

In der Verteidigung ist die Anweisung die Abwehrformation, die eingenommen werden soll. Diese wird im Handball meistens durch Zahlenkombinationen ausgedrückt, welche aussagen, wie viele Spieler in welcher Ebene der Abwehr verteidigen sollen. Beispielsweise wird bei einer „6-0“-Abwehrformation mit allen 6 Feldspielern defensiv, also nahe der 6-Meter-Linie verteidigt. Bei einer „5-1“-Abwehrformation sind hingegen nur fünf Spieler in dieser defensiven Position, währen ein Spieler offensiver verteidigt, also an der Freiwurflinie oder noch weiter vom eigenen Tor entfernt (Kolodziej, 2013).

Für den Angriff sind normalerweise in jeder Mannschaft verschiedene Abläufe definiert, welche die Mannschaft verwenden kann, um die Abwehr auszuspielen und gute Torwurfpositionen zu kreieren. Diese nennt man Auftakthandlungen, welche Lauf- und Passwege vorgeben, dabei jedoch jederzeit abgebrochen werden können, um eine sich ergebende Torchance zu nutzen. Sie werden vom Trainer vorgegeben und im Training eingeübt. Im Spiel ist dann der

Spielführer – meistens der Rückraum Mitte Spieler – für die Auswahl eines bestimmten Ablaufs zuständig. Diesen sagt er namentlich laut an, um seine Mitspieler zu informieren und gegebenenfalls den Startpunkt anzukündigen. Zum Teil werden in den Auftakthandlungen bestimmte Punkte vorgesehen, an denen der Ballführer eine Entscheidung zwischen mehreren Möglichkeiten für die Fortsetzung des Angriffs treffen muss. Zudem können Spieler jederzeit entscheiden, den Ablauf abzubrechen, wenn sich stattdessen unvorhergesehen eine gute Wurfmöglichkeit für ihn selbst oder einen anderen Mitspieler ergeben hat oder andere Faktoren des Spiels dies verlangen.

2.4.3 Bewertung der Torwahrscheinlichkeit durch Expected Goals-Werte

In Mannschaftssportarten werden zunehmend statistikbasierte Modelle eingesetzt, um erzielte Ergebnisse auf Basis gemessener Werte zu erklären oder Anomalien zu erkennen. Eines dieser Modelle sind die sogenannten *Expected Goals* (*XGoals*). Besonders im Fußball hat sich dieses Konzept bereits weit verbreitet. Dabei bewertet das Modell Spielsituationen, in denen Torschüsse abgegeben wurden, basierend auf einer Reihe von spielbezogenen Faktoren. Anhand gelernter Gewichtungen dieser Faktoren wird berechnet, wie hoch die Wahrscheinlichkeit war, in einer bestimmten Situation ein Tor zu erzielen.

Auch im Handball wurde ein solches System von Adams et al. (2023) entwickelt. Dieses Modell bewertet Torwürfe anhand von spielbezogenen Aspekten wie der Entfernung des Schützen zum Tor, der Position des Torhüters und der Distanz zwischen dem Schützen und dem Torhüter. Weitere bewertete Faktoren sind die Entfernung zum nächsten Verteidiger, der Winkel des Schützen zum Tor sowie die Anzahl der Spieler, die sich im Wurfweg befinden. Zudem wird der Druck, den die Verteidiger auf den Schützen ausüben, berücksichtigt. Die Autoren haben diese Beziehungen durch den Einsatz von maschinellem Lernen auf Basis von Daten aus einer Saison der Handball-Bundesliga erarbeitet.

Zwar werden in der Arbeit keine genauen Werte für die genannten Faktoren angegeben, jedoch können diese näherungsweise aus den Grafiken im Artikel entnommen werden. Beispielsweise zählen der Winkel zum Tor, die Entfernung des Torhüters zum Tor sowie die Distanz zwischen Schütze und Torhüter zu den einflussreichsten Faktoren.

Obwohl die probabilistische Natur des Modells bedeutet, dass eine exakte Vorhersage der Torwahrscheinlichkeit nie garantiert werden kann, lassen sich XGoals-Werte dennoch in die Bewertung eines Spiels und einzelner Situationen einbeziehen.

2.5 Vergleichbare Multiagenten-Systeme im Bereich Sport

Wie bereits in Kapitel 2.4.2 beschrieben, sind Mannschaftssportarten ein Gebiet, in dem Multiagentensysteme zum Einsatz kommen. Da in dieser Arbeit ein weiteres solches System für den Handballsport entwickelt wird, werden hier vergleichbare bestehende Systeme betrachtet. Es wird herausgearbeitet, wie diese relevanten Aspekte des Teamsports umsetzen und zudem betrachtet, wie Agenten in diesen Systemen modelliert werden. Dies soll dazu dienen, eine Grundlage für die folgende Analyse und Design des neuen Systems zu bieten.

Im Mannschaftssport treten in der Regel zwei Teams in einem Spiel gegeneinander an. Beim Handball sowie bei anderen Sportarten besteht das Spiel aus einer Menge von Sequenzen, in denen eine Mannschaft angreift, während die andere Mannschaft verteidigt. Hierbei verfolgen die beiden Mannschaften gegenteilige Ziele, da die angreifende Mannschaft probiert, ein Tor zu erzielen. Die verteidigende Mannschaft hat das Ziel, genau dies zu verhindern und kein Gegentor zu kassieren. Somit können die einzelnen Situationen einer Sportart als Nullsummenspiel betrachtet werden, indem das Erreichen des Ziels durch ein Team bedeutet, dass das gegnerische Team sein Ziel verfehlt. Auf ein gesamtes Spiel gesehen ist der Status als Nullsummenspiel zumeist nicht gegeben, da auch ein Unentschieden als Ausgang eines Spiels infrage kommt. Dieses Nullsummenspiel ist für die Bewertung der einzelnen Situationen durch die Agenten relevant. Wenn ein gesamtes Spiel betrachtet wird, so wäre auch der aktuelle Spielstand zu bewerten, um das Verhalten in Richtung Sieg oder zumindest Unentschieden auszurichten.

Einige agentenbasierte Modellierungen wurden im Bereich des Teamsports bereits umgesetzt und als Basis für diese Arbeit herangezogen. Das bekannteste System ist der *RoboCup Soccer Server*, ein MAS für Fußball, für den ein jährlicher Wettbewerb stattfindet, bei dem Forschungsteams mit ihren Agenten-Implementierungen gegeneinander antreten. Das Ziel dieser Initiative ist es, die Kapazitäten von Agentenmodellierungen stetig zu verbessern und bis zum Jahr 2050 eine Mannschaft aus fußballspielenden Robotern erschaffen zu können, die in der

Lage sind, den amtierenden Fußballweltmeister in einem Spiel zu besiegen. Ein weiteres System ist der *Simple Team Sport Simulator 2 (STS2)*, welcher für die Forschung in Zusammenarbeit mit Electronic Arts entwickelt wurde. Dieses System ist darauf ausgelegt, unterschiedliche Sportarten zu simulieren. In der bisherigen Umsetzung liegt dessen Fokus jedoch zunächst nur auf der Simulation von Eishockey. Zudem existiert das *RoboNBA*-System, welches Basketball als Domäne für die Umsetzung als MAS ausgewählt hat. Bei all diesen Systemen wird ein zweidimensionales Modell der jeweiligen Sportart umgesetzt. Die Simulation läuft bei allen Tick-basiert ab, wie es auch in MARS Simulationen der Fall ist (siehe Kapitel 2.2). Der RoboCup Soccer Server ist ein verteiltes System, in dem die Simulation auf einem Computer ausgeführt wird, während die Steuerung der Spieler über verteilte Methodenaufrufe unter Verwendung des UDP/IP Protokolls von anderen Geräten stattfinden kann. Die anderen beiden Systeme hingegen sind jeweils in sich geschlossene Systeme, wo sowohl die Simulation als auch die Spieler innerhalb desselben Programms laufen.

Andere gefundene Systeme modellieren ein dreidimensionales System statt der hier erwähnten 2D-Umsetzungen. Da die Komplexität und der Fokus dieser Systeme stark von der des zu entwickelnden Systems abweichen wurden diese Systeme nicht weiter betrachtet.

Nachfolgend werden relevante Teile der Modelle beschrieben und gegenübergestellt.

Sensorik der Agenten in den betrachteten Systemen

Sowohl in dem Simple Team Sport Simulator 2 (STS2) als auch im RoboNBA System ist die Sensorik der Agenten einfach gehalten. In STS2 erhalten die Agenten vollständige Informationen über ihr gesamtes Umfeld, während in RoboNBA nur Spieler und Objekte, die sich innerhalb eines Radius um den Agenten befinden, gesehen werden können. Hierbei sind aber auch Dinge, die hinter dem Agenten positioniert sind, sichtbar. Der RoboCup Soccer Server hingegen setzt drei Arten von Sensoren um, über welche die Agenten ihr Umfeld wahrnehmen können. Diese sind visuelle Sensoren, auditive Sensoren sowie Körper Sensoren. Letztere bieten den Agenten Informationen über ihren physischen Zustand, also beispielsweise die verbleibenden Ausdauerpunkte, die ihnen zur Verfügung stehen (Caedmon Somers, Jason Rupert, Yunqi Zhao, Igor Borovikov, Jiachen Yang, Ahmad Beirami, 2020; Hu et al., 2005; The RoboCup Soccer Simulator Maintenance Committee, 2024; Yang et al., 2020).

Über diese Sensorik werden die anderen Agenten und Entitäten des Systems wahrgenommen. Dabei gibt es jedoch Limitierungen, sodass nur Dinge innerhalb des Blickwinkels des Agenten sichtbar werden. Zudem hat die Entfernung zu den jeweiligen Akteuren eine Relevanz, da den erhaltenen Informationen basierend darauf Rauschen hinzugefügt wird, wodurch weiter entfernte Agenten ungenauer wahrgenommen werden. Zudem können Nachrichten von Agenten, die zu weit entfernt sind, gar nicht gehört werden. Ähnlich ist es bei visuellen Informationen, wo mit wachsender Entfernung immer weniger spezifische Informationen zur Verfügung stehen (The RoboCup Soccer Simulator Maintenance Committee, 2024).

Bewegungsmodelle und Aktionen in den betrachteten Systemen

Die betrachteten Systeme behandeln die Agenten auf unterschiedliche Weise. Während die Agenten in STS2 direkt in dem Umfeld eingesetzt sind und sich selbst innerhalb dessen steuern können, ist beim RoboCup Soccer Server der Ansatz gewählt, dass Agenten das Gehirn eines Spielers darstellen und über Aktionen einen Körper steuern, der sich in der Umwelt befindet. Für RoboNBA ist keine Angabe gemacht, auf welche Art dies umgesetzt wurde (Caedmon Somers, Jason Rupert, Yunqi Zhao, Igor Borovikov, Jiachen Yang, Ahmad Beirami, 2020; The RoboCup Soccer Simulator Maintenance Committee, 2024).

Die Bewegungen der Agenten sind sowohl bei STS2 als auch bei RoboNBA sehr vereinfacht gehalten, da die Agenten und der Ball sich hier genau in vier Richtungen bewegen können. Zudem ist hier jeweils nur genau eine Schrittgröße vorgesehen. So können die Agenten sich in dem Kachel-Umfeld je Tick in eine der benachbarten Kacheln bewegen. Diese Schritte können in RoboNBA lediglich über die `run`-Aktion ausgeführt werden, während in STS2 keine spezifische Aktion dafür vorgesehen ist, sondern Bewegung nur als Teil anderer Aktionen stattfindet. Im STS2 ist das Passen des Balls durch die fünf Methoden `PASS_1` bis `PASS_5` beschrieben, womit jeweils der Passempfänger im Namen der Aktionen über die Ziffer definiert ist. Zudem bietet die `SHOOT`-Aktion die Möglichkeit, auf das Tor zu schießen. Konkrete Richtungen können hier nicht spezifiziert werden. In RoboNBA hingegen gibt es diese Möglichkeit, da sowohl bei der `shoot`- als auch der `pass`-Aktion Argumente für die Richtung und Stärke der Ausführung übergeben werden müssen (Caedmon Somers, Jason Rupert, Yunqi Zhao, Igor Borovikov, Jiachen Yang, Ahmad Beirami, 2020; Hu et al., 2005; Yang et al., 2020).

Im RoboCup Soccer Server ist den Agenten mehr Freiheit in ihrer Bewegung geboten, da sie sich in alle 360 Winkel bewegen können. Dafür kann sich der Agent durch die `turn`-Aktion ausrichten und kann über die `dash`-Aktion einen Schritt mit einer angegebenen Stärke machen. Zudem kann zusätzlich eine Richtung angegeben werden, ansonsten wird der Schritt in die Richtung, in die der Körper ausgerichtet ist, gemacht. Die Bewegung des Balls wird zudem durch die `kick`-Aktion ausgelöst, wo ebenfalls die Richtung und Stärke übergeben werden. Zudem ist für die Torhüter die `catch`-Aktion verfügbar, um den Ball zu stoppen und sicher an sich zu führen, ohne dass dieser einem abgenommen werden kann. Für das Blockieren des Wegs des Balles zum Tor gibt es jedoch keine besondere Methode, sondern es wird die `dash`-Methode genutzt. Diese Aktionen gehören dabei zu einer Kategorie von Aktionen, von denen nur eine je Tick ausgeführt werden kann (The RoboCup Soccer Simulator Maintenance Committee, 2024).

Kommunikation in den betrachteten Systemen

Ähnlich wie in den vorherigen Abschnitten sind STS2 und RoboNBA auch in dem Bereich der Kommunikation einfacher umgesetzt. In diesen beiden Systemen ist keine Kommunikation zwischen den Agenten vorgesehen. Kooperative Verhaltensweisen müssen also ohne Kommunikation auskommen (Caedmon Somers, Jason Rupert, Yunqi Zhao, Igor Borovikov, Jiachen Yang, Ahmad Beirami, 2020; Hu et al., 2005; Yang et al., 2020).

Anders ist dies in dem RoboCup Soccer Server, wo die Kommunikation als wichtiger Teil betrachtet wird. Hier können Agenten über die `say`-Funktion eine Nachricht versenden. Diese wird als Broadcast Nachricht an sämtliche andere Agenten verschickt. Nachrichten, die in einer zu großen Entfernung abgesetzt werden, sind für Agenten außerhalb der Reichweite nicht empfangbar. Da der Inhalt der Nachrichten auf 10 Zeichen begrenzt ist, ist es hier für die Teams notwendig, dass ihre Agenten Informationen auf irgendeine Weise codieren und empfangene Nachrichten decodieren. Um zu verhindern, dass eine Mannschaft ihre Gegner mit Nachrichten überfluten kann und so effektive Kommunikation unterbindet, wird die Zustellung in drei Gruppen unterteilt: Nachrichten der Mitspieler, Nachrichten der Gegenspieler und Nachrichten des Schiedsrichters. Zudem ist ein Zustellungslimit von maximal einer Nachricht je Mannschaft in einem Tick festgelegt, sodass Kommunikation relevant und effizient stattfinden muss (The RoboCup Soccer Simulator Maintenance Committee, 2024).

Umsetzungen der Kooperation der Teammitglieder

Ohne effektive Kooperation zwischen mehreren Spielern ist es schwierig, die Mannschaftsleistung zu erhöhen. Daher werden hier Methoden betrachtet, die in den drei Systeme zu diesem Zweck zum Einsatz kommen.

Eine viel verwendete Methode, die in den Arbeiten zu STS2 und RoboNBA als primäre Methode zum Einsatz kommt und auch in RoboCup Soccer Server bei einigen der Forschungsteams Verwendung findet, ist das *Hierarchische Lernen*. Bei dieser Methode wird der Lernprozess in mehrere Stufen eingeteilt. In der untersten Stufe werden die simplen Methoden des Systems erlernt, die den Agenten zur Verfügung stehen. Diese sind sozusagen atomare Aktionen, die in sich abgeschlossen sind, wie die zuvor beschriebenen Aktionen `dash`, `turn` und `shoot`. Auf der höheren Stufe werden Verhaltensmuster erlernt, bei denen die Agenten die Aktionen der unteren Stufe kombinieren, um so die Ziele basierend auf dem aktuellen Zustand des Umfelds zu erreichen. Als Lernalgorithmus kommt hier oftmals *Reinforcement Learning* zum Einsatz. Sind die notwendigen Daten vorhanden und nutzbar, ist auch *Imitation Learning* gut geeignet, da dadurch das kooperative Verhalten einfacher erlernt werden kann. Anderenfalls ist zu diesem Zweck eine wohldefinierte Belohnungsfunktion nötig, um das erlernte Verhalten wie gewünscht zu steuern.

Die hierarchische Aufteilung dient dem Zweck, die Komplexität des Lernprozesses zu verringern, indem das Problem des komplexen Verhaltens in kleinere Probleme aufgeteilt wird, die voneinander gekapselt gelöst werden können. Dies entspricht dem Ansatz von Multiagentensystemen, die ebenfalls die Reduktion der Komplexität durch Teilung des Problems bezwecken. Allerdings erscheint der Zeitaufwand in der Entwicklung bei dieser Methode relativ hoch zu sein, da die damit arbeitenden Forschungsteams in mehreren Fällen keine Umsetzung für ganze Mannschaften vorgenommen haben, sondern lediglich für einige Spieler, weil diese separat trainiert werden müssen (Liu et al., 2021; Yang et al., 2020; Zhao et al., 2019).

Ein weiterer interessanter Ansatz ist die Methode des *Cooperative Action Planning,* welches von Akiyama et al. (2012) im RoboCup Soccer Server zum Einsatz kommt. Bei dieser werden Aktions-Zustandspaare definiert und, beginnend von dem aktuellen Zustand eine *Tree Search* gestartet. Kanten in dem entstehenden Baum sind Aktionen und Knoten sind die Zustände. Ein

Modul bewertet während der Erstellung des Baumes für jeden möglichen Knoten die Wahrscheinlichkeit des Erfolgs. Nur wenn ein Knoten als Erfolg bewertet wird, wird dieser dem Baum hinzugefügt. Diese Suche wird fortgeführt, bis eine Maximallänge oder eine terminierende Aktion erreicht ist. Abschließend wird nach dem Best-First-Ansatz gewählt, also der Pfad, welchem die höchste Erfolgswahrscheinlichkeit zugeordnet wurde. Ein wichtiger Aspekt dieser Methode ist, dass lediglich der Agent in Ballbesitz eine Suche durchführt und nachfolgend alle seine Mitspieler per Nachricht darüber informiert. Der Grund hierfür ist, dass dieser Spieler derjenige ist, der den Angriff einleitet und somit die anderen Spieler seiner Idee folgen sollen. Da Nachrichten verloren gehen können, wird in jedem Tick eine erneute Suche durchgeführt und diese erneut den Mitspielern mitgeteilt.

Es ist leicht zu erkennen, dass bei dieser Methode gewisse Parallelen zu GOAP bestehen, da ebenfalls ein Baum für die Suche nach einem optimalen Pfad verwendet wird. Auch bei GOAP nutzt der Planner den Best-First-Ansatz, wählt aber den Pfad mit der geringsten Anzahl an Aktionen beziehungsweise den geringsten Kosten aus. Allerdings sind bei GOAP die Ziele, Aktionen und Zustände so definiert, dass der Planner mit Sicherheit Pfade aufbauen kann, die zum Erfolg führen (sollten solche verfügbar sein). Sie müssen nicht nach ihrer Wahrscheinlichkeit auf Erfolg bewertet werden. Darüber hinaus ist bei GOAP nicht standardmäßig vorgesehen, dass mit jedem Tick ein neuer Plan erstellt wird. Stattdessen wird in jedem Zeitschritt zunächst geprüft, dass der bestehende Plan in Anbetracht des aktualisierten Zustands des Umfelds weiterhin valide ist. Nur im Falle einer Invalidierung des Plans oder wenn ein Plan vollständig ausgeführt wurde, wird ein neuer Plan formuliert. Ein weiterer wichtiger Unterschied zwischen den beiden Methoden ist, dass bei GOAP jeder Agent eigenständig einen Plan für sich erstellt, bei dem er selbst auswählt, welches Ziel für ihn aktuell am relevantesten ist und welche Schritte er verwendet, um dieses zu erreichen.

3 Anforderungsanalyse

Dieses Kapitel erörtert Anforderungen an das zu entwerfende System. Diese sind unterteilt in funktionale und nicht-funktionale Anforderungen, um eine möglichst detaillierte Betrachtung zu ermöglichen. Die aufgestellten Anforderungen dienen abschließend auch zur Bewertung des Erfolgs der Umsetzung des Systems.

3.1 Funktionale Anforderungen

Die Anforderungen an das System sind hauptsächlich funktionaler Art. Diese werden hier aufgeführt.

REQ-1. Simulation von Spielsituationen des Positionsangriffs im Handballsport

Das System ermöglicht die Simulation einzelner Spielsituationen im Handball durch Agenten. Diese Spielsituationen sind solche des strukturierten Positionsangriffs, bei dem eine Mannschaft klar als angreifend definiert ist und die andere als verteidigend. Dabei werden relevante Regeln des Sports umgesetzt, sofern dies sinnvoll ist.

REQ-2. Realitätsnahe Modellierung der Bewegungen

Die Bewegungen der Agenten und des Balls werden realitätsnah und physikalisch korrekt modelliert.

REQ-3. Konfiguration von Spielsituationen

Das System bietet den Anwendern die Möglichkeit, die Spielsituationen zu konfigurieren. Dabei kann festgelegt werden, wie viele Spieler je Mannschaft auf dem Spielfeld sind, welche Rolle ihnen zugeordnet ist und an welcher Position des Spielfelds sie starten. Außerdem kann definiert werden, wo der Ball positioniert wird und ob sich dieser zu Beginn der Simulation in Besitz eines Spielers befindet.

REQ-4. Definition und Anpassung von Auftakthandlungen

Benutzer können Auftakthandlungen für die angreifende Mannschaft einfach definieren und anpassen. Die definierten Auftakthandlungen können dann von den Spielern in dem System umgesetzt werden.

REQ-5. Kommunikation zwischen Agenten

Das System schafft eine Möglichkeit für die Agenten, um miteinander zu kommunizieren. Die Agenten können diese Funktion nutzen, um Informationen untereinander auszutauschen, Auftakthandlungen anzusagen oder auch für andere Zwecke.

REQ-6. Adaptive Verhaltensweisen der Agenten

Agenten nutzen adaptive Verhaltensweisen, um auf den von ihnen wahrgenommenen Zustand ihres Umfelds zu reagieren und ihr Verhalten entsprechend anzupassen.

REQ-7. Berücksichtigung der Höhe in dem System

Das System soll nicht nur eine zweidimensionale Simulation umsetzen, sondern auch die Höhe simulieren. Diese Dimension ist im Handball ein wichtiger Aspekt des Spiels, da Spieler beispielsweise durch Sprungwürfe eine bessere Wurfposition erreichen können und somit den Ball über die Abwehr auf das Tor werfen können, selbst wenn sich Gegenspieler im Weg befinden.

REQ-8. Grafische Oberfläche zur Analyse von Spielsituationen

Das System bietet einem Benutzer eine grafische Nutzeroberfläche, mittels der die Ergebnisse eines Simulationslaufs betrachtet werden können. Diese dient der Analyse der gegebenen Simulation, sowie der Verwendbarkeit des Systems als Mittel für Trainer für das Aufzeigen und Austesten von Auftakthandlungen.

Als Erweiterung der funktionalen Anforderungen dienen zudem auch Anwendungsfälle, die für dieses System definiert wurden. Dabei handelt es sich um Aktionen, die aus Sicht der Akteure (Spieler) innerhalb der Simulation verfügbar sein sollten, um mit der Umgebung, den anderen Akteuren und dem Ball interagieren zu können. Diese Liste von Anwendungsfällen umfasst 14 solcher Fälle, die in Anhang A in tabellarischer Form dargestellt werden.

3.2 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen eines Systems können zudem nicht-funktionale Anforderungen definiert werden. Diese beschreiben Aspekte, die keine spezifischen Funktionen des Systems bestimmen und oft eine unspezifische Formulierung haben. Um diese Anforderungen zu messen, sollten sie, wenn möglich, quantifiziert oder spezifiziert werden.

REQ-9. Erweiterbarkeit und Veränderbarkeit

Das System wird so entwickelt, dass der Code von anderen Entwicklern einfach weiterentwickelt oder angepasst werden kann. Dies kann beispielsweise durch die Verwendung von Schnittstellen (Interfaces) erreicht werden, welche die dahinterliegende konkrete Implementierung verbergen und so leicht austauschbar machen.

REQ-10. Verwendung des MARS-Frameworks

Das System soll unter Verwendung des MARS-Framework entwickelt werden. Dieses an der HAW Hamburg entwickelte Framework für Multiagentensysteme bietet eine Fülle an Möglichkeiten, um die funktionalen Anforderungen zu erfüllen.

4 Entwurf der Simulations- und Modellumgebung

Um die in Kapitel 3 aufgestellten Anforderungen zu erfüllen, muss für das System ein Konzept erstellt werden, in welchem die gegebenen Vorgaben berücksichtigt werden. In diesem Kapitel werden die Spezifikationen beschrieben und erklärt, wie diese die Anforderungen erfüllen. Dabei wird zudem eine Übersicht über das ganzheitliche Modell gegeben, um dann konkrete Aspekte davon genauer zu betrachten.

Für die Erstellung des Konzepts des Systems wurden vergleichbare Systeme betrachtet, welche in Kapitel 2.5 beschrieben wurden. Darüber hinaus wurde das öffentlich verfügbare Lasertag Spiel-System der MARS-Gruppe als Grundlage verwendet. Es wurde ein Fork der Codebasis erzeugt und diese als Startpunkt in der Konzeption verwendet. In diesem Prozess wurden letztlich jedoch fast sämtliche Aspekte dieses Ausgangspunkts verworfen. Allerdings diente diese Codebasis dazu, Konzepte des MARS-Frameworks offenzulegen und diese für die Entwicklung des Systemdesigns zu verwenden. Letztlich wurden die betrachteten Systeme als Inspiration bei gewissen Designentscheidungen genutzt, während das Wissen über den Handballsport und die vorliegenden Anforderungen die treibende Kraft in der Konzeption des Systems darstellten.

4.1 Modellaufbau

Für den Aufbau des Systems wurde Handball in seine relevanten Aspekte aufgeteilt und erkannt, dass das Spielfeld als zentraler Punkt existiert. Das Spielfeld ist der Ort, auf dem alle relevanten Aktionen des Spiels ausgeführt werden. Es besteht aus verschiedenen Bereichen, die in Kapitel 2.4.1 näher beschrieben wurden. Sie spielen eine essenzielle Rolle in der Bewertung von bestimmten Situationen. Dies spiegelt sich in dem fachlichen Klassenmodell in

Abbildung 4 wieder, wo das Spielfeld als zentrales Element dargestellt ist, das aus den Bereichen besteht. Bei der Einteilung der Bereiche wurde sich für eine sehr detaillierte Unterteilung entschieden, sodass beispielsweise jede Art von Linien auf dem Spielfeld als eigene Art modelliert wurde. Zudem werden die Bereiche des Spielfelds in Bezug auf die Ovale, in denen sie liegen, benannt. Dies spiegelt auch die gängigen Redensformen im Handball wider, wonach beispielsweise der Bereich zwischen 6-Meter Linie und 9-Meter Linie als 9-Meter Kreis bezeichnet wird. Zudem ist diese Unterscheidung sinnvoll, da bei der Umsetzung von Regeln diese speziellen Bereiche besondere Behandlungen innehaben können. Dies wurde in Kapitel 2.4.1 bereits detailliert beschrieben.

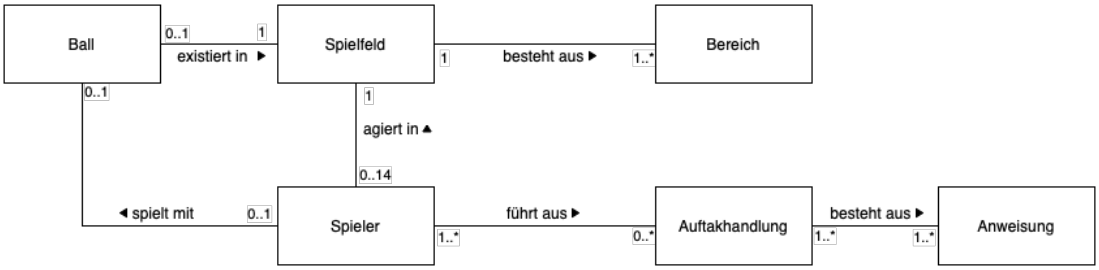

Abbildung 4: Das fachliche Klassenmodell für Handball.

Wie in dem fachlichen Klassenmodell in Abbildung 4 erkennbar, befinden sich die Spieler auf dem Spielfeld und agieren darin, um ihre persönlichen und gemeinsamen Ziele zu erreichen. Der Ball existiert ebenfalls in beziehungsweise auf dem Spielfeld, in dem er allerdings nicht von allein agieren kann. Die Spieler spielen den Wettkampf mit dem Ball, sodass sie mit diesem interagieren, ihn werfen, fangen und bei sich führen können. Als relevanter Aspekt dieser Arbeit sind zudem die Auftakthandlungen in das fachliche Klassenmodell integriert. Diese bestehen jeweils aus einer Menge von Anweisungen für bestimmte Spieler. Die Spieler kennen die Anweisungen und führen diese innerhalb eines Angriffs aus, um sich Torchancen herauszuspielen.

Umgebung

Wie in Kapitel 2.2 beschrieben, sind im Mittelpunkt eines auf dem MARS-Framework aufbauenden Systems die Layers. Diese dienen zur Beschreibung der Simulationsumgebung und sie enthalten die aktiven und passiven Teile der Simulation. Den aktiven Agenten (Akteuren)

bieten sie die Möglichkeit, mit deren Umgebung zu interagieren sowie Zugang zu Daten und Informationen zu erhalten. Die Interaktionen über angebotene Funktionen bestimmter Layers ermöglicht einen Systemaufbau, in dem die Elemente voneinander gekoppelt sind, da sie sich nicht spezifisch gegenseitig kennen, sondern nur über den entsprechenden Layer Zugriff haben. Diese Struktur unterstützt auf natürliche Art die Erfüllung der Anforderung REQ-9.

In dem entworfenen System sind daher mehrere Layers vorgesehen. Im Mittelpunkt steht dabei der `PitchLayer`, welche das Spielfeld darstellt. Diese enthält alle zentralen Elemente der Simulation, wie die MARS-Environments für die Umsetzung von Bewegungen der Spieler. Nähere Informationen zu den Bewegungsmodellen werden in Kapitel 4.2 beschrieben. Der `PitchLayer` enthält zudem die Karte des Spielfelds mit den verschiedenen Arten der Bereich, welche ein wichtiger Teil in der Bewertung von Spielsituationen sind. Um zwischen den Simulationsschritten Logik wie beispielsweise das Zustellen von Nachrichten vor dem nächsten Tick ausführen zu können, implementiert der `PitchLayer` das MARS-Interface `ISteppedLayer`. Dadurch wird der Zugriff auf die Methoden `PreTick` und `PostTick` erlangt.

Für die Bewertung der Spielsituation hinsichtlich der Einhaltung des Regelwerks dient die Klasse `RefereeLayer`. Diese befindet sich im selben Package wie die `PitchLayer`-Klasse und greift direkt auf diese zu, um die Informationen über den Zustand der Umgebung zu erhalten und diese auszuwerten. Darüber hinaus kann sie die Spieler und den Ball über angebotene Methoden auf dem Spielfeld platzieren, wenn beispielsweise nach einem erzielten Tor eine Spielsituation beendet wurde und eine folgende hergestellt werden soll. Bei der Umsetzung des Regelwerks wurde sich primär auf die Regeln in Zusammenhang mit Bereichen des Spielfelds sowie Fouls durch Kontakte zwischen den Spielern fokussiert. Regeln wie die Begrenzung der Schrittzahl der Spieler und das Prellen des Balls wurden zwar in Betracht gezogen. Jedoch wurde befunden, dass diese für die gewünschte Nutzung des Systems durch Handballtrainer keine große Relevanz haben und eine Implementierung nicht den nötigen Mehrwert bieten würde. Diese Abwägung der umzusetzenden Regeln findet in Anbetracht der Anforderung REQ-1 statt. Wie beschrieben ist das Regelwerk des Sports umzusetzen, aber dabei im Einzelfall zu prüfen, ob eine Regel sinnvoll im Rahmen des Systems ist. Bei einer Erweiterung des Systems auf gesamte Spiele könnten diese Regeln jedoch eine sinnvolle Ergänzung sein. Eine

weitere Designentscheidung war, die Bewertung der Spielsituation hinsichtlich des Regelwerks von der `PitchLayer` zu trennen. Der Grund hierfür ist einerseits, dass durch diese Auftrennung auch die Komplexität geteilt wird und der `PitchLayer` lediglich für die Simulation der Umgebung zuständig ist. Dies entspricht dem *Separation of Concerns*-Konzept. Dadurch wäre es einfach möglich, den Schiedsrichter in Zukunft als einen Agenten umzusetzen, der sich ebenfalls in dem Spielfeld bewegt und zum Treffen von Entscheidungen alle relevanten Informationen wahrnehmen muss. Eine solche Erweiterung in zukünftigen Versionen wird durch die Trennung von `PitchLayer` und `RefereeLayer` erleichtert, da letztere durch einen Agenten ersetzt werden könnte.

Ein weiterer Layer des Systems ist der `PlayerMindLayer`, welche dafür zuständig ist, die `PlayerMind`-Agenten in der Simulation zu erzeugen und initialisieren. Zudem können diese Agenten über den Layer auf gewisse Daten des `PitchLayer` zuzugreifen, wie den aktuellen Tick und die Spielfelddimensionen (Höhe und Breite) im zweidimensionalen Raum. Wird nachfolgend in der Arbeit von Agenten gesprochen, so sind diese `PlayerMind`-Agenten gemeint.

Darüber hinaus existieren zwei weitere Layers, die lediglich für das Abfragen von Informationen dienen. Der `PlaysLayer` enthält alle von einem Nutzer per Datei eingegeben Auftakthandlungen, auf welche die Agenten zugreifen können. Der `SystemSettingsLayer` hält Informationen über konkrete Konfigurationen des Systems, wie beispielsweise die maximale Anzahl an Nachrichten, die ein Agent je Tick empfangen kann. Durch diese wird das Programm hinsichtlich der hier enthaltenen Parameter leichter anpassbar, da diese an einem zentralen Punkt für das ganze System angepasst werden können. Zudem wäre es leicht möglich, diese über eine Datei anpassbar zu machen, auch wenn dies aktuell nicht umgesetzt wurde.

Abbildung 5 bietet eine Darstellung des Systems als Klassendiagramm, um einen Überblick über die Beschreibung des Modells in diesem und den folgenden Abschnitten zu erhalten. Dabei handelt es sich um einen Ausschnitt des gesamten Systems, wobei sich auf die grundlegenden Teile beschränkt wurde, wie sie in dem fachlichen Klassenmodell in Abbildung 4 aufgeführt waren und in diesem Kapitel beschrieben sind.

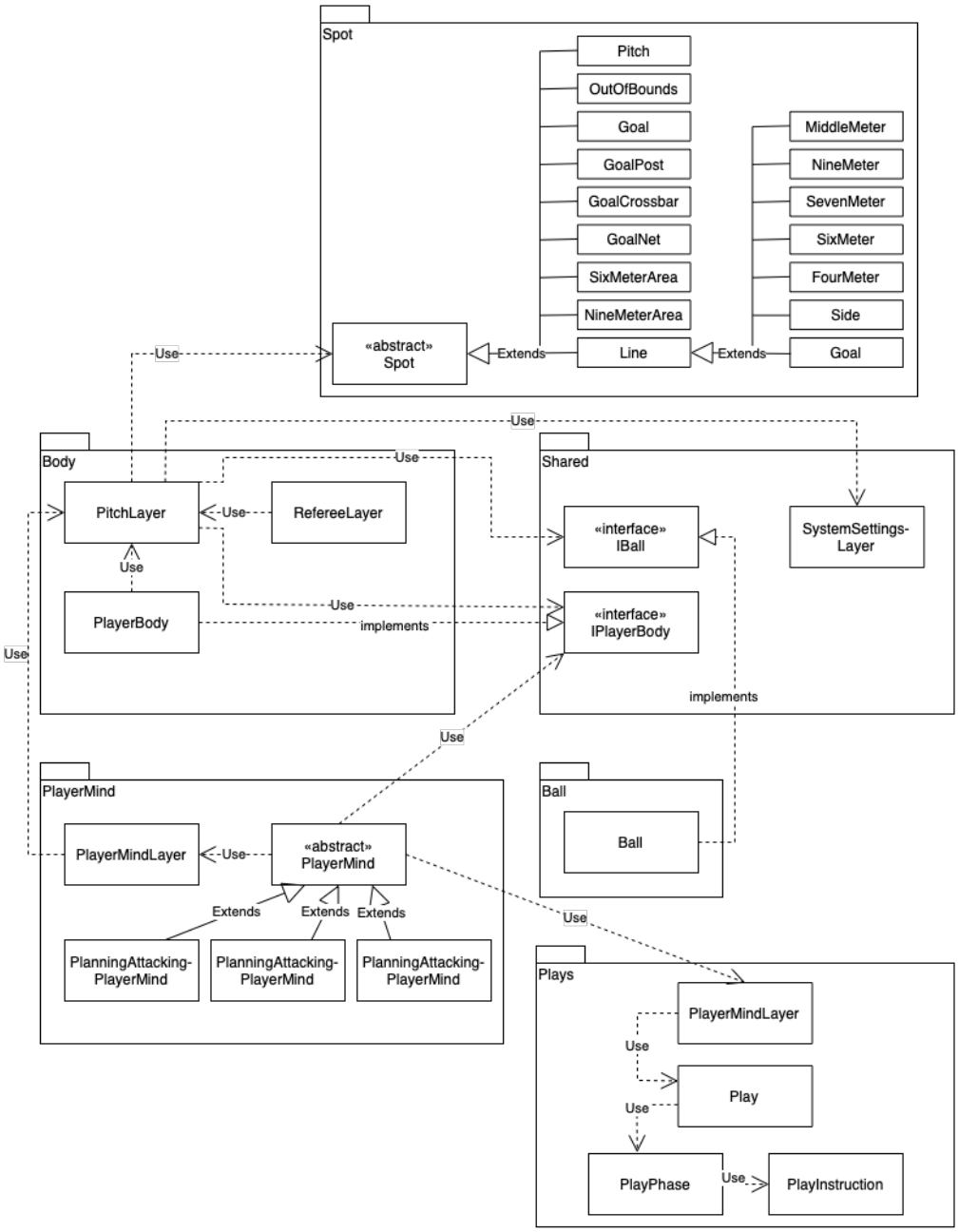

Abbildung 5: Ausschnitt des Aufbaus des Systems als Klassendiagramm.

Akteure

Die Akteure (Agenten) sind die aktiven Elemente der Simulation, sprich solche, die aktives Verhalten zeigen und eigenständig mit der Umgebung und anderen Elementen der Simulation

interagieren. Um das aktive Verhalten umzusetzen, werden die Agenten in einer MARS-Simulation in jedem Simulationsschritt einmal „angetickt“, um dann ihre interne Logik auszuführen. Agenten müssen in MARS das `IAgent`-Interface implementieren, welches einfordert, dass eine `Tick`-Methode implementiert wird. Diese Methode ist es, die in jedem Simulationsschritt durch das System aufgerufen wird und in der die Logik des Agenten umzusetzen ist.

In dem entworfenen System sind die Handballspieler als die Akteure modelliert, da diese aktiv in das Spiel eingreifen und miteinander sowie mit dem Ball interagieren. Konkreter sind die Spieler in zwei Komponenten aufgetrennt, wie es in dem MARS Lasertag Spiel und auch dem RoboCup Soccer Server der Fall ist. Sie bestehen, wie in Abbildung 5 zu sehen ist, aus einem Körper (dargestellt durch die Klasse `PlayerBody`) und einem Gehirn (konkrete Realisierungen der abstrakten Klasse `PlayerMind`). Der Körper ist die physische Repräsentation des Spielers, der sich tatsächlich auf dem Spielfeld befindet und sich dort bewegt und interagiert. Die `PlayerBody`-Klasse ist zwar auch als Agent umgesetzt, implementiert allerdings kein eigenständiges Verhalten (verfügt über eine leere `Tick`-Methode). Vielmehr ist die Implementierung des `IAgent`-Interfaces hier notwendig, damit die Körper durch den `PitchLayer` erzeugt und in den MARS-Environments bewegt werden können. Ein `PlayerMind`-Agent hingegen ist für das Verhalten des Spielers zuständig und existiert physisch betrachtet nicht in der Umgebung. Ihm ist ein `PlayerBody`-Objekt zugeordnet, welches es jedoch nur über das Interface `IPlayerBody` kennt. Die Verwendung des Interfaces statt der konkreten Klasse bietet hier einerseits eine gute Austauschbarkeit der Implementierung des Körpers und kann andererseits verwendet werden, um Funktionen zu verstecken, welche die Klasse `PlayerBody` implementiert, jedoch nicht in der Klasse `PlayerMind` ausführbar sein sollen.

Ein `PlayerMind`-Agent erhält Informationen über seine Umgebung sowie über die Sensorik des Körpers (siehe Kapitel 4.3.1) und interagiert mit der Umgebung über eine Menge von Funktionen, die das Interface `IPlayerBody` ihm bietet. Abbildung 6 stellt das vollständige Klassendiagramm des `IPlayerBody`-Interfaces dar und listet somit sämtliche verfügbaren Aktionen auf.

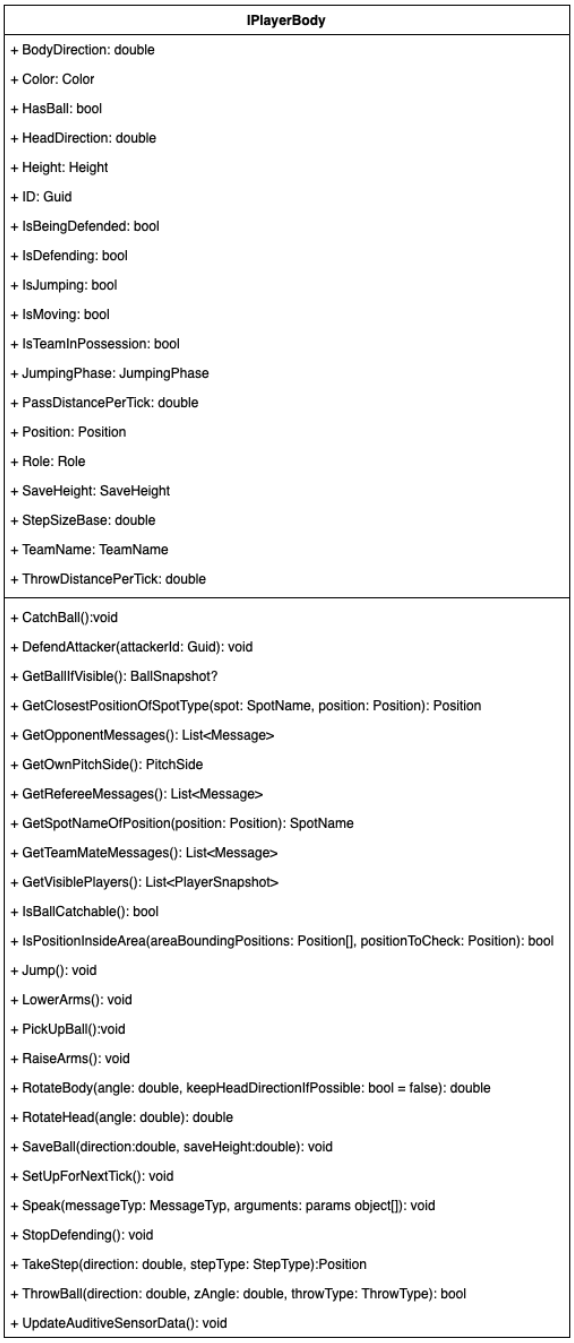

Abbildung 6: Das Klassendiagramm des `IPlayerBody`-Interface, welches durch die Klassen `PlayerMind`, `PitchLayer` und `RefereeLayer` verwendet wird.

Bei den `PlayerMind`-Agenten gibt es eine Unterteilung in drei Gruppen: Torhüter, Angreifer und Verteidiger. Das Ziel der Aufteilung ist es, eine Verringerung der Komplexität zu

erreichen, indem spezialisierte Agenten für die unterschiedlichen Spielertypen existieren. Diese können gezielt die unterschiedlichen Verhaltensweisen umsetzen. Zudem wird so ebenfalls eine besseren Wartbarkeit und Austauschbarkeit erreicht. Als Basis für alle drei Typen ist eine abstrakte Klasse `AbstractPlayerMind` vorgesehen, welche gemeinsame Funktionen bündelt.

Entitäten

Entitäten sind Elemente in einer Simulation, die nicht selbst aktiv agieren. Das heißt, dass deren Zustand sich ausschließlich dann verändern kann, wenn ein Akteur mit ihnen interagiert. Dementsprechend wird der Ball als eine solche Entität modelliert. Er existiert in der Umgebung und die Spieler können mit ihm interagieren, er hat jedoch kein eigenes Verhalten und daher auch keine `Tick`-Methode. In einem MAS verfügt eine solche Entität dafür normalerweise über eine `Update`-Funktion, welche durch die Umgebung aufgerufen wird und welche die Logik enthält, um die Entität basierend auf den erfahrenen Interaktionen zu aktualisieren. In der `Ball`-Klasse ist für diese Implementierungsmuster stattdessen die `Move`-Methode vorgesehen. Sie entspricht der `Update`-Methode und wird von der `PitchLayer`-Klasse jeweils nach einem Simulationsschritt aufgerufen und setzt dann seinerseits die Bewegung des Balls um.

Ähnlich wie bei `PlayerBody` ist für den Ball ein `IBall`-Interface vorgesehen, über welches die Agenten mit dem Ball interagieren.

4.2 Bewegungsmodelle in dem System

Für die Umsetzung der Bewegungen stehen in MARS die Environments zur Verfügung. Es gibt mehrere Arten der Environments, für die Bewegungen der Handballsimulation wurde das `CollisionEnvironment` gewählt. Dies hat zwei essenzielle Gründe. Erstens bietet das `CollisionEnvironment` die Möglichkeit, Bewegungen über Vektoren zu realisieren, indem der `Move`-Methode als Argumente eine Stärke der Bewegung (Vektorlänge) und eine relative Richtung zwischen 0° und 360° (Vektorrichtung) übergeben wird. Dies stellt die ideale Grundlage dafür dar, für die Agenten unterschiedliche Schrittarten und -längen umzusetzen und dabei mit realitätsnahen Werten arbeiten zu können. Selbes gilt für die Bewegungsgeschwindigkeit des Balls. Zweitens integriert das `CollisionEnvironment` die Berechnung

von Kollisionen zwischen relevanten Elementen der Umgebung als Teil der Bewegungsberechnung. Dies bietet eine gute Grundlage für die vorgesehene Umsetzung des Handballregelwerks, da durch die Identifikation von Kollisionen Fouls umgesetzt werden können. Zudem ermöglicht diese Funktion, ein realitätsnahes Bewegungsmodell für den Ball inklusive Abprallen und folgender Fortbewegung. Somit unterstützt diese Designentscheidung die Erfüllung der Anforderungen REQ-1 und REQ-2. Darüber hinaus erhalten die Objekte, die in dem `CollisionEnvironment` existieren, ein Feld `Extent`. Dieser drückt aus, welcher Bereich um die Position herum sie einnehmen. So können die Spieler beispielsweise einen größeren Raum ausfüllen als der Ball.

Da das `CollisionEnvironment` zur Umsetzung von Videospielen gedacht ist, werden hier nicht Agenten und Entitäten, sondern stattdessen Charaktere (Characters) und Hindernisse (Obstacles) betrachtet. Charaktere sind dabei alle beweglichen Elemente der modellierten Welt und müssen das `ICharacter`-Interface implementieren. Hindernisse sind dagegen unbewegliche Gegenstände. Sie müssen das `IObstacle`-Interface implementieren. Beide dieser Interfaces erfordern die Umsetzung einiger Methoden, am relevantesten ist dabei die `HandleCollision`-Methode. Diese wird durch das `CollisionEnvironment` aufgerufen, wenn ein anderer Charakter während seiner Bewegung mit dem jeweiligen Charakter oder Hindernis kollidiert. Sie gibt eine Ausprägung des Enum-Typs `CollisionType` zurück, welche entweder `Pass`, `Block` oder `Remove` sein kann. Diese drücken aus, was das Resultat der Kollision ist und wie das `CollisionEnvironment` darauf reagieren soll.

Da sowohl die Spieler als auch der Ball bewegliche Elemente des Systems sind, sind diese beiden als Charaktere zu betrachten, auch wenn der Ball kein Agent mit eigenem Verhalten ist. Als Hindernisse werden zudem die beiden Tore modelliert, also die drei `Spot`-Arten `GoalCrossbar` (Torlatte), `GoalPost` (Torpfosten) und `GoalNet` (Tornetz).

Bewegungen in MARS sind bei allen umgesetzten Environment-Arten jeweils auf den zweidimensionalen Raum begrenzt, sodass die Positionen der Charaktere und Hindernissen jeweils durch einen X-Wert (Breitengrad) und einen Y-Wert (Längengrad) beschrieben werden. Entsprechend der Anforderung REQ-7 soll in der entwickelten Handball-Simulation jedoch auch die Höhe umgesetzt werden. Da eine vollständige Erweiterung des MARS-Frameworks um Dreidimensionalität den Umfang dieser Arbeit übersteigen würde, wurde sich für eine Lösung

durch 2.5D entschieden. Im Rahmen dieser ist für die Agenten, den Ball und die Tore ein weiteres Attribut `Height` vorgesehen. Dieses ist beim Ball und den Spielern variabel, während es für die Tore statisch ist.

Basierend auf dieser statischen Höhe der Tore muss in deren `HandleCollision`-Methode jeweils geprüft werden, ob in Kombination mit der Höhe des Balls oder des Spielers tatsächlich eine Kollision zustande kommt. In diesem Fall, wird ein Block ausgelöst und der Ball prallt ab, ansonsten tritt tatsächlich keine Kollision auf und der Charakter darf passieren.

4.2.1 Bewegung der Spieler

Um der Anforderung REQ-2 gerecht zu werden, sollen für eine möglichst reale Bewegung der Agenten, unterschiedliche Schrittarten für die Spieler umgesetzt werden. Die Schrittarten werden in drei Kategorien modelliert: vorwärts, seitwärts und rückwärts. Zudem werden je Schrittrichtung unterschiedliche Schrittgrößen zur Verfügung gestellt. Diese abstrakten Schrittgrößen werden verwendet, um die möglichen Distanzen, die ein Agent in einem Tick zurücklegen kann, auf einen bestimmten Rahmen festzulegen und so unrealistische Bewegungen zu verhindern. Die Schrittgrößen stellen Multiplikatoren dar, die gemeinsam mit der Körpergröße des jeweiligen Spieleragenten und einem beigefügten Rauschen für die Berechnung der tatsächlichen Distanz eines Schritts verwendet werden. Ein `PlayerMind`-Agent kann Schritte über die vom `IPlayerBody` zur Verfügung gestellte Methode `TakeStep` ausführen. Dieser wird neben der Schrittart auch die absolute Richtung, in die der Schritt gemacht werden soll, als Argument übergeben.

Dabei ist es notwendig, dass die Kombination aus übergebener Richtung, Schrittart und der Richtung, in die der Körper ausgerichtet ist, übereinstimmen. Hierfür wird die relative Richtung ($|\text{absolute Richtung} - \text{Körperrichtung}|$) ausgewertet. Für einen Schritt vorwärts muss dieser Wert zwischen 315 und 45 Grad liegen, für einen Schritt rückwärts zwischen 135 und 225 Grad und für Schritte seitwärts entsprechend innerhalb der verbleibenden Bereiche. Die Methode gibt als Rückgabewert die Position nach der Bewegung zurück. Wird eine unpassende Kombination übergeben, ist der Rückgabewert stattdessen die aktuelle Position des Agenten es wird keine Bewegung durchgeführt.

Den Torhütern steht darüber hinaus die spezielle Bewegungs-Methode `SaveBall` zur Verfügung. Deren Argument ist ähnlich wie bei `TakeStep` die absolute Richtung, in die eine Torwartbewegung ausgeführt werden soll, um den Ball zu halten. Die Schrittart stellt hier allerdings keinen Parameter der Funktion dar, sondern wird aus der übergebenen Richtung abgeleitet. Dabei sind jedoch nur Schritte zur Seite oder vorwärts vorgesehen, da das Halten nach hinten im Handball in der Regel nicht vorkommt. Als zweites Argument erhält die Methode stattdessen eine Ausprägung des Enum-Typs `SaveHeight`, welche ausdrückt, welches Drittel des Tors in der Höhe gezielt abgedeckt werden soll. Eine solche Torwartbewegung hält mehrere Simulationsschritte an, in denen der Torhüter den Ball in dem ausgewählten Drittel des Tors halten kann. Die Methode erhöht zudem temporär den `Extent` des Torhüters, um darzustellen, dass eine Torwartbewegung im realen Handball in der Regel dazu führt, dass der Torwart einen größeren Bereich mit seinem Körper abdeckt. Tritt während der Torwartbewegung eine Kollision durch den Ball mit Torhüter auf, wird anhand der `Height` des Balls und der gegebenen `SaveHeight` geprüft, ob der Ball gehalten wird. Wenn ja, wird der Ball geblockt, ansonsten darf er passieren.

Um sich für die gewünschten Schritte ausrichten zu können, bietet die `PlayerBody`-Klasse eine `RotateBody`-Methode. Als weiteren Aspekt der Bewegung in dem 2.5-dimensionalen Raum steht zudem die Methode `Jump` zur Verfügung, welche einen Sprung des Agenten simuliert, um beispielsweise einen Torwurf über die Abwehr hinweg durchzuführen oder in den Torraum hineinzuspringen. Sprünge sind dabei in fünf Phasen (Enum-Typ `JumpingPhase`) umgesetzt: *kein Sprung* (`NotJumping`), *Aufstieg Mitte* (`AscentionMiddle`), *Aufstieg Hoch 1* (`AscentionHigh1`), *Aufstieg Hoch 2* (`AscentionHigh2`) und *Abstieg* (`DescentionMiddle`). `AscentionHigh1` und `AscentionHigh2` stellen dabei dar, dass der Agent in zwei Ticks auf der höchsten Höhe bleibt. Diese Phasen werden nach Ausführung der `Jump`-Methode in jedem Simulationsschritt um eine Phase inkrementiert, bis die Phase `NotJumping` wieder erreicht ist. Anders als bei den Schritten stellen die Sprunghöhen dabei jeweils einen Zentimeter Wert dar, der auf die Körpergröße hinzuaddiert wird. Die Höhe spiegelt dabei den höchsten Punkt des Körpers dar.

4.2.2 Bewegung des Balls

Da der Ball eine Entität ist, hat dieser, wie zuvor beschrieben, kein eigenständiges Verhalten. Die Bewegung des Balls entsteht daher ausschließlich durch die Interaktionen der Spieler mit ihm. Zu diesem Zweck bietet das `IBall`-Interface, welches den `PlayerBody`-Objekten für Interaktionen zur Verfügung steht, die Funktionen, um den Ball zu werfen, zu fangen, aufzunehmen oder abzublocken. Um den Ball zu aktualisieren und Bewegungen durchzuführen, wurde die `Move`-Methode modelliert. Diese Methode soll durch den `PitchLayer` jeweils nach dem Simulationsschritt aufgerufen werden. Der Grund hierfür ist, dass so umgesetzt werden soll, dass der Ball jeweils nicht mehr als einmal je Tick geworfen werden kann. Zudem wird auf diese Art modelliert, dass der Ball eine gewisse Zeit vom Punkt des Wurfs zu seinem Ziel benötigt. Darüber hinaus erhalten die Agenten so unter Umständen die Möglichkeit, bereits in dem Tick eines Wurfes auf diesen zu reagieren, sollte das Scheduling des MARS-Frameworks dafür sorgen, dass der Wurf stattfindet, bevor ein beliebiger anderer Agent „angetickt“ wird und sein Verhalten ausführt. Dies erzeugt eine Ebene der Dynamik in der Simulation, da basierend auf der Zufälligkeit des Schedulings der Zeitpunkt der Reaktionen der Agenten auf einen Wurf beeinflusst wird.

Aus diesem Grund hat die `Throw`-Funktion die Aufgabe, dem Ball die Parameter zu übergeben, anhand derer die Bewegung berechnet wird. Hierfür sind die Methodenattribute vorgesehen, die neben der absoluten Richtung und Stärke des Wurfs auch die Starthöhe und der Austrittswinkel umfassen. Die ersten beiden Attribute dienen der Berechnung der Bewegung im zweidimensionalen Raum, wie das `CollisionEnvironment` es anbietet. Die letzten beiden Attribute sind hingegen für die Berechnung der Höhe des Balls zu einem jeweiligen Zeitpunkt vorgesehen, um die 2.5-Dimensionalität umsetzen zu können. Für die spezifischere Behandlung der Beschleunigung des Balls ist zudem die interne Methode `Accelerate` vorgesehen. Die Methoden `Catch` und `PickUp` dienen jeweils dazu, in Ballbesitz zu gelangen, wobei `Catch` zum Annehmen des Balls in der Luft genutzt wird und `PickUp`, wenn der Ball nah über dem Boden ist (also eine Höhe von weniger als 50 Zentimetern hat). Für beide Methoden wird das `IPlayerBody`-Objekt übergeben, welches den Spieler repräsentiert, der die Methode aufruft. Intern sollen die beiden Methoden Prüfungen durchführen, ob sie von dem Spieler ausgeführt werden dürfen, um dann die interne Methode `PossessBall` zu verwenden, welche

die Ballannahme umsetzt. So kann der Teil der Logik, der bei beiden Methoden identisch ist, an einer Stelle gebündelt werden. Dies vermeidet Redundanzen und vereinfacht so die Wartbarkeit, Austauschbarkeit und Erweiterbarkeit. Die `Block`-Methode erhält als Attribut ein Objekt vom Typ `CollisionObjectSnapshot`, welches Information darüber hält, ob der Blocker ein Spieler oder ein Teil des Tors ist. Im Falle eines Blocks durch einen Spieler wird vermerkt, welcher Spieler als letztes den Ball berührt hat, um die Auswertung der Spielsituationen durch den Schiedsrichter zu unterstützen.

Für die Modellierung der Bewegung des Balls bietet wie beschrieben das `CollisionEnvironment` die Grundlage, um sowohl die Bewegungen auszuführen als auch die Kollisionen zu entdecken. Allerdings ist in der Bewegung durch das `CollisionEnvironment` nicht vorgesehen, dass die `ICharacter`-Objekte nach einer Kollision abprallen und sich weiterbewegen. Da dies jedoch ein essenzieller Teil der Bewegung eines Balls ist, ist es an dieser Stelle vorgesehen, diesen Fall zu behandeln. Das vom MARS-Framework gebotene Verhalten wird dabei nicht erweitert, sondern der `CollisionTypeBlock` genutzt und die Methode `Block` auf dem `Ball`-Objekt aufgerufen. Wenn diese Methode ausgeführt wird, wird dem `Ball`-Objekt signalisiert, dass gegebenenfalls eine weitere Bewegung notwendig ist und kann die resultierenden Berechnungen und folgende Bewegungen durchführen. Die Umsetzung auf diese Art beinhaltet die Behandlung des Randfalls, dass innerhalb eines Simulationsschritts mehr als ein Abprallen auftritt, da auf diese Weise beliebig viele Blocks nacheinander behandelt werden können.

4.3 Spezifikation den Agenten

Nachdem zuvor die Grundlagen für die Systemarchitektur sowie für die Bewegungen der Agenten beschrieben wurden, ist es nun notwendig, näher zu erklären, welche anderen Aspekte als Basis für das Verhalten der Agenten modelliert wurden.

4.3.1 Sensorik der Agenten

Die Sensorik ist ein wichtiger Teil eines Multiagentensystems, da diese den Agenten die Möglichkeit gibt, die Umgebung wahrzunehmen. Nur so wird adaptives Verhalten möglich, wenn die Agenten auf das Wahrgenommene reagieren können.

Aus diesem Grund wurden für das entworfene MAS zwei Arten der Sensorik vorgesehen: die visuelle Sensorik und die aurale Sensorik. Im Vergleich zu dem RoboCup Soccer Server System wurde auf eine physische Sensorik (Körpersensorik) verzichtet. Der Grund dafür ist, dass eine Umsetzung von Attributen wie Ausdauer für die Simulation von einzelnen Spielsituationen nicht als notwendig erachtet wurde. Allerdings könnten die Attribute, die aussagen, ob der Agent in Ballbesitz ist, ob er verteidigt wird oder selbst einen Gegenspieler verteidigt, sowie die eigene Höhe als Aspekte solch einer Sensorik erachtet werden. Sollte das System in Zukunft auf die Simulation ganzer Spiele erweitert werden, wäre auch eine Ergänzung einer expliziten Körpersensorik sinnvoll.

Die aurale Sensorik bietet den Agenten die Möglichkeit, miteinander zu kommunizieren. Hierfür existiert eine `Speak`-Methode, welche eine Broadcast-Nachricht über den `PitchLayer` an alle anderen Spieler versendet. Es wurde sich für Broadcasting entschieden, da laute Kommunikation in einem Handballspiel meistens Rufe sind, die alle Spieler vernehmen können. Persönliche Absprachen zwischen zwei Spielern auf kurzem Raum wurden hier außer Acht gelassen. Eine Nachricht (`Message`) besteht dabei jeweils aus einem Nachrichten-Typ (`MessageType`), der angibt, was für Informationen die Nachricht enthält, der `Id` des Absenders und einer Liste von Objekten, die den Inhalt der Nachricht ausmacht. Diese ist als Liste von Objekten modelliert, da auf diese Weise bei den verschiedenen Nachrichtentypen unterschiedliche Inhalte umgesetzt werden können. Nähere Informationen zu diesen Protokollen werden in Kapitel 4.3.2 beschrieben. Ähnlich wie im RoboCup Soccer Server wurde sich auch hier entschieden, die Zustellung der Nachrichten in drei Gruppen einzuteilen: die Nachrichten der Spieler des eigenen Teams, die der Spieler der gegnerischen Mannschaft und die des Schiedsrichters. Dadurch wird verhindert, dass die Spieler einer Mannschaft die Gegenspieler mit Nachrichten fluten und so die Kommunikation unterbinden können. Zudem wird die Zustellungsrate je Tick begrenzt, um zu modellieren, dass menschliche Spieler nicht unbegrenzt gehörte Aussagen verarbeiten können. Dies macht es notwendig, dass die Agenten für ihr Verhalten genau bewerten, ob sie eine Nachricht schicken sollten, damit die wirklich relevanten Informationen eine höhere Wahrscheinlichkeit haben zugestellt zu werden. Die Zustellung der Nachrichten findet jeweils in dem folgenden Simulationsschritt, nachdem sie abgesetzt wurden, statt. Der Versand sowie die Zustellung findet über den `PitchLayer` statt, welche die Nachrichten eines Ticks jeweils nach den drei Gruppen speichert und die Zustellung jeweils in

der `PreTick`-Methode für jeden Agenten durchführt. Diese führen für die Speicherung der gehörten Nachrichten ebenfalls drei Listen und bieten eine Methode `UpdateAuditiveSensorData` an. Es wurde sich hier dafür entschieden, die Zustellung nach dem *Pushing*-Konzept umzusetzen. Damit geschieht das Hören von Nachrichten einmalig je Tick geschieht und es muss sich nicht für jeden Agenten gemerkt werden, welche Nachrichten diesem genau zugestellt wurden, sollten mehrere versandt worden sein. Wenn die Agenten selbst die Nachrichten bei dem `PitchLayer` abfragen würden, wäre dieses Merken der individuell zugestellten Nachrichten notwendig, um zu verhindern, dass bei mehreren Abfragen unterschiedliche Informationen erhalten werden.

Dahingegen wurde für das Abrufen der visuellen Sensordaten der gegenteilige Ansatz gewählt. Diese können `PlayerMind`-Agenten über das `IPlayerBody`-Objekt eigenständig abfragen (*Pulling*-Effekt). Der Grund hierfür ist, dass Agenten während eines Ticks ihren Körper und Kopf drehen können und sich so ein neues Blickfeld ergibt. Daher müssen die Agenten in diesem Fall eigenständig die Sensordaten erneut abfragen. Zudem ist vorgesehen, dass jeder Spieler mindestens zu Beginn seiner `Tick`-Methode einmal seine `SetUpForTick`-Methode aufruft, in welcher Vorbereitungen für die Ausführung des Verhaltens getroffen werden, wie beispielsweise das Abrufen der visuellen Sensordaten.

Die visuellen Informationen, die ein Spieler über die anderen Spieler und den Ball erhalten kann, sind über die Klassen `PlayerSnapshot` und `BallSnapshot` beschrieben, welche in Abbildung 7 abgebildet sind. Diese Attribute stellen eine Teilmenge der Attribute der tatsächlichen `PlayerBody`- und `Ball`-Klasse dar. Die Umsetzung der Sensordaten über eine eigenständige Klasse dient neben der Einschränkung der sichtbaren Attribute auch dazu, dass `PlayerMind`-Agenten keinen direkten Zugriff auf die anderen `IPlayerBody`- und `IBall`-Objekte erhalten, auf denen sie Funktionen aufrufen könnten. So wird die durchgängige Daten-Integrität des Systems sichergestellt.

Für die visuelle Sensorik ist die Umsetzung eines Blickfelds für die Spieler vorgesehen, welches dynamisch basierend auf der Ausrichtung des Kopfes berechnet wird. Nur Elemente, die sich innerhalb dieses Blickfelds befinden, können auch tatsächlich durch die Spieler gesehen werden, sodass nur für diese die entsprechenden Snapshots übergeben werden. Für die Berechnung des Blickfelds ist ein Winkel vorgesehen. In Kombination der Ausrichtung des Kopfes

und dieses Winkels können Positionen ermittelt werden, die das trichterförmige Blickfeld eingrenzen. Dadurch erhalten die Agenten insgesamt betrachtet unvollständige Informationen über die Umgebung, da es möglich ist, dass sie andere Spieler nicht sehen. Sollten sich alle Elemente der Simulation innerhalb des Blickfelds befinden, könnten sie aber sehr wohl in einem Tick ein vollständiges Bild der Umgebung erhalten. Über die Limitierung der zugestellten Nachrichten und des Blickfelds hinaus ist jedoch keine weitere Maßnahme vorgesehen, um die Sensorinformationen zu maskieren und somit unvollständigere Information zu schaffen. Dies stellt somit einen Mittelweg zwischen den als Referenz betrachteten Systemen dar, da die Agenten nicht davon ausgehen können, mit vollständigen Informationen über ihre Umgebung zu arbeiten, was dem Modell des RoboCup Soccer Server ähnelt. Allerdings sind die Informationen, die sie über die Sensorik erhalten, sicher korrekt, was eher dem Ansatz der vollständigen Informationen entspricht, der in STS2 und RoboNBA zum Einsatz kommt.

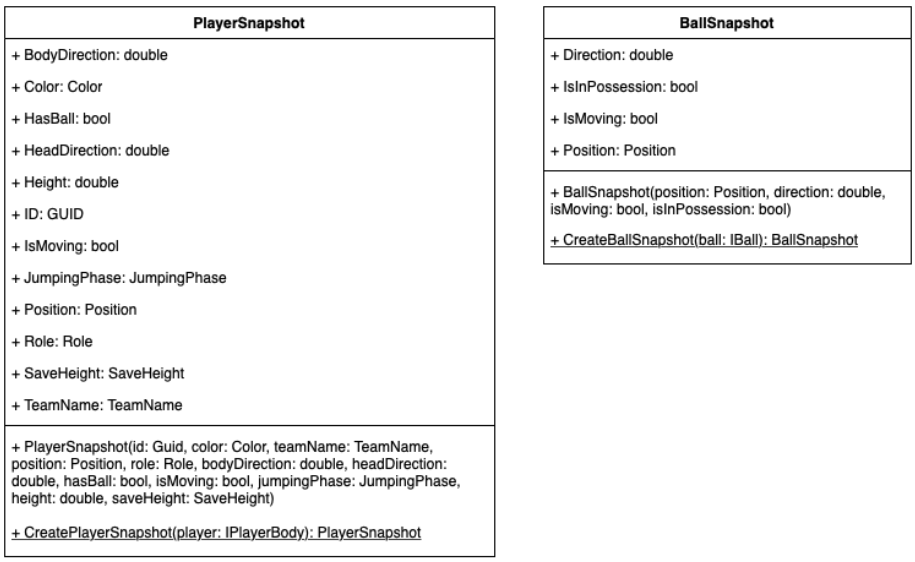

Abbildung 7: Die Klassendiagramme des strukturierten Datentypen `PlayerSnapshot` und `BallSnapshot`.

Sowohl der Winkel für die Berechnung des Blickwinkels als auch der Wert für das Zustellungslimit der Nachrichten sind als konfigurierbare Attribute vorgesehen, welche über den `SystemSettingsLayer` abgerufen werden können.

4.3.2 Kommunikationsprotokolle in dem System

Wie in Kapitel 4.3.2 beschrieben, haben die Agenten die Möglichkeit, miteinander zu kommunizieren. Eine Nachricht besteht aus der `Id` des Absenders, einem Nachrichtentyp (`MessageType`) und einer Liste von Objekten als Inhalt (`Content`). Tabelle 1 beschreibt die Kommunikationsprotokolle, indem aufgeführt wird, welcher Inhalt bei den `MessageTypes` erwartet wird.

Ein `PlayCall` dient zum Ansagen einer Auftakthandlung (`Play`) für die angreifende Mannschaft. Dementsprechend ist als Inhalt hier der Name des angesagten `Plays` als String vorgesehen. Die Namen der `Plays` sind solche, die von dem Systemnutzer für die aktuelle Simulation definiert wurden. Weitere Details hierzu sind in Kapitel 4.4.2 beschrieben).

Das `PlaySignal` dient zu genaueren Absprachen innerhalb eines Ablaufs. Um zu spezifizieren, was genau mit der Nachricht signalisiert werden soll, wird als erstes Attribut des Inhalts ein Objekt des Enum-Typs `PlaySignalType` übergeben. Die möglichen Ausprägungen dieses Enum-Typs sind `ChangePhase`, `BeginAction` und `FinishedAction`. Das zweite Attribut ist eine Zahl, die eine Phase des `Plays` repräsentiert. Im Falle des Signalisierens eines Phasenwechsels dient dies dazu, um die neue Phase mitzuteilen. Das explizite Angeben der Phase soll als Synchronisationsmechanismus dienen, da auf diese Weise ein Mitspieler, der möglicherweise eine vorherige Nachricht verpasst hat, wieder in das `Play` einsteigen kann. Ansonsten wird hier die aktuelle Phase übergeben, was ebenfalls der Synchronisation dient, da der Empfänger prüfen kann, dass beide die gleiche Phase als aktuelle Phase gespeichert haben. Für die Varianten `FinishedAction` übergibt der Absender seine eigene Rolle (Enum-Typ `Role`, siehe Tabelle 3), um dem Spielmacher mitzuteilen, wer mit seiner Anweisung fertig geworden ist. Im Fall `BeginAction` übergibt der Sender die Art der Aktion, die er ausführt.

Als weiteres Element für die Synchronisation der Agenten untereinander dient der Nachrichtentyp `Acknowledge`. Mit diesem können die Agenten ausdrücken, dass sie eine Information erhalten haben. Beispielsweise ist vorgesehen, dass der Spielmacher auf eine `FinishedAction`-Nachricht den Erhalt dieser Benachrichtigung bestätigt, oder eine Benachrichtigung über den Wechsel der Phase des `Plays` sendet. Erhält ein Agent keine solche

Bestätigung, kann er seine Information erneut senden. Der Inhalt einer `Acknowledge`-Nachricht ist der `PlaySignalType` und die `Role`, die in der erhaltenen Nachricht enthalten waren, um zu spezifizieren, welche Nachricht bestätigt wird. So ist im Falle von verlorenen Nachrichten klar, welche noch nicht bestätigt wurden.

Der `PlayerInfo`-Nachrichtentyp ist primär für die Defensive vorgesehen, sodass sich Verteidiger über Gegenspieler informieren können, die ein Mitspieler aktuell nicht sehen kann. So wird es den Verteidigern ermöglicht, trotzdem aktuelle Informationen zu erhalten und reagieren zu können.

Damit Schiedsrichter im Falle einer Unterbrechung der Spielsituation mit den Spielern kommunizieren können, existiert zudem der Nachrichtentyp `RefereeGameIntervention`. Hier kann der Schiedsrichter eine Nachricht in String-Form übergeben. Die Besonderheit bei der Kommunikation durch den Schiedsrichter (`RefereeLayer`) ist, dass diese in dem aktuellen Systementwurf über keine `Id` verfügt. Da eine `Message` diese jedoch erfordert, ist dem Schiedsrichter konzeptionell das Objekt `Guid.Empty` zugeordnet.

Tabelle 1: Die verfügbaren Nachrichtentypen und der zugehörige Inhalt.

MessageType	***Inhalt***
`PlayCall`	`[playName: string]`
`PlaySignal`	`[playSignalType: PlaySignalType, playPhase: int, roleOfSignallingPlayer?: Role]`
`PlayerInfo`	`[player: PlayerSnapshot]`
`Acknowledge`	`[playSignalType: PlaySignalType, roleOfSignallingPlayer?: Role]`
`RefereeGameIntervention`	`[messageContent: string]`

4.3.3 Gedächtnis der Agenten

Die `PlayerMind`-Agenten speichern sich eine interne Repräsentation der gesehenen Spieler und des Balls ab. Wie in Kapitel 4.3.1 beschrieben, erhalten sie in jedem Tick Informationen über sichtbaren Spieler in Form von `PlayerSnapshot`- und `BallSnapshot`-Objekten. Um sich diese Informationen zu merken und zudem abzuspeichern, wann ein bestimmtes Objekt zuletzt gesehen wurde, existieren für die interne Repräsentation die Klassen `InternalPlayerSnapshot` und `InternalBallSnapshot`.

Um eine Art Lerneffekt bei den Spielern umzusetzen, wurde für sie ein Gedächtnis über vergangene Würfe modelliert. Dieses kann von allen Spielertypen zu unterschiedlichen Zwecken verwendet werden, ist aber primär für die Angreifer und Torhüter vorgesehen. Die modellierte `ThrowMemory`-Klasse kann in einer Liste an jedem `InternalPlayerSnapshot` ergänzt werden. Auf diese Art können sich Torhüter (und wenn gewollt auch Verteidiger) für jeden Schützen merken, in welchen Situationen er wirft und wie sein Wurfbild aussieht. Die Angreifer können sich merken, welche Ecken der Torhüter am häufigsten abdeckt und unter welchen Parametern vorherige Würfe erfolgreich oder nicht erfolgreich waren. Dies spiegelt auch den realen Handball in gewisser Weise wider, wo die Spieler im Verlauf eines Spiels die Muster der Gegenspieler erkennen, um daraus Vorteile zu erlangen.

Wie dem Klassenmodell in Abbildung 8 zu entnehmen ist, verfügt die `ThrowMemorySnapshot`-Klasse über Instanzvariablen für das Abspeichern des Ticks, in dem der Wurf stattgefunden hat, der `Id` des Schützen sowie der `TeamName` der Mannschaft, zu der der Schütze gehört. Letzteres ist insbesondere mit Hinsicht auf eine mögliche Erweiterung des Systems zum Simulieren ganzer Spiele und für die Möglichkeit des Sortierens nach Mannschaften beigefügt.

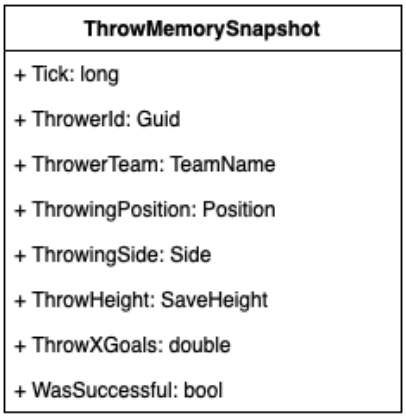

Abbildung 8: Klassendiagramm der `ThrowMemorySnapshot`-Klasse, die durch `PlayerMind`-Agenten genutzt wird, um sich Details zu vergangenen Würfen zu merken.

Darüber hinaus gibt die `ThrowingPosition` an, von welcher Position aus der Wurf ausgeführt wurde. So können Spieler beispielsweise alle gemerkten Würfe nach der Region, wo sie abgegeben wurden, filtern, um spezifischere Muster erkennen zu können. Um diese Muster noch zu erweitern, kann zudem der XGoals-Wert (siehe Kapitel 2.4.3) für den Schützen bei seinem Wurf in `ThrowXGoals` abgelegt werden. Angreifer können dies nutzen, um zu erkennen, welche XGoals-Werte als vielversprechend gelten. Verteidiger und Torhüter können darüber Würfe frühzeitig antizipieren und so ihre Chancen verbessern, ein Tor zu verhindern. Die `ThrowingSide` drückt aus, auf welche Seite des Tores geworfen wurde oder welche Richtung der Torwart abgedeckt hat. Der dafür entworfene Enum-Typ `Side` verfügt über die Ausprägungen `Left`, `Middle` und `Right`, um das Tor in Drittel einzuteilen. Diese Umsetzung entspricht der Teilung in Drittel, die auch in der Höhe des Tors zum Einsatz kommt. Auch diese kann in der Variable `ThrowHeight` abgespeichert werden.

Über das Attribut `WasSuccessful` wird ausgedrückt, ob ein Torwurf erfolgreich war. Es ist daher vorgesehen, dass die Agenten eine `ThrowMemorySnapshot`-Instanz über mehrere Ticks hinweg befüllen und mit Beendigung des Wurfprozesses persistieren. Um diese Aufgabe zu erleichtern, ist für das Werfen und Blocken des Balles geplant, dass ein Event ausgelöst wird, welches die Agenten abonnieren können, um entsprechend darauf zu reagieren. Auch hier erhalten die Agenten allerdings nicht sicher vollständige Informationen, da sie die Werte für die beschriebenen Attribute mit den ihnen vorliegenden Daten selbst berechnen müssen. So

können unterschiedliche Agenten beispielsweise verschiedene Ergebnisse für die XGoals herausbekommen, wodurch sie in der Zukunft womöglich unterschiedlich auf dieselbe Situation reagieren. Über das Event erhalten sie lediglich den aktuellen Tick, die `Id` des Werfers, die Position und die Höhe des Balls. Der Grund hierfür ist, dass Torhüter ansonsten eine viel komplexere Behandlung umsetzen müssten, um sich zu merken und zu errechnen, wo ein Ball gelandet ist, der an ihrem Halteversuch vorbeigekommen ist.

4.4 Verhalten der Agenten

Da das Verhalten der Agenten ein essenzieller Teil eines MAS ist, wird hier detailliert beschrieben, wie dies modelliert ist. Dabei kommt GOAP zum Einsatz, um das Verhalten der Agenten zu steuern. Zudem wird das erarbeitete Modell für die Umsetzung der Auftakthandlungen durch die angreifende Mannschaft beschrieben.

4.4.1 Goal-Oriented Action Planning

Um ein adaptives Verhalten der Spieler zu erreichen, wird *Goal-Oriented Action Planning (GOAP)* umgesetzt. Dabei trifft jeder Agent seine eigenen Entscheidungen, basierend auf seiner Wahrnehmung der Umgebung. Da Spieler in der Regel keine vollständigen Informationen haben, sollten sich in verschiedenen Simulationsläufen unterschiedliche Verhalten der individuellen Agenten entwickeln.

Das MARS-Framework enthält eine Umsetzung von GOAP, welche verwendet werden kann, um die planenden Agenten zu entwickeln. Dafür erhält jede Art von Agenten einen Planner der Klasse `GoapPlanner`. Diesem werden bei der der Erzeugung ein `GoapAgentStates`-Objekt übergeben, welches den internen Zustand des Agenten durch eine Menge von `GoapStateProperty`-Objekten darstellt. Zudem erhält der Planner jeweils die Aktionen als Menge von `IGoapAction`-Instanzen und die Ziele als Menge von `IGoapGoal`-Instanzen. Da es sich bei beiden um Interfaces handelt, wurden die abstrakte Klasse `HandballGoapAction` und die konkrete Klasse `HandballGoapGoal` entworfen, die für die Übergabe der entsprechenden Objekte verwendet werden können.

Die `HandballGoapAction` kennt dabei das `PlayerMind`-Objekt, in dem es verwendet wird. Dieses wird bei der jeweiligen konkreten Umsetzung einer Aktion genutzt, um die `ExectueAction`-Methode zu implementieren, innerhalb derer die zugehörige Aktion des Agenten aufgerufen wird. Nach der Erstellung eines Objektes müssen über die Methoden `AddOrUpdatePrecondition` und `AddOrUpdateEffect` die Vorbedingungen und Nachbedingungen hinzugefügt werden.

Die Objekte vom Typ `HandballGoapGoal` erfordern hingegen keine Entwicklung weiterer Klassen, denn alle notwendigen Informationen können dem Konstruktor übergeben werden. Dieser bekommt als Attribute dieselbe `GoapAgentState`-Instanz wie der Planner und ein `Predicate` übergeben. Dieses `Predicate` wird ausgeführt, wenn der Planner auf der Instanz die Methode `UpdateRelevance` aufruft. Zudem erhält das `HandballGoalGoal` zwei Werte, um auszudrücken, wie die Priorität dieses Ziels ist, wenn das `Predicate` zu wahr ausgewertet wird und eines für den Fall, dass die Auswertung falsch ergibt. Auf dieser Basis können die GOAP-Modelle für die Angreifer, Verteidiger und Torhüter beschrieben werden.

Die Aktionen werden durch Vorbedingungen und Effekte beschrieben. Die Effekte werden als „Idealfall“ betrachtet, also können diese durch die Aktion erreicht werden, es ist aber nicht sichergestellt, dass der Effekt jedes Mal erreicht wird. Dies kann an den parallelen Verhaltensweisen der anderen Akteure liegen, oder auch daran, dass zum Beispiel eine Bewegungs-Aktion mehrere Schritte lang ausgeführt muss, um zu dem gewünschten Punkt zu gelangen.

Die Modellierung der GOAP-Modelle für die Angreifer, Verteidiger und Torhüter war jeweils durch viel Testen bestimmter Kombinationen geprägt und die anfänglichen Modelle mussten im Rahmen der Implementierung sehr angepasst und erweitert werden. Daher wird die detailliertere Beschreibung der Modelle nicht an dieser Stelle vorgenommen, sondern in Kapitel 5.4.3 ausgelagert.

4.4.2 Umsetzung der Spielabläufe und Anweisungen

Ein essenzieller Aspekt des zu entwickelnden Systems sind die Auftakthandlungen, die von der angreifenden Mannschaft ausgeführt werden sollen, um sich Torchancen zu erspielen. Dies ist durch die Anforderung REQ-4 dokumentiert. Trainer sollen in der Lage sein, diese

möglichst detailliert zu definieren und so ihre Ideen genau in der Simulation umsetzen zu lassen. Um dies zu ermöglichen, sollten gängige Begrifflichkeiten des Sports verwendet werden und individuelle Anweisungen an Spieler in bestimmten Rollen gegeben werden können.

Damit dies möglich ist, soll eine Datei im JSON-Format verwendet werden, in der die Auftakthandlungen formuliert werden. Diese kann von einem Anwender einerseits direkt ausgefüllt werden, oder eine entsprechende Funktionalität in der Nutzeroberfläche entwickelt werden, um diese einfacher zu erstellen und bearbeiten.

Um diese Abläufe darzustellen, dient die `Play`-Klasse, welche einen kompletten Ablauf darstellt. Diese enthält einen Namen für die Auftakthandlung, um sie identifizieren zu können. So können mehrere Auftakthandlungen definiert werden, aus denen die Spieler auswählen können. Um möglichst granulare Anweisungen zu ermöglichen, ist jedes `Play` in mehrere Phasen unterteilt, die nacheinander durchlaufen werden. Diese sind durch die `PlayPhase` Klasse modelliert. Die Klasse beschreibt jeweils die Anweisungen innerhalb einer Phase für einen bestimmten Spieler über seine Rolle. Über ein Dictionary werden die `PlayPhase`-Objekte in der `Play`-Instanz phasenweise sortiert verwaltet. Für die Beschreibung der konkreten Anweisungen ist die `PlayInstruction`-Klasse entwickelt worden. Eine Anweisung wird durch das Attribut `InstructionType` definiert. Diese stellen gängige Anweisungen für Spieler innerhalb einer Auftakthandlung dar. Diese beinhalten unter anderem das Passen zu einem Mitspieler (`Pass`), das Einlaufen von außen (`Runner`) und das Stellen einer Sperre (`Block`). Jede Anweisung spricht jeweils einen Spieler konkret an und in der Regel funktioniert sie auch unabhängig von anderen Anweisungen. Lediglich für die Kreuzung ist es notwendig, dass zwei Spieler in derselben Phase die zusammengehörenden Anweisungen `CrossFor` und `CrossWith` übergeben bekommen, damit sie gemeinsam den Ablauf ausführen können. Neben dieser Spezifikation, was eine Anweisung aussagt, wird einer `PlayInstruction` zudem eine `FieldPosition` zugewiesen, welche ausdrückt, wo die Aktion ausgeführt werden soll. Darüber hinaus können der Anweisung weitere Informationen hinzugefügt werden, wobei unterschiedliche Details für die verschiedenen `InstructionType`-Ausprägungen notwendig sind. Diese Varianten sind als Liste von Objekten umgesetzt, ähnlich wie optionale Argumente in einer Funktion. In Tabelle 2 wird aufgeführt, was genau durch die `FieldPosition` bei

dem jeweiligen `InstructionType` ausgesagt wird und welche weiteren Details zusätzlich zu übergeben sind.

Tabelle 2: Die `InstructionType`-Ausprägungen und die zugehörigen Parameter.

InstructionType	***Aussage der FieldPosition***	***Weitere Angaben***
`Pass`	Keine Bedeutung	1. `Role` des anzuspielenden Mitspielers.
`Throw`	Keine Bedeutung	Keine weiteren Angaben
`Block`	Defensive Lücke, in der die Sperre zu setzen ist.	1. Boolescher Wert, der aussagt, ob die Sperre an dem linken Verteidiger der Lücke (von Mittellinie aus betrachtet) gesetzt werden soll.
`Runner`	Defensive Lücke, bis zu der der Einlaufende rennen soll.	Keine weiteren Angaben
`CrossFor`	Defensive Lücke, in der die Kreuzung durchgeführt werden soll.	1. `Role` des Mitspielers, mit dem gekreuzt wird, also der angespielt werden soll.
`CrossWith`	Defensive Lücke, in der die Kreuzung durchgeführt werden soll.	1. `Role` des Mitspielers, der die Kreuzung ausführt.
`MoveTo`	Die Position, zu der der Spieler sich bewegen soll.	1. `Double`-Wert, der aussagt, wie viele Meter von der Torlinie aus entfernt das Ziel des Spielers ist.

5 Realisierung des Systems

Nachdem der Modellaufbau in Kapitel 4 erklärt wurde, sind nachfolgend die Implementierungsdetails zu beschreiben. Hierbei werden insbesondere solche Aspekte dargestellt, die von der Modellierung abweichen oder bei denen es aus anderen Gründen relevant ist, genauer auf die Umsetzung einzugehen.

5.1 Wahl der Programmiersprache

Da das MARS-Framework in der Programmiersprache C# geschrieben ist, wurde diese Sprache für die Umsetzung des Systems verwendet. Für die Erstellung eines zugehörigen Visualisierungsprogramms wurde sich für eine Implementierung als *Node.JS*-System entschieden, welches das Framework *Electron.JS* einsetzt, um mit den Sprachen HTML, CSS und TypeScript eine alleinstehende Applikation zu erzeugen, ohne dabei einen Webbrowser verwenden zu müssen. Die Entscheidung, diese Sprachen zu verwenden, basiert darauf, dass sie Möglichkeiten bieten, um eine attraktive Nutzeroberfläche zu erstellen und so das System für Nutzer interessanter und einfacher nutzbar zu machen. Um die Visualisierung der Abläufe zu unterstützen, wurde zudem das Framework *Excalibur.JS* eingesetzt, bei dem es sich um eine Game-Engine handelt und die somit viele Möglichkeiten für eine flüssige und anschauliche Darstellung bietet.

5.2 Umsetzung des Analyseprogramms

Zu Beginn der Arbeit an dem System wurde das Analyseprogramm, welches für das Lasertag Spiel zur Verfügung gestellt wird, für die Betrachtung der Simulationsläufe verwendet. Dieses ist in Python unter Verwendung der *PyGame*-Bibliothek umgesetzt, also unter Einsatz einer Game-Engine, die ebenfalls das Konzept der Ticks und Akteure umsetzt. In diesem Fall wird für die Akteure jedoch kein kompliziertes Verhalten umgesetzt, sondern lediglich deren

Position aktualisiert. Dieses System liest die in der MARS-Simulation erstellten Log-Dateien sowie die CSV-Datei, die das Spielfeld beschreibt, ein. Auf diese Weise kann das Feld dargestellt und unterschiedliche Bereiche eingefärbt werden. So werden auch die Positionen der Agenten in jedem Zeitschritt in das Analyseprogramm übertragen, um visualisiert zu werden.

Im Laufe der Arbeit stellte sich jedoch heraus, dass dieses System für die Nutzung im Rahmen des entwickelten Multiagentensystems limitiert ist und dauerhaft unzureichende Performanz in Bezug auf die erreichte Geschwindigkeit (*Frames per Second*, *FPS*) zu erkennen war. Die Framerate lag dauerhaft bei weniger als 5 FPS, wodurch Dynamiken innerhalb der Simulation schwierig nachzuempfinden waren. Der Grund hierfür ist, dass innerhalb von GamePy nur Akteure eingefärbt werden können. Um die Details des Handballspielfelds adäquat umzusetzen, wurde die Karte für das Spielfeld in der Handball Simulation bedeutend größer angelegt die des Lasertag Spiels. Daher wurde die Anzahl der Akteure hier zu groß, was bedeutende Einbußen in der Animationsgeschwindigkeit zur Folge hatte.

Dies machte es also notwendig, dass das Analyseprogramm in großem Stil überarbeitet, oder komplett neu aufgesetzt werden musste. Da die Lösung mit Python und PyGame zudem Limitierungen in der Gestaltung der Nutzeroberfläche hatte, wurde sich hier für letzteres entschieden, um die Gestaltungsvorteile anderer Technologien nutzen zu können. Hier wurde sich für die Entwicklung der Nutzeroberfläche wie in Kapitel 5.1 aufgeführt, für die Verwendung HTML und CSS entschieden. Ferner wurde sich für die Verwendung von TypeScript entschieden, da diese Technologie oftmals zusammen mit HTML und CSS zum Einsatz kommt und dabei die Vorteile einer typisierten Sprache mit sich bringt. Um das System als eigenständige Applikation ausführen zu können, kommt darüber hinaus das Electron.JS-Framework zum Einsatz, welches eben dies ermöglicht. Um nicht selbst die Update-Schleife inklusive des Sicherstellens von bestimmten FPS-Raten implementieren zu müssen, wurde als Basis ebenfalls eine Game-Engine in Form von Excalibur.JS eingesetzt.

Um das auftretende Problem aus der vorherigen Version des Programms zu beheben, wird mit Starten der Applikation zunächst die Spielfeld-Datei aus dem MARS-System ausgelesen und basierend auf den CSV-Einträgen ein HTML-Canvas-Element pixelweise eingefärbt. Anschließend wird dieses Element als Bild abgespeichert, welches nachfolgend als Hintergrund des durch Excalibur.JS dargestellten Bereichs verwendet wird. Für diese Darstellung wird das

vorherige Canvas-Element wiederverwendet. So werden nicht unnötig Ressourcen für die Darstellung des Spielfelds aufgebracht, da dieses sich im Laufe einer Simulation nicht verändert und somit nicht dynamisch umgesetzt sein muss.

Um den Effekt einer flüssigen Animation der Spieler und des Balls zu erzeugen, sind die Ticks in mehrere Teilschritte aufgeteilt. Zwischen zwei Ticks werden jeweils Delta-Werte für die X- und Y-Positionen eines Akteurs berechnet, sodass die Bewegungen von der einen zur anderen Position in kleineren Schritten angezeigt werden. Dies ist eine gängige Methode, um flüssige Animationen umzusetzen. Durch diese Unterteilung ist es zudem möglich, die Kollisionen und Fortbewegungen des Balls innerhalb eines Ticks auch in der Visualisierung darzustellen und so nachvollziehbar zu machen.

Neben der Betrachtung der Simulationsergebnisse bietet das Analyseprogramm zudem Funktionen für das Anpassen bestimmter Konfigurationsdateien für den Nutzer (siehe Kapitel 5.3). So können Trainer und Trainerinnen die Ausgangspositionen der Spieler und des Balls sowie die Auftakthandlungen über eine ansprechende Nutzeroberfläche bearbeiten. Dies soll die Nutzbarkeit für Anwender und Anwenderinnen erleichtern, da sie die Dateien nicht manuell anpassen zu müssen.

5.3 Konfiguration und Ausführung des Systems

In einem auf MARS basierenden Multiagentensystem wird eine ausführbare Klasse mit `Main`-Methode verwendet, um notwendige Definitionen für die Simulation vorzunehmen und diese nachfolgend auszuführen. In dem entwickelten System findet dies innerhalb der `Program`-Klasse in der Datei *Program.cs* auf Root-Ebene statt. Ein Ausführen dieser Datei startet somit das System. In Kapitel B.1 im Anhang B kann diese Klasse genauer betrachtet werden.

Innerhalb dieser `Main`-Methode wird ein Objekt der `ModelDescription`-Klasse erzeugt. Diesem Objekt werden über entsprechende Methoden sämtliche Layer- und Agententypen, die innerhalb einer Simulation verwendet werden sollen, hinzugefügt. Für grundlegende Änderungen in der Systemkonfiguration bezüglich des eingesetzten Layer muss diese `Main`-Methode entsprechend überarbeitet werden.

Darüber hinaus sieht MARS detailliertere Konfigurationen vor, welche in Form einer JSON-Datei beschrieben werden können, welche in der `Main`-Methode eingelesen wird. Innerhalb dieser Datei werden Details spezifiziert, wie die Anzahl bestimmter Agententypen, die in der Simulation eingefügt oder die Anzahl an Ticks, die ausgeführt werden sollen. Zudem ist es hier möglich, konkrete Attribute mit Werten zu belegen, die in der Simulation entweder als Konstanten oder als Ausgangspunkte verwendet werden können. Ebenso können weitere Konfigurationsdateien für bestimmte Layer- und Agententypen angegeben werden. Diese können ebenfalls Attribute, oder auch andere Details wie die Grundrisse für Karten innerhalb der Simulation, enthalten. Die Datei heißt in dem entwickelten System *config.json* und befindet sich auch auf Root-Ebene und ist in Kapitel B.2 von Anhang B beigefügt.

Innerhalb dieser Datei wird die Anzahl der `PlayerBody`-Agenten festgelegt. Diese ist in der Regel auf 14 gesetzt, es könnte jedoch auch eine geringere Zahl gewählt werden, um beispielsweise Überzahlsituationen zu simulieren. Die Anzahl von 14 sollte allerdings nicht überschritten werden, da dies im Handball unzulässig wäre. Ebenfalls wird hier die Anzahl der Bälle mit dem Wert 1 belegt. Dieser sollte nicht verändert werden, da abweichende Werte nicht sinnvoll und unzulässig wären. Für den `PitchLayer` werden hier zudem die Werte für die die Dimension des Spielfelds in der X- und Y-Achse auf 440 respektive 240 gesetzt. Diese Werte basieren auf den Spielfelddimensionen von 40 Metern in der Länge (X-Achse) und 20 Metern in der Breite (Y-Achse). Dazu addiert werden je Seite 2 Meter, um das Aus darstellen zu können. Die Dimensionen werden dann mit der Anzahl an Spielfeldkacheln je Meter multipliziert. In diesem Fall ist eine Größe von 10 Zentimeter je Kachel vorgesehen, also 10 Kacheln pro Meter. So ergibt sich für die X-Dimension die Berechnung $(40\,\text{Meter} + (2 \times 2\,\text{Meter})) \times 10\,\text{Kacheln} = 440\,\text{Kacheln}$. Analog dazu kann die Berechnung der Y-Dimension durchgeführt werden. Diese Kachelgröße wurde gewählt, da so ein Mittelweg zwischen zu großer Anzahl an Kacheln, welche die Geschwindigkeit beeinflussen, und einer akzeptablen Granularität für die Darstellung der kreisförmigen Bereiche des Spielfelds erzielt wird.

Für weitere Konfigurationsmöglichkeiten der Agenten und Layers sind weitere Dateien vorgesehen, die nachfolgend beschrieben werden. Die jeweiligen Dateien befinden sich in dem Resources-Verzeichnis des Systems.

handball_pitch_10x10.csv

Diese Datei stellt die Karte des Spielfelds dar. Die Anzahl an Spalten und Zeilen in dieser CSV-Datei (exklusive möglicher Kopfzeilen) stimmen mit den in der *config.json*-Datei gesetzten X- und Y-Dimensionen überein. Jedes Feld in der Datei wird mit einem ganzzahligen Wert zwischen 0 und 15 ausgefüllt, welcher definiert, um was für einen Spielfeldbereich es sich an der Position handelt. Eine Belegung mit 0 definiert beispielsweise einen ganz normalen Spielfeldbereich ohne weitere Bedeutung, während eine 1 für das Aus und eine 3 für den Torraum steht, die jeweils besondere Bedeutungen für die Spieler und den Ball haben.

ball_config.csv

Diese Datei wird genutzt, um Attribute für den Ball zu definieren, die dieser zu Beginn der Simulation haben soll. Die Werte, die hier gesetzt werden können, sind *xSpawn*, *ySpawn*, *playerInPossessionRole* und *playerInPossessionColor*. Mit den ersten beiden kann der Ball an einer bestimmten Position platziert werden, mit dem zweiten Paar an Attributen wird der Ball einem bestimmten Spieler gegeben. Ist letzteres definiert, so haben die *xSpawn* und *ySpawn* Werte keine Relevanz.

player_config.csv

Diese Datei dient zur Definition bestimmter Attribute, welche die Spieler zu Beginn der Simulation innehaben. Dabei handelt es sich bei den Attributen *memberId*, *teamColor* , *teamName*, *jerseyNumber*, *role* und *bodyHeightInCentimeters* um Konstanten, die sich innerhalb der Simulation nicht ändern. Sie beschreiben, welcher Mannschaft ein Spieler angehört, welche Nummer er hat und welche Rolle er in dem Team einnimmt. Das letzte Attribut definiert die Körpergröße des Spielers in Zentimetern. Die Attribute *xSpawn*, *ySpawn* und *initialBodyDirection* stellen die Ausgangswerte für den Ort, an dem der Spieler initial platziert wird und in welche Richtung er schaut, dar und haben nach Beginn der Simulation keine weitere Bedeutung.

Durch die Anzahl der Einträge in dieser Datei kann die Anzahl an Spielern innerhalb der Simulation ebenfalls gesteuert werden, wenn weniger Einträge beschrieben werden, als `PlayerBody`-Agenten in der *config.json*-Datei definiert sind.

plays.json

Diese JSON-Datei ist für die Eingabe der Anweisungen an die Agenten vorgesehen, um Auftakthandlungen in der Simulation durchführen zu können. Die Eingaben basieren auf den in Kapitel 4.4.2 beschriebenen `Play`-Objekten, die hier in JSON-Format in der Datei eingetragen werden können. Dabei können beliebig viele Phasen je `Play` beschrieben werden, während bei mehreren Anweisungen an einen Spieler nur eine je Phase durchgeführt wird. Zudem ist es möglich, mehrere `Play`s für die Spieler zu definieren, da in der JSON-Datei ein Array aus `Play`-Objekten beschreibbar ist. Erhalten die Spieler-Agenten mehrere `Play`s, so suchen sie eines aus diesen aus, wenn sie eine Auftakthandlung ankündigen wollen. Diese Umsetzung wurde insbesondere in Hinsicht auf mögliche Erweiterbarkeit gewählt, sodass beispielsweise bei der Simulation eines ganzen Spiels in Zukunft mehrere Auftakthandlungen von einer Mannschaft genutzt werden können. Das Kapitel B.3 in Anhang B stellt eine beispielhafte Beschreibung dieser Datei dar.

system_settings.json

Um einige Werte, die innerhalb der Simulation für Berechnungen oder Limitierungen verwendet werden, leicht anpassbar zu machen, ist die Datei *system_settings.json* definiert. Diese wird von dem SystemSettingsLayer genutzt, um die Werte auszulesen und so innerhalb des Systems zur Verfügung zu stellen. In der Datei beschreiben die Attribute *TicksPerSecond*, *MessageReceivingMaximum* und *ViewAngle* zuvor beschriebene Limitierungen oder Grundlagen für Berechnungen.

Die weiteren Attribute bestimmen Modifikatoren oder Zentimeter-Werte für die Berechnung des Attributs `Height` im Falle des Springens oder gehobener Arme. Die Attribute hierfür sind: *RaisedArmsModificator*, *NotJumpingHeight*, *AscentMiddleHeight*, *AscentHigh1Height*, *AscentHigh2Height*, *DescentMiddleHeight*.

defensive_formation.csv

Diese Datei dient zur Spezifikation, in welcher Abwehrformation die Verteidiger auftreten sollen. Dies wird als Attribut des `DefensiveFormationLayer` in das System injiziert und so für die Abwehrspieler verfügbar gemacht. Gültige Werte sind zwar alle Ausprägungen des

Enum-Typs `DefensiveFormation`, allerdings ist zum Zeitpunkt dieser Arbeit nur die sogenannte „6:0"-Deckung (`SixZero`) umgesetzt, da der Fokus auf die Implementierung der Angreifer gelegt wurde.

5.4 Implementierung

In diesem Abschnitt wird auf bestimmte Aspekte der Implementierung eingegangen, insbesondere die Änderungen und Abweichungen zum Entwurf, die im Prozess der Entwicklung vorgenommen werden mussten. Zudem werden konkrete Umsetzungsdetails wie die Verhaltensweisen unter Verwendung von GOAP erläutert. Auf die Teile des Systems, die wie in dem Entwurf vorgesehen umgesetzt werden konnten, wird nicht genauer eingegangen, wenn es hierfür keinen besonderen Grund gibt.

5.4.1 Abweichungen von dem Systementwurf

Während des Entwicklungsprozesses stellte sich heraus, dass die Informationen über den Zustand des `Ball`-Objekts in jedem Tick nicht korrekt in die Log-Datei eingetragen wurden, als diese Klasse wie vorgesehen als Realisierung des `IEntity`-Interface implementiert wurde. Zwar wurde die erwartete Datei erstellt und auch die Kopfzeile wie erwartet mit den *public*-Attributen der Klasse befüllt, jedoch wurden die Zustandswerte der einzelnen Ticks nicht in der Datei eingetragen. Da das Persistieren dieser Daten bei den `PlayerBody`-Objekten, welche eine Realisierung des `IAgent`-Interface sind, korrekt funktionierte, wurde diese Problematik gelöst, indem auch die Klasse `Ball` das `IAgent`-Interface realisiert. Die einzige konkrete Veränderung, die sich hierdurch ergibt, ist, dass die `Ball`-Klasse nun ebenfalls eine `Tick`-Methode enthält, die in jedem Tick aufgerufen wird. Um den Ball jedoch weiterhin als Entität zu behandeln, ist diese Methode lediglich ein Rumpf, welche keine Logik beinhaltet. Dadurch besitzt der Ball nach wie vor kein eigenständiges Verhalten. Weiterhin wird der Ball innerhalb des Systems wie eine Entität behandelt, auf die Art, wie es im Entwurf beschrieben wurde. Dies ermöglicht es auf sehr einfache Art, die Implementierung anzupassen und die `Ball`-Klasse wie vorgesehen als Realisierung des `IEntity`-Interface umzusetzen, wenn die korrekte Funktionsweise verfügbar ist.

Ein weiteres Problem während der Implementierung des Systems trat in der Umsetzung des Bewegungsmodells auf. Während das vom MARS-Framework angebotene `CollisionEnvironment` viele Möglichkeiten bietet, um die Bewegungen in dem System wie gewünscht umzusetzen, trat während der Entwicklung ein wiederkehrender Fehler auf. Bei der Berechnung der Folgeposition eines bewegten, `ICharacter` implementierenden Objekts, welches durch eine Kollision unterwegs gestoppt wurde (durch die Rückgabe von `CollisionType.Block`), wurde regelmäßig eine falsche Position ermittelt, auf die der Agent als Resultat der geblockten Bewegung bewegt wurde. Der Grund hierfür scheint in den berechnenden Methoden innerhalb der `PositionHelper`-Klasse zu liegen. Anscheinend sind die ermittelten Parameter für die X- und Y-Werte der Position hier in Winkeln berechnet, statt dass die tatsächlichen Koordinaten ermittelt werden. Dies führte fortlaufend dazu, dass die Spieler oder der Ball nach einer Kollision plötzlich auf die andere Seite des Spielfelds befördert wurden. Dies machte es notwendig, der `TakeStep`-Methode in der `PlayerBody`-Klasse sowie der `Move`-Methode in der `Ball`-Klasse eine Behandlung dieser Problematik hinzuzufügen. So wird nach der Bewegung über die `Move`-Methode des `CollisionEnvironment` eine Prüfung durchgeführt, ob die folgende Position der Erwartung entspricht und wenn nicht, die Folgeposition über die `PosAt`-Methode des `CollisionEnvironment` korrigiert. Hierfür ist zudem eine extra Bedingung und eine Flag innerhalb des Systems notwendig, um die normale Logik, die durch diese potenziell erneut auftretende Kollision ausgeführt werden würde, zu umgehen und stattdessen den `CollisionType.Pass` zurückzugeben, um diese Korrektur zuzulassen. Neben der Anpassung der hier beschriebenen Methoden wurden die notwendigen Funktionen für die Behandlung in der Utility-Klasse `HandballMASHelper` umgesetzt, um Redundanzen zu vermeiden, da alle Klassen auf diese Funktionen zugreifen können. Diese Utility-Klasse bietet zudem weitere Funktionen an, die insbesondere der Berechnung von Winkeln, Distanzen und Positionen auf dem Spielfeld dienen. Außerdem wird über die Klasse die Nutzung eines einheitlichen `FastRandom`-Zufallsgenerator ermöglicht. Auf diese Art können Simulationsläufe unter Verwendung eines Seeds wiederholt werden, auch wenn diese aufgrund des Scheduling durch MARS nicht komplett identisch wären.

5.4.2 Umsetzungsdetails

Wie in dem Systementwurf beschrieben, sollen sowohl die Bewegung des Balls als auch die Bewertung der Spielsituation durch den Schiedsrichter nach allen `Tick`-Methoden der Agenten ausgeführt werden. Hierfür bietet das MARS-Framework bei Verwendung des `ISteppedLayer`-Interface für Layers die Methoden `PreTick`, die nach Inkrementieren des aktuellen Ticks und vor Aufruf der `Tick`-Methoden aufgerufen wird, und `PostTick`, die ausgeführt wird, nachdem die Logik in den `Tick`-Methoden durchgeführt wurden. Für die Bewegung des Balls ist die `PostTick`-Methode die geeignete Wahl, da diese wie erwähnt nach sämtlichen `Tick`-Methoden des Systems ausgeführt wird, so also die oben genannte Bedingung erfüllt. Zudem wird auf diese Weise das Persistieren des Zustands des Balls, der sich durch die Bewegung verändern kann, noch innerhalb des Simulationsschritts durchgeführt, in dem diese ausgelöst wird. Wirft ein Spieler beispielsweise den Ball in einem Tick X, so ist die initiale Bewegung durch diesen Wurf bereits in Tick X durchgeführt und wird in der entsprechenden Log-Datei widergespiegelt. Dies ermöglicht es, die Bewegungen nachvollziehbar zu visualisieren, da die Synchronisation zwischen Aktionen der Spieler-Agenten und den Bewegungen des Balls vorhanden ist.

Für die Prüfung der Spielsituation sollten zuerst die Aktionen der Spieler-Agenten und die Bewegung des Balls abgeschlossen sein, um sicherzustellen, dass sich keine ungewollten Seiteneffekte ergeben und, wie zuvor beschrieben, um Nachvollziehbarkeit zu ermöglichen. Diese Bedingung spricht dafür, die Ausführung dieser beiden Aufgaben zeitlich voneinander zu trennen, da die Nebenläufigkeit von MARS sonst zu Konflikten oder Seiteneffekten führen könnte. Um dies zu erreichen, wird die Prüfung durch den Schiedsrichter in der `PreTick`-Methode des folgenden Simulationsschritts ausgeführt. So wird das Persistieren der Bewegungen nicht durch das mögliche Erstellen neuer Spielsituationen beeinflusst. Dies vereinfacht die Behandlung für die Visualisierung, da lediglich die neuen Positionen nach einem Verschieben durch den Schiedsrichter in einer Datei vermerkt werden müssen, anstatt die Positionen der Elemente der Simulation jeden Tick manuell abzuspeichern.

5.4.3 Verhaltensweisen der Agenten

Wie in Kapitel 4.4.1 beschrieben, existieren drei Agententypen, die sich in ihren Verhaltensweisen unterscheiden. Die Spieler sind in Torhüter (`PlanningGoalkeeperPlayerMind`), Angreifer (`PlanningAttackingPlayerMind`), Verteidiger (`PlanningDefending-PlayerMind`) unterteilt. Für alle drei Agententypen wurden Verhaltensmodelle unter Verwendung von GOAP entwickelt, um adaptives Verhalten zu erreichen und die Dynamik in der Simulation zu erhöhen. Dabei sind die Verhaltensmodelle für die Torhüter und Verteidiger einfach gehalten, während das für die Angreifer entwickelte Modell detaillierter ist. Der Grund hierfür ist, dass der Hauptfokus dieser Arbeit in der Umsetzung des zugrunde liegenden Systems und der Angreifer liegt, um die Anforderungen REQ-1 und REQ-4 zu erfüllen. Daher wird in diesem Kapitel primär auf das Verhaltensmodell der Angreifer eingegangen, während die anderen nur kurz beschrieben werden.

Im Laufe der Entwicklung mussten die Modelle ständig angepasst werden, da Fehler in den anfänglich aufgestellten Konzepten entdeckt wurden oder die modellierten Kombinationen nicht zu den erhofften Resultaten führten. Die hier beschriebenen Modelle stellen die aktuelle Version dar, bei denen jedoch weitere Anpassungen notwendig wären, um ein authentisches Verhalten zu erzeugen. Eine detaillierte Auswertung der Verhaltensweisen wird in den folgenden Kapiteln vorgenommen.

Um den Agenten die Bewertung der aktuellen Spielsituation in Bezug auf die Torgefahr, die von bestimmten Angreifern ausgeht, zu erleichtern, wird das Konzept der XGoals verwendet (siehe Kapitel 2.4.3). Die von Adams et al. (2023) beschriebenen Werte für einzelne Aspekte wurden dabei in Prozentwerte umgewandelt, um den Einfluss dieser Aspekte auf die Torwahrscheinlichkeit zu quantifizieren. Basierend auf diesen Werten wurde eine Formel entwickelt, die den XGoals-Wert möglichst präzise berechnet. In der Klasse `PlayerMind` wurden entsprechende Methoden implementiert, die die relevanten Aspekte mithilfe der verfügbaren Sensordaten bewerten. Einige Werte wurden jedoch ausgelassen, wenn eine Bewertung nicht möglich oder nur schwer umzusetzen war. Mit der Methode `CalculateXGoalsForPlayer` können die Agenten so einen *double*-Wert zwischen 0 und 1 berechnen, der die Wahrscheinlichkeit eines Torerfolgs ausdrückt und somit bei der Bewertung der Spielsituation unterstützt.

Des Weiteren ist für die Steuerung des Verhaltens wichtig, welche Rolle innerhalb des Teams ein Agent einnimmt. Die verfügbaren Rollen wurden bereits in Kapitel 2.4.1 beschrieben. Aus Gründen der Einfachheit wurde sich in diesem System darauf begrenzt, die offensiven Rollen zuzuweisen. Dies entspricht dem üblichen Vorgehen im Handball, da Spieler eher über diese Rollen eingeteilt werden. Im System sind diese Rollen durch den Enum-Typ `Role` umgesetzt, derer Ausprägung ein Kürzel für die englischen Begriffe für die Rollen darstellt. In Tabelle 3 werden die möglichen Ausprägungen aufgelistet.

Tabelle 3: Die Ausprägungen des Enum-Typs `Role`.

Rollenname auf deutsch	***Rollenname auf Englisch***	***Ausprägung***
Torwart	Goalkeeper	`Role.GK`
Linksaußen	Left Wing	`Role.LW`
Rückraum Links	Left Back	`Role.LB`
Rückraum Mitte	Center Back	`Role.CB`
Rückraum Rechts	Right Back	`Role.RB`
Rechtsaußen	Right Wing	`Role.RW`
Kreisspieler	Line Player	`Role.LP`

PlanningGoalkeeperPlayerMind

Die Torhüter sind in dem System mit einer Menge von drei Zielen ausgestattet, die sie verfolgen können. Diese Ziele sind das Verhindern eines Tors (`SaveGoal`), das Bereitsein für einen möglichen Wurf (`BeReadyForThrow`) und das Aktualisieren der ihm verfügbaren Daten (`GetAllInfo`). Aus dieser Menge ist das primäre Ziel `SaveGoal`, da dies die Hauptaufgabe der verteidigenden Mannschaft in dem situativen Nullsummenspiel Handball ist. Da dieses Ziel jedoch nur aktiviert wird, wenn die entsprechenden Ausprägungen der Zustandsparameter gegeben sind, existieren die weiteren Ziele, um optimal vorbereitet zu sein, wenn der Ernstfall auftritt.

Die komplette Menge der Zustandsparameter ist in Tabelle 4 aufgelistet und dabei in logische Aspekte gruppiert, über welche die Parameter Aussagen treffen. Die Zustandsparameter des

Aspekts „Torgefahr“ geben Auskunft darüber, ob und wodurch eine akute Torgefahr durch das gegnerische Team ausgeht, um darauf reagieren zu können. Die Teilmenge „Positionierung“ dient der Bewertung der eigenen Position und Ausrichtung, teilweise in Relation zu dem Ball.

Um die Ziele zu erreichen, stehen den Torhütern zudem eine Reihe von Aktionen zur Verfügung, welche ein bestimmtes Verhalten auslösen und so die Zustandsparameter beeinflussen können. Die Aktionen, welche die Torhüter ausführen können, heißen `LookForBall`, `Move`, `RotateBodyToBall`, `SaveBall`. Durch `LookForBall` und `RotateBodyToBall` kann der Agent sich in Richtung des Balls ausrichten und nach diesem suchen, während `Move` dazu dient, sich optimal in Relation zu dem Ball und dem Tor zu positionieren. `SaveBall` berechnet auf Basis des Gedächtnisses, in welcher Richtung und Höhe der Ball gehalten werden soll und führt die `Save`-Methode des `IPlayerBody`-Objekts aus. Dabei werden durch Normalverteilung generierte Zufallswerte genutzt, um gelegentlich Exploration durchzuführen, statt die logisch erarbeiteten Werte zu verwenden. Die Wahrscheinlichkeit für Exploration nimmt dabei mit steigender Anzahl an Würfen, die für einen bestimmten Gegenspieler vermerkt sind, ab.

Tabelle 4: GOAP-Zustandsparameter für `PlanningGoalkeeperPlayerMind`.

Aspekt	***Zustandsparameter***	***Mögliche Werte***
Torgefahr	HasBallCarrierBeeline	True/False
	HasBallCarrierHighXGoals	True/False
	IsBallCarrierInCloseThrowZone	True/False
	IsBallCarrierJumping	True/False
	IsBallThreateningGoal	True/False
Positionierung	IsAtDesiredDistanceToGoal	True/False
	IsFacingBall	True/False
	IsInGoodTacticalPosition	True/False
	IsOnBeelineOfBallToGoal	True/False

PlanningDefendingPlayerMind

Die Verteidiger haben ebenfalls die Möglichkeit, eines von drei verfügbaren Zielen zu verfolgen. Die Ziele sind das Verteidigen des Spielers in Ballbesitz (`DefendBallCarrier`), das Verteidigen eines Wurfs (`DefendThrow`) und das Verschieben und Positionieren innerhalb der Abwehr (`KeepInFormation`). Diese Ziele werden basierend auf den in Tabelle 5 aufgeführten Zustandsparameter aktiviert.

Die in der Tabelle aufgeführten Gruppierungen ähneln denen aus Tabelle 4, allerdings unterscheiden sich die Parameter. Die Parameter, die unter die Gruppierung „Torgefahr" fallen, dienen der Bewertung, wie viel Torgefahr durch die Angreifer, insbesondere auch in der direkten Nähe, ausgestrahlt wird. Die Gruppe „Verteidigungszustand & Positionierung" stellt dar, ob der Agent korrekt positioniert ist. Zudem beinhaltet sie relevante Aspekte über den Zustand dieses Verteidigers und des ihm zugeordneten Gegenspielers.

Tabelle 5: GOAP-Zustandsparameter für `PlanningDefendingPlayerMind`.

Aspekt	***Zustandsparameter***	***Mögliche Werte***
Torgefahr	IsClosebyOpponentThreatening	True/False
	IsBallClose	True/False
	IsBallThrown	True/False
	HasBallcarrierBeeline	True/False
Verteidigungszustand & Positionierung	IsDefending	True/False
	IsOpponentInPossession	True/False
	IsInPosition	True/False

Die Aktionen, die diesem Agententypen zur Verfügung stehen, sind `Block`, `CloseGap`, `DefendAttacker` und `GetInPosition`. Dabei dienen `CloseGap` und `GetInPosition` dazu, sich innerhalb der Abwehrformation zu bewegen und zu positionieren. Die Bewegungen finden in Relation zu der Position des Balls, des zugeordneten Gegenspielers, dem zugeordneten Bereich des Spielfelds und der Position des Tores statt. `DefendAttacker` dient dazu, den ballführenden Gegenspieler zu verteidigen und zu stören. `Block` setzt das Verteidigen eines Torwurfs um.

PlanningAttackingPlayerMind

Die Agenten der Klasse `PlanningAttackingPlayerMind` verfügen, wie zuvor beschrieben, über ein detaillierteres GOAP-Modell als die beiden anderen Agenten-Arten. Dieses wird hier näher beschrieben.

Die Angreifer können ebenfalls mehrere verschiedene Ziele verfolgen, wobei das Hauptziel eines Spielers das Erzielen eines Tores ist (`ScoreAGoal`). Neben diesem Ziel kann der Agent auch als Ziel haben, einen Mitspieler anzuspielen, der eine bessere Chance hat, ein Tor zu erzielen (`PassToTeammate`). Während die Aktivierung dieser Ziele von mehreren Zustandsparametern abhängt, ist vor allem erwähnenswert, dass eine der Vorbedingungen ist, dass der Agent den Ball hat. Das Ziel, sich auf eine Lücke in der Abwehr zuzubewegen (`MoveTowardDefensiveHole`), sowie das Ziel sich nach den anderen Spielern und dem Ball umzusehen (`GetCurrentInfo`), kann unabhängig von dem Ballbesitz aktiviert werden. Ohne in Ballbesitz zu sein sind die zusätzlichen Ziele, die ein Angreifer verfolgen kann, sich selbst freizulaufen und eine Gefahr für die Verteidiger auszustrahlen (`GetOpenAndPoseThreat`), oder eine Chance für einen Mitspieler zu ermöglichen (`CreateScoringChanceForTeammate`).

Die vollständige Menge der Zustandsparameter, welche die Grundlage der Entscheidungen der Angreifer in dem System bilden, sind in Tabelle 6 aufgeführt und in fünf verschiedene Aspekte unterteilt. Die Teilmenge, die als „Spielzustand“ beschrieben ist, enthält nur einen Zustandsparameter und drückt aus, ob das Spiel durch den Schiedsrichter angehalten wurde. Dies dient primär, um einige der Aktionen auszuschließen und das Spiel auf angemessene Weise wieder zu starten (spezifisch durch einen Pass zum Mitspieler oder einen Torwurf). Die Teilmenge „Ballbesitz“ beinhaltet Zustandsparameter, die ausdrücken, ob der Spieler in Ballbesitz ist oder der Ball aktuell frei ist und hat wie im vorherigen Abschnitt beschrieben, primär den Zweck die Ziele zu steuern.

Tabelle 6: GOAP-Zustandsparameter für `PlanningAttackingPlayerMind`.

Aspekt	***Zustandsparameter***	***Mögliche Werte***
Ballbesitz	HasBall	True/False
	IsBallFree	True/False
	IsClosestTeammateToBall	True/False
Torwahrscheinlichkeit	HasScoringOpportunity	True/False
	HasTeammateScoringOpportunity	True/False
	IsInCloseRange	True/False
	IsInThrowingRange	True/False
	IsPosingThreatToDefence	True/False
	IsTeamScoringAGoal	True/False
	HasHighXGoals	True/False
	IsTallerThanDefender	True/False
	HasTeamOverloadOnASide	True/False
	IsGkPreparedForThrow	True/False
Taktik und Position	HasCurrentInfoOnOutfieders	True/False
	IsFacingBall	True/False
	IsFacingGoal	True/False
	IsSeeingBall	True/False
	IsSeeingOpenTeammate	True/False
	IsOpen	True/False
	IsTeammateOpen	True/False
	HasBeelineToGoal	True/False
	IsJumping	True/False
	CanBlock	True/False
	IsCloseToDefence	True/False
Auftakthandlungen & Anweisungen	HasInstructionInPlay	True/False
	IsPlayCalled	True/False
	KnowsPlays	True/False
	IsPlaymaker	True/False
Spielzustand	IsGameRunning	True/False

Die Teilmenge „Torwahrscheinlichkeit“ umfasst solche Zustandsparameter, die Aussagen über die Chance auf ein Tor durch den Agenten selbst oder seine Mitspieler treffen. Sie sind oftmals Effekte der modellierten Aktionen sowie Zielzustände, die ein Agent zu erreichen versucht. Die Teilmenge „Taktik und Positionierung“ beinhaltet dahingegen eher solche Zustandsparameter, die Informationen oder Bewertungen der eigenen Positionierung und das Wissen über das Umfeld darstellen. Parameter dieser Gruppe sind oftmals notwendige Zwischenschritte für das Erreichen eines Zielzustands, da sie als Vorbedingungen der Aktionen eingesetzt werden. Zuletzt existiert die Gruppe der „Auftakthandlungen & Anweisungen“, welche die Zustandsparameter mit Bezug auf die Ausführung von `Play`s und `PlayInstruction`s bündeln.

In Anhang C ist eine Auflistung aller Aktionen mit den zugehörigen Vorbedingungen und Effekten zu finden. Die Aktionen und ihre Umsetzungen werden hier detaillierter beschrieben.

Die Aktion `Throw` führt einen Wurf auf das gegnerische Tor aus und wählt dabei, basierend auf den Erinnerungen an vorherige Würfe aus, in welche Ecke des Tors geworfen wird. In einigen Fällen wird die Höhe des Wurfs, die Seite des Wurfs oder beide zufällig ausgewählt, um Exploration auszuführen und die Berechenbarkeit zu verringern. Ähnlich wie bei `PlanningGoalkeeperPlayerMind` wird dies mit Hilfe den durch Normalverteilung generierte Zufallswerte und Wahrscheinlichkeiten, die mit steigender Zahl von Würfen in dem Gedächtnis abnimmt, umgesetzt. Um sich in eine bessere Situation für einen Torwurf zu bringen, verfügt ein Angreifer in Ballbesitz zudem über die Aktion `Jump`, durch die hochgesprungen wird, um über die Verteidiger hinweg zu werfen. `WaitToThrow` dient als Zwischenschritt nach dem Aufruf von `Jump`, wenn sich in der aktuellen `JumpingPhase` immer noch keine Möglichkeit bietet zu werfen. Um sich auf Lücken in der Abwehr zwischen den Gegenspielern zuzubewegen und so zu versuchen, eine freie Wurfbahn zu erhalten, steht zudem die `MoveTowardsDefensiveHole`-Aktion zur Verfügung. Diese kann zudem auch ohne Ballbesitz ausgeführt werden, um Druck auf die Abwehr auszuüben. Ist ein Spieler in Ballbesitz, kann er, statt sich eine Wurfsituation zu erarbeiten und zu werfen, auch durch die Aktion `Pass` einen Pass zu einem seiner Mitspieler spielen, um so einem Mannschaftskameraden einen Torwurf zu ermöglichen. Bei dieser Aktion prüft der Spieler, welche der Mitspieler anspielbar sind (also freistehen und den Ball sehen können) und spielt dann einen Pass zu einem von ihnen.

Ist der Spieler nicht in Besitz des Balls, steht ihm neben `MoveTowardsDefensiveHole` zudem die Aktion `GetOpen` zur Verfügung, durch die er sich freiläuft und sich dem Ballführer anbietet, um einen Pass empfangen zu können. Sollte der Ball zudem frei sein, also auf dem Boden liegen, kann der Spieler sich über `PickUpBall` zum Ball begeben und diesen aufheben, falls er nah genug zum Ball ist. Wenn der Zustandsparameter `CanBlock` für den Spieler mit dem Wert `true` belegt ist, ist zudem die Aktion `BlockFor` verfügbar, die bewirkt, dass der Spieler eine Sperre an einem Verteidiger setzt, um einem Mitspieler den Weg zum Tor freizumachen. Zunächst ist diese Aktion nur für den Spieler mit der Rolle `Role.LP` vorgesehen, da diese im Handball am häufigsten Sperren stellen und sich aus Komplexitätsgründen für diese Eingrenzung entschieden wurde.

Für das Ausführen der Auftakthandlungen, die ein Trainer von außen in das System gegeben hat, stehen zudem die Aktionen `CallPlay` und `ExecutePlayInstruction` zur Verfügung. Über `CallPlay` kann der Spielmacher (aktuell immer der Spieler mit der Rolle `Role.CB`) ein `Play` ansagen, welches dann aktiv wird. Nur wenn bereits ein `Play` angesagt wurde und der Spieler in der aktuellen Phase eine `PlayInstruction` zugewiesen hat, kann die Aktion `ExecutePlayInstruction` ausgeführt werden. Diese führt die entsprechende Methode für die gegebenen Anweisung aus.

Um sich nach anderen Spielern oder dem Ball umzusehen, wenn er diese aktuell nicht sehen kann, verfügt ein `PlanningAttackingPlayerMind`-Agent über die Aktionen `LookForBall` und `LookForPlayer`.

Abbildung 9 zeigt eine mögliche Sequenz von Aktionen, die ein `PlanningAttackingPlayerMind`-Agent verfolgen kann, um von einem Startzustand, in dem er in Ballbesitz ist, sein Ziel zu erreichen, ein Tor zu erzielen. Dieser sogenannte „Happy-Path“ stellt den Optimalfall dar, in dem die ausgeführten Aktionen direkt zu den definierten Effekten führen und nicht mehrfach dafür aufgerufen werden müssen. Für die Angreifer existieren mehrere Möglichkeiten, von dem beschriebenen Ausgangspunkt das Ziel zu erreichen. Für die Darstellung wurde sich hier entschieden, den Weg über die Ausführung der durch den Anwender spezifizierten Auftakthandlung zu beschreiben, da dies in der Regel der priorisierte Weg in diesem System sein sollte.

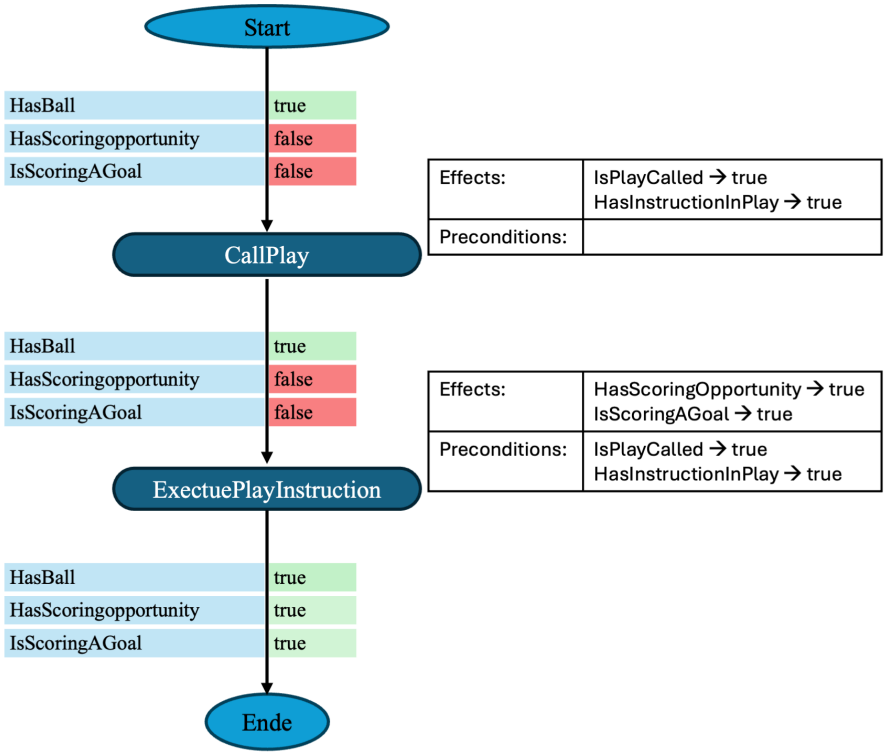

Abbildung 9: Darstellung des „Happy Path“, einem optimalen formulierten Plan für einen `PlanningAttackingPlayerMind`-Agenten und das Ziel `ScoreAGoal`.

5.5 Tests

Um die korrekte Funktionsweise der Kernfunktionalität des Systems sicherzustellen, wurden Unit-Tests entwickelt. Diese überprüfen die wichtigsten Funktionen für die Interaktionen innerhalb der Simulation.

Aufbau

Um die Funktionsweise zu testen, wird eine `TestPlayerMind`-Klasse angelegt, die simple Verhaltensweisen implementiert, um die relevanten Funktionen auszuführen und die Ergebnisse zu überprüfen. Innerhalb der Methoden führt dieser Agent zudem Prüfungen des Systemzustands durch. Darüber hinaus wird ein `TestPlayerMindLayer` ergänzt, welcher für die

Eingabe des konkreten Testfalls und den Zugriff auf bestimmte Felder des `PitchLayer` zuständig ist. Diese beiden ersetzen den `PlayerMindLayer` sowie die verschiedenen `PlayerMind`-Realisierungen des Systems, da die Korrektheit der Methoden für Interaktionen zu prüfen ist und nicht die konkreten Verhaltensweisen der Agenten.

Zu testende Aspekte

Zunächst wird in der Initialisierung geprüft, dass die Daten, die ein Nutzer oder eine Nutzerin in den Konfigurationsdateien eingegeben hat, korrekt in das System injiziert werden. Für die `PlayerBody`-Klassen wird hier insbesondere geprüft, dass die Team-Namen und Farben sowie die zugewiesene Rolle, die Position auf dem Spielfeld und die Ausrichtung des Körpers initial korrekt gesetzt wird. Für die `Ball`-Klasse wird geprüft, dass dieser entweder an der angegebenen Position platziert wurde oder, wenn diese Argumente übergeben wurden, dass der Ball sich in Besitz des spezifizierten Spielers befindet und an dessen Position platziert wurde. Zudem wird geprüft, dass ein übergebenes `Play` korrekt in das System eingelesen wird und die Agenten so den Zugriff auf die Anweisungen von außen erlangen.

Die zu testenden Funktionen sind für ein `PlayerBody`-Objekt einerseits die `TakeStep`-Methode, bei der zu testen ist, dass bei gültigen Argumenten (eine gültige Kombination aus `StepType`, Ausrichtung des Körpers und Richtung des Schritts) der Agent an die erwartete Position bewegt wird. Bei Übergabe von ungültigen Argumenten sollte stattdessen die Position nach Ausführung unverändert sein. Andererseits werden einige Fälle der `HandleCollision`-Methode überprüft. Diese Prüfung wird mit dem Test des Haltens durch den Torhüter kombiniert, indem ein angreifender `TestPlayerMind`-Agent einen bestimmten Wurf ausführt und der Torhüter die `SaveBall`-Methode in die entsprechende Richtung und `SaveHeight` ausführt.

6 Experimente und Ergebnisse

Um das entwickelte System zu überprüfen und in Hinblick auf die in Kapitel 3 beschriebenen Anforderungen bewerten zu können, werden eine Reihe von Experimenten durchgeführt. Diese Experimente werden hier beschrieben und deren Ergebnisse dokumentiert. Es ist dabei zu beachten, dass die Betrachtung dieses Systems in einigen Aspekten sehr subjektiv, beziehungsweise nicht messbar ist, da es darum geht die Plausibilität der Simulation als eine Darstellung des Handballsports zu bewerten.

Die objektiven Aspekte der Ergebnisse sind neben der Auswertung der Log-Dateien auch die durch GOAP geplanten Aktionssequenzen sowie die konkreten Entscheidungen, die innerhalb von bestimmten Aktionen getroffen werden, wie beispielsweise das Ziel eines Wurfs. Hierfür werden extra Log-Dateien erstellt, in denen diese Punkte festgehalten und nachfolgend ausgewertet werden können.

Für die Experimente, die komplexe Angriffe überprüfen sollen, wird eine Anzahl von 160 Ticks gewählt, da jeder Zeitschritt in den verwendeten Systemeinstellungen eine halbe Sekunde darstellt. Ein durchschnittlicher Angriff in der Handball Bundesliga dauert laut dem Analysten Julian Rux (2023, 2024) etwa 33,5 Sekunden, sodass dies ausreichend Zeit für zwei Angriffe lassen sollte, wenn diese realitätsnah ausgeführt werden.

Das innerhalb des Systems verwendete `FastRandom`-Objekt wird standardmäßig kein expliziter Seed übergeben, sodass jeder Experimentdurchlauf Unterschiede aufzeigen sollte. Für die Reproduzierbarkeit wird der Seed jedoch jeweils geloggt und in dem entsprechenden Dateinamen der Experiment-Daten festgehalten.

6.1 Experiment 1: Einzelverhalten

Dieses Experiment hat den Zweck, das Einzelverhalten eines Angreifers zu untersuchen, um auszuwerten, wie ein Agent sich verhält, wenn er auf sich allein gestellt ist und keine Mitspieler mit einbeziehen kann.

Aufbau

Das Experiment sieht eine Eins-gegen-Eins-Situationen vor, die der angreifende Spieler zu bewältigen hat. Der Angreifer tritt in der Situation gegen einen einzigen Verteidiger an und muss sich gegen diesen durchsetzen. Zudem steht der Torhüter als Hinterhalt im Tor. In diesem Experiment ist nicht vorgesehen, dem Angreifer Anweisungen in Form eines `Plays` zu übergeben.

Es werden zwei verschiedene Situationen vorgesehen, in denen der Angreifer gegen den Verteidiger antritt. In der ersten Situation wird das Einzelverhalten in der Mitte des Feldes betrachtet, während in der zweiten Situation überprüft werden soll, wie sich ein Spieler auf außen verhält.

Tabelle 7: Die Experiment-Parameter für das Experiment 1.

Parameter	***Wert***	***Anmerkung***
Anzahl Ticks	160	Nach der Beendigung der ersten Spielsituation treten im Eins-gegen-Eins keine weiteren relevanten Handlungen auf, da das Spiel nicht durch einen Pass gestartet werden kann.
Anzahl Angreifer	1	
Anzahl Verteidiger	1, 2	In dem ersten Abschnitt des Experiments nur ein Verteidiger (`Role.GK`), danach wird ein Feldspieler ergänzt.
Auftakthandlung eingegeben	Nein	

Ablauf

Für das Experiment werden je Situation je fünf Simulationsläufe ausgeführt. Diese Durchläufe werden in zwei Abschnitte geteilt, innerhalb derer die Startkonfiguration jeweils identisch ist. So kann geprüft werden, wie sich das System in Hinsicht auf die Generalisierung verhält, also

wie sehr sich die Resultate der Simulationsläufe unterscheiden. Zwischen den beiden Abschnitten werden diese Konfigurationen geändert, um unterschiedliche Ausgangslagen betrachten zu können.

In den ersten fünf Durchläufen ist der angreifende Spieler derjenige mit der Rolle `Role.CB` und dieser wird mittig zum Tor positioniert. Er tritt gegen den Verteidiger mit derselben Rolle an, welcher leicht versetzt zu dem Angreifer auf dem Spielfeld platziert wird, sodass der Angreifer zunächst einen freien Weg zum Tor hat. In der zweiten Situation ist der Angreifer ein Außenspieler mit der Rolle `Role.LW`, der in der Ecke außen positioniert wird und gegen den Verteidiger `Role.RW` antritt.

Ergebnisse

In der ersten Situation, wo der Angreifer mittig und der Verteidiger versetzt dazu beginnt zeigt sich, dass der Angreifer in allen Durchläufen beginnt, sich direkt auf das Tor zuzubewegen, bis er nah genug an das Tor kommt. Dann gibt er einen Wurf auf das Tor ab. Der Verteidiger plant in den ersten Simulationsschritten jeweils eine Kombination aus `GetInPosition` gefolgt von `CloseGap` und führt diese, auch ohne Formulieren eines neuen Plans zwischendrin, nacheinander aus. Nachdem der Angreifer auf das Tor wirft, ändert sich das Verhalten zu `Block`. Der Torhüter plant, bis der Torwurf abgegeben wird, die Aktion `Move`. Danach wechselt er zur Aktion `SaveBall`. Hier fällt auf, dass diese Aktion in einigen Durchläufen bereits in dem Tick ausgeführt wird, in dem der Ball geworfen wird, während dies in anderen erst in dem darauffolgenden Tick stattfindet. Dies zeigt, dass die gewünschte Dynamik bezüglich der Reaktionen auf Würfe, die durch die Bewegungsmodelle erreicht werden sollte, gegeben ist.

Die Auswertung der Logs sowie die Betrachtung der grafischen Darstellung zeigt, dass die Bewegungen der Agenten sich teilweise in den Richtungen, in die eine Aktion ausgeführt wird, unterscheiden. Dies ist insbesondere bei dem Werfen und Halten zu beobachten. Aus den fünf Durchläufen wirft der Angreifer dreimal nach rechts und zweimal in die Mitte des Tors (von sich aus gesehen). Der Torhüter führt seine Haltebewegung einmal zur linken Seite und je zweimal in die Mitte und nach rechts aus. Bei der Ausführung nach links in Durchlauf eins ist zu beobachten, dass der Torhüter den Ball hätte halten können, dies aber nicht gelingt, da er die mittlere Höhe abdeckt, während der Ball nach oben geworfen wurde.

Um zu prüfen, ob eine Änderung in dem Ablauf auftritt, wenn der Verteidiger aus einer besseren Ausgangsposition startet, wurde das Experiment um drei weitere Durchläufe aus der Mitte erweitert, in denen der Verteidiger etwas mittiger startet. Hier kann ein ähnliches Verhalten wie in dem vorherigen Aufbau beobachtet werden, jedoch schafft es der Verteidiger hier besser, sich in den Weg des Angreifers zu stellen. Dabei richtet er sich jedoch in den drei beobachteten Situationen eher etwas zur linken Seite des Angreifers aus, also zur anderen Seite versetzt, im Vergleich zu der Startposition. Durch diese Position gelingt es dem Verteidiger jedoch in dem dritten Durchlauf, den Torwurf zur linken Seite des Angreifers abzublocken und somit ein Tor zu verhindern.

Bei Beobachtung der zweiten Situation stellt sich heraus, dass der Angreifer jeweils direkt zu Beginn einen Torwurf aus der Ecke abgibt, also aus einem sehr schlechten Winkel zum Tor. So kann kein weiteres Verhalten für diese Situation analysiert werden. Vermutlich tritt dieses Verhalten auf, weil der Angreifer sich in der Nahwurfzone befindet und zudem einen direkten Weg zum Tor hat. So sind in dem beschriebenen GOAP-Modell die Voraussetzungen gegeben, damit der Angreifer den Plan formuliert, auf das Tor zu werfen.

6.2 Experiment 2: Mannschaftsverhalten

Der Zweck dieses Experiments ist, die volle Funktionsweise des Systems zu überprüfen, indem zwei vollständige Mannschaften gegeneinander antreten. Darüber hinaus soll in diesem Experiment auch die Umsetzung der Auftakthandlungen überprüft werden.

Aufbau

Da hier vollständige Mannschaften zum Einsatz kommen, werden sechs `PlanningAttackingPlayerMind`-Agenten, sechs `PlanningDefendingPlayerMind`-Agenten und zwei `PlanningGoalkeeperPlayerMind`-Agenten in die Simulation geladen. Die Feldspieler erhalten jeweils (über die zugeordneten `PlayerBody`-Objekte) eine der sechs Spielerrollen zugeordnet, sodass eine standardmäßige Mannschaft repräsentiert wird.

Die Angreifer erhalten Anweisungen für eine Auftakthandlung, da dies ein zentraler Punkt des Systems ist. Es handelt sich dabei um eine simple Auftakthandlung, bei der in der ersten Phase

der Agent mit der Rolle `Role.LP` einen Block an dem linken Verteidiger in Lücke 4 setzen soll (also zwischen dem vierten und fünften Abwehrspieler von links). In Phase zwei soll dann der Spieler mit der Rolle `Role.CB`, der den Ball hat, eine Kreuzung für den Spieler `Role.RB` spielen. Als alternative Anweisung soll er einen Pass zum Spieler `Role.LB` spielen. In Phase 3 soll dann derjenige der beiden, der den Ball erhalten hat, einen Torwurf abgeben. Diese Auftakthandlung ist in Abbildung 10 abgebildet.

Tabelle 8: Die Experiment-Parameter für das Experiment 2.

Parameter	***Wert***	***Anmerkung***
Anzahl Ticks	160	
Anzahl Angreifer	7	Eine vollständige Mannschaft. Jeder Rolle genau einmal zugeordnet.
Anzahl Verteidiger	7	Eine vollständige Mannschaft. Jeder Rolle genau einmal zugeordnet.
Auftakthandlung eingegeben	Ja	Eingabe durch manuell erstellte Konfigurationsdateien.

Ablauf

Die Angreifer werden ihrer Rolle entsprechend positioniert, sodass beispielsweise die Rückraumspieler auf etwa 11 Meter Entfernung von der Torlinie auf der Mitte und nahe der jeweiligen Seitenlinie initialisiert werden. Die Außenspieler werden in den Ecken des Spielfelds platziert und der Kreisläufer mittig zwischen den Abwehrspielern. Der Ball wird dem Rückraum Mitte Spieler übergeben. Die Verteidiger werden in einer „6:0“-Verteidigungsformation etwa eineinhalb Meter von dem Torkreis entfernt positioniert. Die Torhüter werden jeweils mittig vom Tor direkt vor der Torlinie platziert.

Für dieses Experiment sind fünf Simulationsläufe vorgesehen, die jeweils mit denselben Ausgangslagen starten. Innerhalb des Durchgangs prüft der `RefereeLayer` wie beschrieben auf Umstände, die die Angriffssituation beenden und stellt nachfolgend eine neue her. Hierbei wurde sich darauf beschränkt, jeweils mit einem Anwurf für die Angreifer fortzufahren. So können die Ausgangslagen für die neuen Situationen miteinander verglichen werden.

Ergebnisse

In den fünf Durchläufen ist zunächst zu beobachten, dass sie alle auf ähnlich Art starten, da der Angreifer mit der Rolle `Role.CB` auf Höhe der Mitte des Tors im Ballbesitz ist. Er läuft auf die Verteidigung zu, wobei die beiden Verteidiger mit Rolle `Role.LP` und `Role.CB` die Lücke zwischen ihnen beiden etwas zu schieben. Dies geschieht durch eine Kombination der Aktionen `CloseGap` und `GetInPosition`. Das Ergebnis des Wurfes, der stattfindet, sobald der Angreifer nah genug zum Tor kommt, führt dabei zu unterschiedlichen Ergebnissen. In Durchlauf 1, 4 und 5 wird der Wurf geblockt, während in Durchlauf 2 und 3 durch die beiden Abwehrspieler hindurch beziehungsweise an ihnen vorbei geworfen wird. So zeigt sich auch, dass sich der anvisierte Bereich des Tors zwischen den Durchläufen unterscheidet.

Nach dem Torwurf resultiert die Situation entweder in einem Tor oder darin, dass der Ball im Aus landet und es wird wie beschrieben eine neue Situation hergestellt, die mit einem Mittelanwurf beginnt. Dort ist in den Durchläufen 1, 2, 3 und 4 zu beobachten, dass die Angreifer an der Mittellinie bleiben und dort mehrere Pässe untereinander hin und her spielen, bis nach einer Sequenz von 2 bis 4 Pässen zumeist ein Pass gespielt wird, der nicht gefangen wird, sodass der Ball an dem Empfänger vorbeifliegt oder von diesem abprallt und jeweils in das Aus geht. Hier entsteht eine Schleife, in der sich dieses Muster wiederholt, wobei jedoch die Sequenzen von Aktionen und Passempfängern unterschiedlich sind. Bei diesem Muster sticht der Durchlauf 4 etwas hervor, da die Spieler dort anders als in den vorherigen Durchläufen nach dem Fangen des Balls ein paar Ticks warten, während sich die Mitspieler drumherum zu dem Ballführer bewegen. Dies erscheint wie ein Anbieten der Mitspieler. Bei Betrachtung der Logs stellt sich allerdings heraus, dass die Ballführer jeweils die Aktion `Pass` planen, diese aber nicht ausgeführt wird. Vermutlich liegt dies daran, dass die Mitspieler noch nicht zum Ball schauen und das Passen somit nicht erfolgreich sein würde. Außerdem wird in diesem Durchlauf zu einem Zeitpunkt eine bedeutend längere Sequenz von erfolgreichen Pässen ausgeführt als der zuvor beschriebene Durchschnitt, sodass erst nach 9 erfolgreichen Pässen ein Fehler unterläuft.

Darüber hinaus sticht Durchlauf 5 heraus. In diesem ist zunächst dasselbe Muster zu erkennen, bevor plötzlich bei einem neu hergestellten Mittelanwurf der Spieler mit Rolle `Role.LB` den Ball erhält und, anstatt wie in den vorherigen Durchläufen weiter zu passen, anfängt, in Richtung Tor zu laufen. Seine Mitspieler tun ihm dies gleich, sodass sich die Angreifer nach und

nach in Richtung Tor begeben. Der Ballführer läuft durch die Lücke in der Abwehr innerhalb des Bereichs für seine Rolle und wirft auf das Tor, trifft allerdings den rechten Pfosten und der Ball geht in das Aus. Nachfolgend findet erneut das Muster der Pässe an der Mittellinie statt. Zu beobachten ist hier zudem, dass die Verteidiger die Lücke nicht wie erhofft schließen, obwohl diese die Aktionen `CloseGap` und `GetInPosition` abwechselnd ausführt. Dies spricht dafür, dass die festgelegten Bereiche für die `PlanningDefendingPlayerMind`-Agenten ausgeweitet werden müssen.

In Durchlauf 2 landet der Ball nach Pässen auf dem Boden des Torraums der Abwehr, wo er liegen bleibt. Die Angreifer laufen nach dem Pass, der dazu führt, ebenfalls los in Richtung Tor. Da der Ball jedoch im Torraum landet, können die Agenten hier nicht zum Ball gelangen.

Die Auswertung der Logs über die erstellten Pläne zeigt, dass die durchschnittliche Länge der Pläne, die die Agenten erstellen, relativ klein ist. Insbesondere die Torhüter verfolgen zumeist nur Pläne, die eine einzige Aktion enthalten. Ähnlich ist es bei den `PlanningAttackingPlayerMind`-Agenten, auch wenn diese einen etwas höheren Durchschnittswert haben. Im Durchschnitt erstellen die Verteidiger die längsten Pläne mit einem Wert von 1,20322. Im Hinblick auf die einzelnen Spieler ist zu erkennen, dass die Spieler mit Rolle `Role.RW`, `Role.RB` und `Role.CB` im Angriff im Schnitt die längsten Pläne formulieren, während auf Seite der Verteidiger die Spieler auf der linken Seite die längsten Pläne vorzeigen. Hier scheint also die Länge der Pläne zwischen direkten Gegenspielern zu korrelieren. Allerdings formulieren die mittigen Verteidiger (`Role.CB` und `Role.LP`) die kürzesten Pläne, was mit der Positionierung in der Mitte des Feldes zu tun haben könnte.

Hierbei ist zu beachten, dass diese Zahlen nicht mit einbeziehen, ob die Planlänge beispielsweise verkürzt ist, da der Plan nicht ausgetauscht, sondern in zwei oder mehr Ticks hintereinander die geplante Aktionssequenz ausgeführt wurde und die Länge deswegen abgenommen hat. Für die Angreifer zeigt sich in der Tabelle allerdings, dass fast in jedem Tick ein neuer Plan formuliert wurde und diese Frage so nicht auftritt, jedoch ist es möglich, dass die durchschnittliche Länge neu formulierter Pläne bei den Torhütern und insbesondere bei den Verteidigern etwas höher ausfällt als es die Werte in der Tabelle erscheinen lassen.

In Hinsicht auf die Ausführung des übergebenen `Play` ist festzustellen, dass der Spieler `Role.CB` in den meisten Durchläufen zumindest einige Male die Aktion `CallPlay` und `ExecutePlayInstruction` plant. Erstere wird dabei auch teilweise ausgeführt, während letztere in den meisten Durchläufen nicht aufgerufen wird. Hier sticht der Durchlauf 2 heraus, da in diesem die `ExecutePlayInstruction`-Aktion zunächst in 22 Ticks durch den Angreifer mit der Rolle `Role.LP` ausgeführt wird. Nachfolgend tun die Spieler mit den Rollen `Role.CB` und `Role.RB` dies in den restlichen Ticks der Simulation. Dabei beginnt der kreuzende Spieler `Role.CB` einen Tick eher als der Spieler mit der Rolle `Role.RB`, der den Ball nach Kreuzung erhalten soll. Dies zeigt also, dass die Ausführung der Auftakthandlungen durchaus von den Agenten geplant werden kann, auch wenn dies in den meisten Durchläufen nicht passiert ist. Zudem ist zu erkennen, dass die Phasen eingehalten werden, da `Role.CB` und `Role.LP` erst dann die `ExecutePlayInstruction`-Aktion planen und ausführen, nachdem der Angreifer mit Rolle `Role.LP` dies nicht mehr tut, also mit seiner Anweisung fertig ist. Dass der Agent `Role.CB` vor dem Agenten `Role.RB` mit der Ausführung anfängt, zeigt zudem, dass Kommunikation stattfindet, da der Spielmacher `Role.CB` bereits einen Tick vorher weiß, dass die nächste `PlayPhase` erreicht wurde. Daraufhin kommuniziert er dies mit den Mitspielern, kann aber selbst schon mit dem Ausführen der Anweisung anfangen. Da Nachrichten erst im folgenden Tick zugestellt werden, ergibt sich dieser Unterschied von einem Tick zwischen den beiden.

6.3 Experiment 3: Nutzung des Systems

Im Rahmen des dritten Experiments wird untersucht, wie gut das entwickelte System von Handballtrainern genutzt werden kann und inwieweit es die Arbeitsweise von Trainern unterstützt. Im Mittelpunkt steht dabei die Frage, ob ein Trainer die Benutzeroberfläche des Systems intuitiv bedienen kann und ob die angebotenen Funktionen seiner gewohnten Vorgehensweise beim Aufzeigen und Anpassen von Spielzügen entsprechen. Besonders relevant ist hierbei, ob das System die kognitive Arbeitsweise der Trainer widerspiegelt und ihnen ermöglicht, Taktiken so einzugeben, wie sie es in der Praxis tun würden.

Ein weiterer wichtiger Aspekt dieses Experiments ist die Analyse der Fehlertoleranz des Systems. Es wird geprüft, ob Trainer verstehen können, wie sie eventuell gemachte Fehler bei der

Eingabe von Taktiken erkennen und beheben können. Darüber hinaus soll untersucht werden, ob das System ausreichende Unterstützung in Form von Anleitungen oder Hilfestellungen bietet, die den Trainer bei der Nutzung des Systems an die Hand nehmen und durch den Prozess der Erstellung und Anpassung von Spielzügen führen.

Somit zielt dieses Experiment darauf ab, die Nutzbarkeit und Effizienz des Systems aus der Perspektive der Trainer zu bewerten und aufzuzeigen, in welchen Bereichen das System eventuell verbessert werden muss, um den Anforderungen der Praxis gerecht zu werden.

Hintergrund

Vor der Beschreibung des Aufbaus dieses Experiments ist es zunächst notwendig auszuführen, wie ein Trainer oder eine Trainerin in der Regel bei der Beschreibung einer Auftakthandlung vorgeht. Dies schafft die Basis dafür, nachfolgend erklären zu können, wie das Vorgehen an dem System getestet wird und so Schlüsse für die Bewertung zu ziehen.

Die meisten Trainer und Trainerinnen gehen bei der Beschreibung von Auftakthandlungen ähnlich vor, wenn sie diese beispielsweise mithilfe einer Taktiktafel vermitteln. Der Prozess beginnt damit, dass die Positionen der Spieler festgelegt werden, in denen sie sich befinden sollten, um die Auftakthandlung zu starten. Daraufhin werden die Abläufe und Anweisungen konkretisiert und beschrieben, an welchen Punkten des Ablaufs normalerweise Entscheidungen für den Fortlauf getroffen werden müssen. Unter Umständen wird abschließend definiert, welcher Spieler zum Abschluss kommen soll, sollte sich nicht bereits vorher eine Gelegenheit ergeben haben. Die einzelnen Aspekte werden nachfolgend genauer betrachtet.

1. **Ausgangspositionen**: Der Trainer beginnt, indem er die Spieler auf dem Spielfeld positioniert. Dies ist entscheidend, da die Ausgangspositionen die Grundlage für den weiteren Ablauf des Spielzugs bilden. Beispielsweise kann hier festgelegt werden, dass der Linksaußen Spieler sich nicht in der Ecke des Spielfelds befinden, sondern seinem nächsten Mitspieler im Feld entgegenkommen soll.

2. **Anweisungen zur Bewegung**: Nachdem die Ausgangspositionen festgelegt sind, beginnt der Trainer damit, konkrete Anweisungen an einzelne Spieler zu definieren. Hierbei wird der Reihenfolge nach vorgegangen, wie die Anweisungen zeitlich ausgeführt

werden sollten. Beispielsweise können hierbei mehrere Anweisungen zur gleichen Zeit ausgeführt werden, oder beschrieben werden, dass für den Beginn einer bestimmten Anweisung auf das Ende eines anderen Teilablaufs gewartet werden muss.

3. **Entscheidungspunkte**: Während einer Auftakthandlung gibt es oftmals Momente, in denen der Trainer antizipiert, dass unterschiedliche Defensiv-Szenarien auftreten könnten. An diesen Punkten beschreibt der Trainer daher, welche unterschiedlichen Möglichkeiten der handelnde Angreifer hat, um die Auftakthandlung fortzuführen. So ergeben sich unterschiedliche Anweisungen, die der Angreifer befolgen kann.

4. **Torabschluss**: Es kann vorkommen, dass ein Trainer für eine Auftakthandlung vorsieht, dass ein bestimmter Spieler diesen mit einem Torwurf abschließt, sollte sich in dem vorherigen Ablauf nicht schon eine andere Tormöglichkeit ergeben haben. Dies definiert der Trainer dann ebenfalls. Beispielsweise könnte es sein, dass eine Auftakthandlung bei dem linken Rückraumspieler beginnt, und bis zum Rechtsaußen durchgespielt wird, da keine der Anweisungen auf dem Weg dahin eine Lücke in der Abwehr geschaffen haben. Der Trainer könnte dann definieren, dass der Rechtsaußen-Spieler den Torabschluss suchen soll.

Durch diesen strukturierten Ablauf wird sichergestellt, dass jeder Spieler genau weiß, welche Rolle er in der jeweiligen Situation spielt und an welchen Stellen besonders damit zu rechnen ist, dass eine Entscheidung getroffen werden muss. Diese detaillierten Vorgaben ermöglichen es, die Auftakthandlungen präzise zu planen und trotzdem während des Spiels flexibel auf die gegnerischen Verhaltensweisen zu reagieren.

Aufbau

Da in diesem Experiment die Nutzung durch einen Trainer getestet werden soll, wird hier zunächst eine simple Auftakthandlung beschrieben, die in das System eingegeben werden soll. Im Verlauf des Experiments soll diese nachfolgend angepasst und erweitert werden. Die vorgesehene Auftakthandlung ist in Abbildung 10 dargestellt. Es beginnt damit, dass der Linksaußen Spieler einen Pass zum Rückraum Links spielt. Nachdem er diesen Pass gespielt hat, läuft er zur Lücke 4 ein, während der Rückraum Links Spieler den Ball weiter zum Rückraum Mitte spielt. Der Rückraum Mitte spielt den Ball zum Rückraum Rechts, der daraufhin einen Torwurf

ausführen soll. Dies stellt also ein `Play` mit vier Phasen dar, in dem in der zweiten Phase zwei Spieler Anweisungen erhalten und sonst jeweils einer.

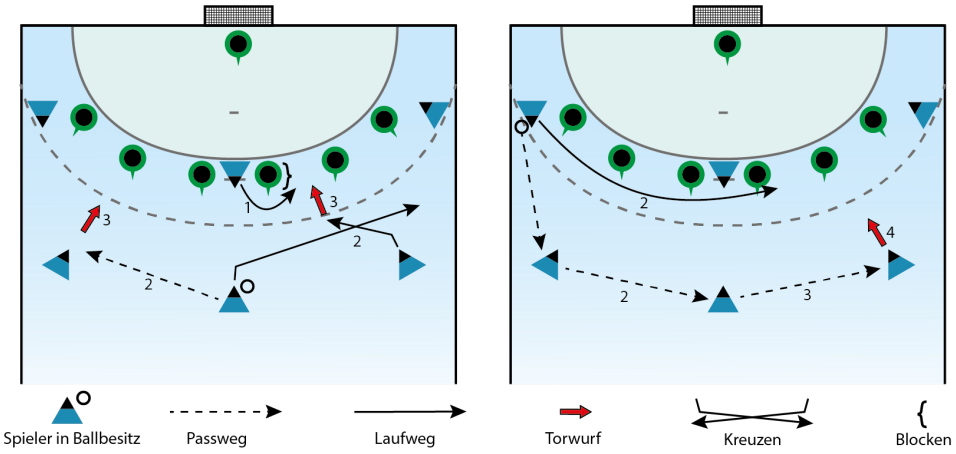

Abbildung 10: Darstellung der Auftakthandlungen aus Experiment 2 (links) und Experiment 3 (rechts).

Als Ausgangspunkt für dieses Experiment werden die Dateien *plays.json* und *player_config.csv* geleert. Der Anwender hat nun die Aufgabe, in dem Analyseprogramm die Spieler für die Ausgangssituation zu platzieren die beschriebene Auftakthandlung umzusetzen und beide abzuspeichern. Nachfolgend soll iterativ die Simulation ausgeführt, die Ergebnisse betrachtet und basierend darauf weitere Anpassungen an der Auftakthandlung vorgenommen werden, um diese zu optimieren.

Tabelle 9: Die Experiment-Parameter für das Experiment 3.

Parameter	***Wert***	***Anmerkung***
Anzahl Ticks	160	
Anzahl Angreifer	7	Eine vollständige Mannschaft. Jeder Rolle genau einmal zugeordnet.
Anzahl Verteidiger	7	Eine vollständige Mannschaft. Jeder Rolle genau einmal zugeordnet.
Auftakthandlung eingegeben	Ja	Eingabe durch Trainer in dem Visualisierungsprogramm. Im Verlauf weitere Anpassungen, um die Auftakthandlung zu optimieren

Ablauf

Das Experiment beginnt damit, dass der Trainer die Spieler gemäß der beschriebenen Auftakthandlung positioniert. Der Linksaußen Spieler wird in der Nähe der Kreuzung der Seitenaus-Linie und dem 9-Meter-Kreis platziert und erhält den Ball. Die Rückraumspieler befinden sich etwa 10 bis 11 Metern von der Torlinie entfernt in breiten Positionen, um die Angriffsformation zu unterstützen. In der Funktion zur Bearbeitung der Auftakthandlungen im Visualisierungsprogramm findet der Anwender eine leere Ansicht vor, da die *plays.json*-Datei für dieses Experiment geleert wurde. Anschließend werden über diese Funktion die zuvor beschriebenen Anweisungen für die Auftakthandlung Schritt für Schritt eingegeben und abgespeichert.

Nachdem die grundlegende Auftakthandlung eingegeben wurde, wird die Simulation mit den eingetragenen Anweisungen ausgeführt. Der Trainer analysiert die Simulationsergebnisse, die im Visualisierungsprogramm angezeigt werden, um die Effektivität des Spielzugs zu bewerten und mögliche Optimierungsmöglichkeiten und Entscheidungspunkt zu identifizieren.

Auf Basis dieser Analyse passt der Trainer die Auftakthandlung an, indem er etwa die bestehenden Anweisungen der Spieler anpasst oder zusätzliche Phasen und Aktionen ergänzt. Ziel dieser Anpassungen ist es, für die Angriffsstrategie Optimierungsmöglichkeiten umzusetzen, welche in der Analyse ausgemacht wurden und so bessere Torchancen zu erarbeiten. Nach jeder Anpassung wird die Simulation erneut ausgeführt, und die resultierenden Ergebnisse durch den Trainer bewertet.

Ergebnisse

Bei der Ausführung des Experiments wurde beobachtet, dass der Anwender für die Aufgabe die Spieler zu platzieren als ersten Instinkt hat, die Spieler in der Animationsansicht anzuklicken und zu versuchen diese an eine andere Stelle zu ziehen. Erst auf den zweiten Blick fällt der zuständige „Edit Positions“-Button auf. In der Pop-Up Ansicht zum Bearbeiten der Positionen wird erneut versucht zunächst die Spieler-Kreise auf das Spielfeld zu ziehen. Die Nutzungshinweise fallen nicht direkt auf, sodass versucht wird intuitiv zu handeln. Nachdem diese jedoch wahrgenommen wurden, ist es dem Anwender möglich, die Spieler wie gewünscht zu positionieren und dem Linksaußen Spieler den Ball zuzuweisen, wie vorgesehen.

Für das Definieren der Auftakthandlungen werden die relevanten Knöpfe zum Hinzufügen von Anweisungen und Phasen intuitiv verwendet. Bei der Bearbeitung der Anweisungen wird beobachtet, dass der Anwender sämtliche aufgeführten Felder befüllen möchte. Dadurch war es notwendig zu erklären, welche Felder für welche Anweisungen notwendig sind. Mit diesem Wissen war es dem Trainer möglich die vorgegebene Auftakthandlung entsprechend einzugeben.

Abbildung 11 zeigt Bildschirmfotos der Eingabe der vorgegebenen Daten durch den Anwender während der Ausführung des Experiments. Auf dem linken Bild ist die Positionierung der Spieler abgebildet, während rechts die Eingabe der Auftakthandlung zu sehen ist.

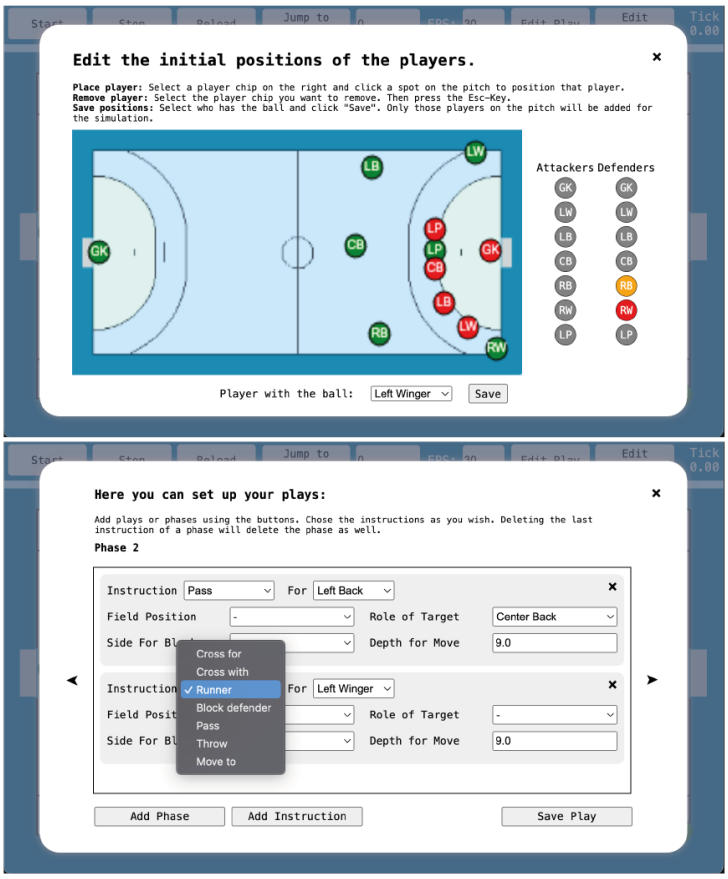

Abbildung 11: Bildschirmfotos der Nutzung des Systems durch einen Anwender in Experiment 3.

Nach dem Positionieren der Spieler und Definieren der vorgegeben Auftakthandlung wird die Simulation ausgeführt. Die Betrachtung der Spielerbewegung in dem Analyseprogramm zeigt jedoch, dass die Auftakthandlung nicht ausgeführt wurde, wie es auch bei Experiment 2 beobachtet werden konnte. Stattdessen bewegt der Linksaußen sich mit dem Ball auf die Lücke zwischen dem gegnerischen Außen rechts und Halb Links zu. Nach einigen Schritten verfügt er über einen freien Weg zum Tor und führt einen Torwurf aus. Daher konnte der beschriebene Ablauf für das Experiment nicht wie gewünscht ausgeführt werden. Trotzdem wurde das Ergänzen weiterer Anweisungen nachfolgend getestet. Da dem Anwender bereits bekannt war, wie die Auftakthandlungen bearbeitet werden können, stellte dies jedoch keine weitere Herausforderung dar.

7 Diskussion der Ergebnisse und Bewertung des Systems

Nachfolgend werden in diesem Kapitel die Ergebnisse aus den im vorherigen Abschnitt beschriebenen Experimenten ausgewertet. Basierend darauf wird zunächst die Qualität des Systems in Hinsicht auf die gesetzten Anforderungen betrachtet. Abschließend werden die Ergebnisse dahingehend ausgewertet wie nutzbar das System für Trainer und Trainerinnen ist und ob es diesen einen Mehrwert für ihre Arbeit bieten kann.

Betrachtet man die Ergebnisse der Experimente in Kapitel 6, so kann man festhalten, dass einige Ansätze in dem Verhalten der Spieler zu erkennen sind, um eine realitätsnahe Handballsimulation darzustellen. Die Agenten zeigen ein adaptives Verhalten, indem sie auf die unterschiedlichen Umstände reagieren und ihre Pläne sich entsprechend unterscheiden. Allerdings muss auch festgestellt werden, dass die geplanten Verhaltensweisen noch nicht ausgereift sind. Hier wäre es wünschenswert, wenn die formulierten Pläne längere Sequenzen von Aktionen enthalten würden, um ihren Weg zum Erreichen eines Ziels vorauszuplanen. Zudem ist zu erkennen, dass einige der verfügbaren Aktionen gar nicht oder kaum zum Einsatz kommen. Dies lässt darauf schließen, dass weiterer Feinschliff für die Modellierung und Umsetzung des Verhaltens durch GOAP notwendig ist. Analysiert man die einzelnen Durchläufe im Detail, so ist erkennbar, dass Unterschiede in der Ausführung bestimmter Aktionen auftreten. Zum Beispiel werden durch die Würfe andere Bereiche des Tors anvisiert oder verschiedene Passempfänger beim Mittelanwurf ausgewählt. Diese Erkenntnisse bedeuten, dass die Anforderung REQ-6 zwar erfüllt ist, jedoch weitere Verbesserungen in diesem Bereich vorzunehmen sind, um eine bessere Imitation des Handballsports zu erschaffen.

Betrachtet man weiterhin die Bewegungen der Agenten in der grafischen Oberfläche des Analyseprogramms, so ist durchaus zu erkennen, dass es sich um ein Handballspiel handelt. Die

Bewegungen der Spieler wirken nachvollziehbar. Wenn man allerdings betrachtet, wie viele Ticks die Agenten benötigen, um von der Mittellinie in die Nahwurfzone zu gelangen, so ist festzuhalten, dass dies länger als in der realen Welt dauert. Hier sind die Schrittgrößen für die Spieler anzupassen, um realistischere Sequenzen zu erhalten. Die Geschwindigkeit des Balls basiert auf Durchschnittsgeschwindigkeiten, die Würfe in der Handball-Bundesliga erreichen, In dem Aspekt ist jedoch eine Anpassung des Modells notwendig, da die Geschwindigkeit des Balls zu kleinschrittig reduziert wird, wenn die Höhe 0 ist und die Reibung des Bodens diesen schneller abbremsen sollte. Somit sind auch für die Anforderung REQ-2 Aspekte gegeben, die diese erfüllen, da die subjektive Wahrnehmung das Handballspiel erkennt und nachvollziehen kann. Allerdings sind auch hier weitere Verbesserungen vorzunehmen.

Um auf die Anforderung REQ-7 einzugehen, kann festgehalten werden, dass die Höhe in dem System durch 2.5-Dimensionalität umgesetzt wurde, selbst wenn das MARS-Framework nur 2-dimensionale Modelle für vergleichbare Multiagentensysteme vorsieht. Die Höhe ist ein relevanter Aspekt für Torwürfe durch die Angreifer, das Halten des Balles und auch, ob ein Pass fangbar ist oder ein Wurf über dem Tor im Aus landet. Somit erfüllt das System diese Anforderung. Hier könnte eine Darstellung der Höhe in dem Analyseprogramm helfen, den Anwendern diesen Aspekt nachvollziehbar zu machen.

Weiterhin ermöglicht das System es, Angriffssituationen des Handballsports zu simulieren und dabei weitgehende Konfigurationen vorzunehmen. Dies wird durch die variablen Ausgangssituationen der Experimente aufgezeigt. Neben der Konfiguration der Situationen ist es dem Anwender des Systems zudem möglich, Auftakthandlungen für die angreifende Mannschaft zu formulieren. Durch die Aufteilung in Phasen und die Möglichkeit, mehrere Optionen für einen einzelnen Spieler in derselben Phase zu definieren, ist eine gute Grundlage gegeben, dass Trainer detailliert mit dem Programm arbeiten können und dies so einen Mehrwert für ihre Arbeit schafft. In dem zweiten und dritten Experiment ist allerdings auch zu beobachten, dass mit dem aktuellen GOAP-Modell die Ausführung der übergebenen Anweisungen bestenfalls zweitrangige Priorität ist. Aus dem Prozess der Implementierung kann dahingegen berichtet werden, dass die Basis für die Ausführung der Auftakthandlungen funktionsfähig ist und die Abläufe in isoliertem Zustand wie vorgegeben ausgeführt werden können. Auch waren im Prozess der Entwicklung zwischenzeitlich Konfigurationen des GOAP-Modells vorhanden, in denen die

Auftakthandlungen vermehrt ausgeführt wurden. Dort waren jedoch kaum andere Aktionen in den formulierten Plänen zu finden. Ein wahrscheinlicher Grund für den Fokus auf den anderen Aktionen ist die Modellierung der Kosten für die einzelnen Aktionen, die das Ausführen der Aktion `MoveToDefensiveHole` gegenüber den zwei Aktionen `CallPlay` und `ExecutePlayInstruction` bevorzugt. Hier gilt es, eine goldene Mitte zu treffen, um Auftakthandlungen auszuführen und zudem adaptive Abweichungen davon zu sehen, wenn die wahrgenommene Situation dafür einen Grund bietet. Wenn dies gelingt, könnte REQ-4 vollständig als erfüllt betrachtet werden. REQ-1 hingegen wurde bereits erfüllt.

In Hinblick auf REQ-5 können aus den Experimenten nicht allzu viele Schlüsse gezogen werden. Die Möglichkeit zur Kommunikation wurde in das System eingebaut, welche insbesondere bei der Ausführung der Auftakthandlungen zum Einsatz kommt, um Absprachen zu treffen, `Plays` anzusagen und Konsens in der Mannschaft zu erreichen. Wie beschrieben ist in Experiment 2 erkennbar, dass diese Kommunikation auch funktioniert, da die Absprachen innerhalb der Auftakthandlungen getroffen werden. Somit kann diese Anforderung als erfüllt betrachtet werden.

In Hinsicht auf die nicht-funktionalen Anforderungen kann festgehalten werden, dass das System auf dem MARS-Framework aufbaut (REQ-10) und dessen Kapazitäten verwendet, um ein eine Multiagentensimulation für den Handballsport umzusetzen. Das System wurde mit dem Hintergedanken entworfen, dass es in Zukunft weiterentwickelt werden könnte, um beispielsweise nicht nur einzelne Situationen, sondern ganze Spiele fortlaufend zu simulieren. Bei der Entwicklung wurde daher auf Erweiterbarkeit und Austauschbarkeit geachtet, indem beispielsweise an vielen Stellen über Interfaces auf Objekte zugegriffen wird und die Agenten sich nicht gegenseitig kennen. Stattdessen können sie nur über Snapshots und Interfaces, die sie über die Layers wahrnehmen, miteinander interagieren. So werden Abhängigkeiten vermieden. REQ-9 kann somit auch als erfüllt betrachtet werden.

Bezogen auf die Nutzbarkeit des Systems für Trainer und Trainerinnen zeigt Experiment 3, dass die geschaffenen Möglichkeiten, Auftakthandlungen zu definieren, in der aktuellen Form noch nicht komplett intuitiv nutzbar sind, sondern teils weiterer Erklärungen bedürfen. Neben einigen Limitierungen in der Verständlichkeit bietet das Analyseprogramm jedoch einfache Konfigurationsfunktionen, welche gut für Anwender zu verwenden sind. Insbesondere die

Möglichkeit direkt auf die beobachteten Simulationsergebnisse in derselben Anwendung reagieren zu können sind hier ein positiver Aspekt für die Nutzbarkeit. Somit wäre es sinnvoll, die Nutzeroberfläche um detailliertere Nutzungshinweise zu erweitern sowie die Beschriftungen prägnanter zu formulieren. Hierbei kann die Implementierung mehrerer Sprachoptionen dabei helfen, das Verständnis zu erleichtern. Auch das Einbauen von visuellen Hinweisen könnte helfen, das Verhalten auf natürliche Art zu leiten. Dies könnte beispielsweise erreicht werden, indem bei dem Formulieren der Anweisungen nur die Felder klickbar gemacht werden, welche in dem gegebenen Kontext Pflichtfelder sind. Die weiteren Felder sollten deaktiviert und farblich unauffälliger gestaltet werden. Die Möglichkeit, Auftakthandlungen nach Betrachtung der Ergebnisse auf leichte Weise zu erweitern, bietet den Anwendern jedoch einen komfortablen Arbeitsablauf und schafft die Grundlage dafür, das System nicht bloß als Visualisierungshilfe, sondern auch als Hilfsmittel zur Auswertung taktischer Mittel zu verwenden.

Wie bereits erörtert wird die Ausführung der Auftakthandlungen durch das GOAP-Modell nicht ausreichend priorisiert und somit die gegebenen Anweisungen zu selten ausgeführt. Dies schränkt die Nutzbarkeit des Systems für Trainer ein, da sie sich nicht darauf verlassen können, dass ihre Eingaben berücksichtigt werden. Hier wäre es denkbar die Dynamik des GOAP-Modells zu reduzieren, indem eine Anpassung dahingehend vorgenommen wird, dass die Anweisungen priorisiert ausgeführt werden. Diese Reduktion der Verhaltensdynamik würde die Eigenschaften als Multiagentensystem verschlechtern, dafür wäre auf diese Weise die Anforderung REQ-4 besser erfüllt. Insgesamt ist basierend auf den in Experiment 2 und 3 betrachteten Ergebnisse zu sagen, dass das System durch die fehlende Priorisierung der Auftakthandlungen aktuell nicht verwendet werden kann, um Erkenntnisse für die Spielstrategie zu ziehen. Zudem kann es nicht zum Einsatz kommen, um Auftakthandlungen zu vermitteln. So bietet das System Trainern aktuell keine Verbesserung gegenüber Methoden wie der Taktiktafel und kann diese auch nicht ersetzen. Wie beschrieben ist weitere Entwicklungsarbeit notwendig, um die Simulationsergebnisse dahingehend zu verbessern und in Zukunft den gewünschten Mehrwert für Anwender und Anwenderinnen zu schaffen.

8 Schlussfolgerungen und Ausblick

Zusammenfassend lässt sich schließen, dass das System die Anforderungen, die aufgestellt wurden, nicht vollständig erfüllt sind. Durch die beschriebenen Limitierungen, insbesondere in der Modellierung des Agentenverhalten durch GOAP, bietet das System sich nicht als Alternative zu gängigen Methoden wie der Taktiktafel an. Allerdings sind Ansätze erkennbar und Vorkehrungen getroffen worden, sodass sämtliche Anforderungen zumindest teilweise erfüllt wurden. Es ist in dem aktuellen Zustand kein flüssiges Handballspiel gewährleistet und die Anweisungen der Trainer werden mit der jetzigen GOAP-Umsetzung nicht in der gewünschten Frequenz ausgeführt. Wie beschrieben, war im Laufe der Implementierung jedoch schon erkennbar, dass die notwendigen Grundlagen für die Ausführung der Auftakthandlungen geschaffen wurden. Die Arbeit an diesem Umstand sollte in weiterführenden Arbeiten fortgeführt werden, um die Erfüllung der Anforderungen ganzheitlich zu erreichen. Das besondere Potenzial des Systems liegt darin, dass Trainer und Trainerinnen Auftakthandlungen auf einfach Weise testen können. Durch die Integration der Bearbeitungsmöglichkeiten der Auftakthandlungen in das Analyseprogramm ist eine schnelle Reaktion auf betrachtete Ergebnisse möglich. Insbesondere die Möglichkeit die Simulation direkt aus der Nutzeroberfläche ausführen zu können würde diesen Optimierungskreislauf noch praktischer machen. Jedoch bleibt die Nutzbarkeit des Systems auch bei einer solchen Erweiterung weiterhin davon abhängig, ob ein Durchbruch in dem Agentenverhalten erzielt werden kann.

Die Verfehlung einiger Aspekte können womöglich darauf zurückgeführt werden, dass in der Modellierung zu umständlich gedacht wurde und stets angestrebt wurde, möglichst nah an der Realität des Handballs zu arbeiten. Auch die Betrachtung der Referenzsysteme als Grundlage für die Modellierung des im Rahmen dieser Arbeit erstellten Systems könnte dazu beigetragen haben, da diese Systeme als primären Zweck den Einsatz für die Forschung mit Multiagentensystemen oder Lernalgorithmen haben. Eine geeignetere Abgrenzung zwischen dem realen

Modell und dem in dieser Arbeit umgesetzten konzeptionellen Modell hätte hier zu besseren Ergebnissen führen können. Dasselbe gilt für die Modellierung des Verhaltens durch GOAP, da hier einige Aktionen gar nicht in den Plänen formuliert werden. Dies ist womöglich in der Größe des Zustandsraums begründet, in dem die Übersicht beim Modellieren verloren gehen kann. Im Vergleich zu anderen Umsetzungen von GOAP, wie das von Lenfers et al. (2018) beschriebene GOAP-Modell ist allerdings auch zu erwähnen, dass die Komplexität in dem Handballspiel durch die komplett freie Bewegungsmöglichkeit höher ist. Zudem ist es in dem hier beschriebenen Modell essenziell, dass die Agenten bei ihrem Verhalten die Sensorinformationen über die anderen Agenten in der Umgebung einbeziehen, während die Agenten in anderen betrachteten GOAP-Systemen einen größeren Fokus auf die eigenen Umstände legen können und andere Agenten nicht beachten.

Für weitere Arbeit an dem System könnte es zudem sinnvoll sein auszuprobieren, ob eine Kombination von GOAP und Lernalgorithmen zu einem besseren Ergebnis führen kann. Hier wäre es zum Beispiel möglich, über den Einsatz von Lernen die geeigneten Relevanz-Werte für die GOAP-Ziele der Agenten zu finden. Des Weiteren wäre es möglich, zur Steuerung des Verhaltens in diesem System hierarchisches Lernen anstelle von GOAP einzusetzen. Dies scheint in der Referenzliteratur im Kleinen Erfolge erzielt zu haben und könnte daher auch hier zu einer Verbesserung des Systems führen.

Wie in der Arbeit erwähnt, wäre eine mögliche Erweiterung für das System in der Zukunft, die Begrenzung auf einzelne, in sich geschlossene Situationen aufzubrechen und ein ganzes Spiel simulieren zu lassen. Zudem könnte in Zukunft der `RefereeLayer` in einen Agenten umgewandelt werden, sodass auch der Schiedsrichter ein Verhalten modellieren muss, um das Spiel bestmöglich wahrnehmen zu können. So könnten zudem Fehlentscheidungen in den Simulationen auftreten, was den Grad des Realismus erhöhen würde. Dies ist jedoch nur gemeinsam mit der Erweiterung auf ganze Spiele sinnvoll.

Zu guter Letzt wäre das Ergänzen eines Trainer-Agenten denkbar, um taktische Anweisungen an die Mannschaften zu geben, die auf den Beobachtungen des Spiels und insbesondere der gegnerischen Mannschaft beruhen. Diese letzten beiden Erweiterungen wären insbesondere anzustreben, wenn ein Einsatz des Systems in der Forschung erwünscht ist.

Literaturverzeichnis

Abdollah Amirkhani & Amir Hossein Barshooi. (2021, November 17). *Consensus in multi-agent systems: A review*. Artificial Intelligence Review (2022) 55:3897–3935. https://doi.org/10.1007/s10462-021-10097-x

Adams, M., David, A., Hesse, M., & Rückert, U. (2023). Expected Goals Prediction in Professional Handball using Synchronized Event and Positional Data. *Proceedings of the 6th International Workshop on Multimedia Content Analysis in Sports*, 83–91. https://doi.org/10.1145/3606038.3616152

Akiyama, H., Aramaki, S., & Nakashima, T. (2012). Online Cooperative Behavior Planning Using a Tree Search Method in the RoboCup Soccer Simulation. *2012 Fourth International Conference on Intelligent Networking and Collaborative Systems*, 170–177. https://doi.org/10.1109/iNCoS.2012.83

Caedmon Somers, Jason Rupert, Yunqi Zhao, Igor Borovikov, Jiachen Yang, Ahmad Beirami. (2020, Februar). *Simple Team Sports Simulator (STS2)* [Github Repository]. https://github.com/electronicarts/SimpleTeamSportsSimulator

Clemen, T., Ahmady-Moghaddam, N., Glake, D., Lenfers, U. A., Ocker, F., Osterholz, D., & Strobele, J. (2022). Toward A Movement Paradigm For Artificial Human Agents. *2022*

Annual Modeling and Simulation Conference (ANNSIM), 176–187. https://doi.org/10.23919/ANNSIM55834.2022.9859271

Clemen, T., Lenfers, U. A., Dybulla, J., Ferreira, S. M., Kiker, G. A., Martens, C., & Scheiter, S. (2021). A cross-scale modeling framework for decision support on elephant management in Kruger National Park, South Africa. *Ecological Informatics*, *62*, 101266. https://doi.org/10.1016/j.ecoinf.2021.101266

Hu, B., Liu, J., & Jin, X. (2005). Multiagent RoboNBA Simulation: From Local Behaviors to Global Characteristics. *SIMULATION*, *81*(7), 465–485. https://doi.org/10.1177/0037549705058425

International Handball Federation. (2024). *IX. Spielregeln für Hallenhandball* [Regelwerk].

Julian Rux. (2023, Dezember 14). ÜberZahl – Die Zahlenkolumne: Andy Schmids Wunsch zur Angriffsdauer des SC Magdeburg. *ÜberZahl - Die Zahlenkolumne*. https://www.liquimoly-hbl.de/de/n/saison-23-24/lm-hbl/ueberzahl/ueberzahl---die-zahlenkolumne--andy-schmids-wunsch-zur-angriffsdauer-des-sc-magdeburg----news/

Julian Rux. (2024, Juni 6). ÜberZahl – Die Zahlenkolumne: SCM mit Rekorden auf allen Ebenen zur Meisterschaft. *ÜberZahl - Die Zahlenkolumne*. https://www.liquimoly-hbl.de/de/n/saison-23-24/lm-hbl/ueberzahl/ueberzahl---die-zahlenkolumne--scm-mit-rekorden-auf-allen-ebenen-zur-meisterschaft---news/

Kolodziej, C. (2013). *Erfolgreich Handball spielen: Technik, Taktik, Training* (4., neu bearb. Aufl. (Neuausgabe)). BLV Buchverlag.

Lenfers, U. A., Ahmady-Moghaddam, N., Glake, D., Ocker, F., Ströbele, J., & Clemen, T. (2021). Incorporating Multi-Modal Travel Planning into an Agent-Based Model: A Case Study at the Train Station Kellinghusenstraße in Hamburg. *Land*, *10*(11), 1179. https://doi.org/10.3390/land10111179

Lenfers, U. A., Weyl, J., & Clemen, T. (2018). Firewood Collection in South Africa: Adaptive Behavior in Social-Ecological Models. *Land*, *7*(3), 97. https://doi.org/10.3390/land7030097

Liu, S., Lever, G., Wang, Z., Merel, J., Eslami, S. M. A., Hennes, D., Czarnecki, W. M., Tassa, Y., Omidshafiei, S., Abdolmaleki, A., Siegel, N. Y., Hasenclever, L., Marris, L., Tunyasuvunakool, S., Song, H. F., Wulfmeier, M., Muller, P., Haarnoja, T., Tracey, B. D., … Heess, N. (2021). *From Motor Control to Team Play in Simulated Humanoid Football* (arXiv:2105.12196). arXiv. http://arxiv.org/abs/2105.12196

MARS Group. (o. J.-a). *Agent* [Documentation]. Agent. Abgerufen 30. März 2024, von https://www.mars-group.org/docs/tutorial/development/agent

MARS Group. (o. J.-b). *Concepts* [Documentation]. Concepts. Abgerufen 30. März 2024, von https://www.mars-group.org/docs/tutorial/mars-basics/

MARS Group. (o. J.-c). *Entity* [Documentation]. Entity. Abgerufen 30. März 2024, von https://www.mars-group.org/docs/tutorial/development/entity

MARS Group. (o. J.-d). *Environments* [Documentation]. Environments. Abgerufen 30. März 2024, von https://www.mars-group.org/docs/tutorial/development/environments/

MARS Group. (o. J.-e). *Layers* [Documentation]. Layers. Abgerufen 30. März 2024, von https://www.mars-group.org/docs/tutorial/development/layers

Orkin, Jeff. (2008). *Applying Goal-Oriented Action Planning to Games*.

Prokopenko, M., & Wang, P. (2016). *Disruptive innovations in RoboCup 2D Soccer Simulation League: From Cyberoos'98 to Gliders2016* (arXiv:1612.00947). arXiv. http://arxiv.org/abs/1612.00947

Saeed Rahimi, Antoni B. Moore, & Peter A. Whigham. (2022). *A vector-agent approach to (spatiotemporal) movement modelling and reasoning*. nature - scientific reports??? https://doi.org/10.1038/s41598-022-22056-9

The RoboCup Soccer Simulator Maintenance Committee. (2024, Juli 14). *The RoboCup Soccer Simulator Users Manual*. https://rcsoccersim.readthedocs.io/en/latest/index.html

Tolk, A., Clemen, T., Gilbert, N., & Macal, C. M. (2022). How Can We Provide Better Simulation-Based Policy Support? *2022 Annual Modeling and Simulation Conference (ANNSIM)*, 188–198. https://doi.org/10.23919/ANNSIM55834.2022.9859512

Yang, J., Borovikov, I., & Zha, H. (2020). *Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery* (arXiv:1912.03558). arXiv. http://arxiv.org/abs/1912.03558

Zhao, Y., Borovikov, I., Rupert, J., Somers, C., & Beirami, A. (2019). *On Multi-Agent Learning in Team Sports Games* (arXiv:1906.10124). arXiv. http://arxiv.org/abs/1906.10124

A Anwendungsfälle

Hier werden die Anwendungsfälle des Systems dargestellt. Diese sind eine Ergänzung für die funktionalen Anforderungen an das zu entwerfende Programm. Die Anwendungsfälle werden aus Sicht eines Spieler-Agenten innerhalb des Systems betrachtet.

Name	UC-1
Beschreibung	Der Spieler kann Spieler und den Ball sehen, wenn diese sich innerhalb des Blickfelds des Spielers befinden.
Akteure	Spieler
Auslöser	Der Spieler sendet eine Anfrage für sein aktuelles Blickfeld.
Vorbedingungen	–
Nachbedingungen	Der Akteur hat aktuelle Informationen über die Akteure, die für ihn sichtbar sind.
Standardablauf	1. Die Blickrichtung des Spielers wird geprüft und anhand dessen berechnet, in welchem Raum andere Spieler und der Ball gesehen werden können. 2. Es wird geprüft, ob die Positionen der anderen Spieler und des Balls innerhalb dieses Raumes liegen. 3. Die sichtbaren Spieler und der Ball werden zurückgegeben. 4. Der internen Zustände in Bezug auf die anderen Spieler werden aktualisiert.

Tabelle 10: Anwendungsfall „Sehen anderer Spieler und des Balls“.

Name	UC-2
Beschreibung	Der Akteur macht einen Schritt von einer bestimmten Art und in eine bestimmte Richtung.
Akteure	Spieler
Auslöser	Der Spieler entscheidet sich, sich bewegen zu wollen.
Vorbedingungen	• Der Spieler ist in die (grobe) Richtung ausgerichtet, in die er einen Schritt machen möchte • Der Spieler hat im aktuellen Simulationsschritt noch keinen Schritt gemacht. • Der Spieler hat nicht unmittelbar zuvor einen Schritt getätigt, der die gewählte Schrittart verhindert. • Der Spieler befindet sich nicht in der Luft.
Nachbedingungen	Der Spieler befindet sich an einer neuen Position entsprechend der Schrittrichtung und -art.
Standardablauf	1. Es wird geprüft, ob die Vorbedingungen den Schritt erlauben. 2. Das System berechnet die Schrittgröße für den Spieler basierend auf dem gegebenem Schritttyp. 3. Das System prüft, ob bei der Bewegung eine Kollision auftritt und wie mit dieser umzugehen ist. 4. Das System bewegt den Agenten an die berechnete Endposition.

Tabelle 11: Anwendungsfall „Einen Schritt machen“.

Name	UC-3
Beschreibung	Der Spieler wirft den Ball in eine bestimmte Richtung, bei einem bestimmten Austrittswinkel (oben/unten) und mit einer bestimmten Kraft.
Akteure	Spieler
Auslöser	Der Spieler entscheiden sich den Ball zu werfen.
Vorbedingungen	• Der Spieler ist in Ballbesitz
Nachbedingungen	• Der Spieler ist nicht in Ballbesitz • Der Ball hat eine Richtung und eine Geschwindigkeit
Standardablauf	1. Es wird geprüft, dass der Spieler in Ballbesitz ist. 2. Die Geschwindigkeit für die Art des Wurfes wird berechnet. 3. Das System prüft, ob der Ball auf dem Weg mit etwas kollidiert und berechnet gegebenenfalls die Änderungen und weitere Bewegung durch das Abprallen. 4. Das System bewegt den Ball an die berechnete Position.

Tabelle 12: Anwendungsfall „Den Ball werfen“.

Name	UC-4
Beschreibung	Der Spieler möchte den Ball fangen, wenn er in der Luft in seiner Nähe ist.
Akteure	Spieler
Auslöser	Der Ball fliegt zu dem Spieler.
Vorbedingungen	• Der Ball ist nicht in Besitz eines Spielers. • Der Ball ist in der Luft. • Der Ball befindet sich in der Nähe des Spielers. • Der Spieler kann den Ball sehen.
Nachbedingungen	• Der Spieler ist in Ballbesitz. • Der Ball verharrt an der Position des Spielers.
Standardablauf	1. Es wird geprüft, dass der Ball in der fangbaren Nähe in der Luft ist und der Spieler der Ball sieht. 2. Der Fänger fängt den Ball und wird als Spieler in Ballbesitz gesetzt. 3. Die weitere Bewegung des Balls wird gestoppt. 4. Die Position des Balls wird gleich der Position des Fängers gesetzt.

Tabelle 13: Anwendungsfall „Den Ball fangen“.

Name	UC-5
Beschreibung	Der Spieler nimmt den Ball auf, wenn dieser sich in der Nähe befinden und auf dem Boden, oder nah über dem Boden, ist.
Akteure	Spieler
Auslöser	Der Ball befindet sich in der Nähe des Spielers.
Vorbedingungen	• Der Ball befindet sich unterhalb der fangbaren Höhe. • Der Ball befindet sich in der erreichbaren Nähe des Spielers. • Der Ball ist nicht in Besitz eines Spielers. • Der Spieler sieht den Ball.
Nachbedingungen	• Der Spieler ist in Ballbesitz. • Der Ball verharrt an der Position des Spielers.
Standardablauf	1. Es wird geprüft, dass der Ball in der fangbaren Nähe in der Luft ist und der Spieler den Ball sieht. 2. Der Spieler nimmt den Ball auf und wird als der Spieler in Ballbesitz vermerkt. 3. Die weitere Bewegung des Balls wird gestoppt. 4. Die Position des Balls wird gleich der Position des Fängers gesetzt.

Tabelle 14: Anwendungsfall „Den Ball aufnehmen“.

Name	UC-6
Beschreibung	Der Spieler springt hoch.
Akteure	Spieler
Auslöser	Ein Sprung verschafft eine bessere Position zum Wurf, oder ermöglicht das Blocken eines Wurfs.
Vorbedingungen	• Der Spieler springt nicht bereits.
Nachbedingungen	• Der Spieler befindet sich in der Luft.
Standardablauf	1. Der Spieler wird in die erste Sprungphase gesetzt. 2. Jeden folgenden Simulationsschritt wird die nächste Sprungphase gesetzt, bis wieder gelandet wurde.

Tabelle 15: Anwendungsfall "Springen"

Name	UC-7
Beschreibung	Der Spieler dreht seinen Körper um einen gegebenen Winkel.
Akteure	Spieler
Auslöser	Eine Drehung des Körpers ist notwendig, um einen Schritt zu machen, sich zum Ball oder für einen Wurf auszurichten.
Vorbedingungen	• Der Agent ist in eine Richtung ausgerichtet, von der aus er die Zielrichtung erreichen kann.
Nachbedingungen	• Der Spieler ist in die gewünschte Richtung ausgerichtet.
Standardablauf	1. Es wird geprüft, ob eine Drehung um den gewünschten relativen Winkel möglich ist. 2. Der Körper des Agenten wird um den gewünschten Winkel, oder den Maximalwinkel gedreht, sollte dieser überstiegen sein.

Tabelle 16: Anwendungsfall „Den Körper drehen“.

Name	UC-8
Beschreibung	Der Spieler dreht seinen Kopf um einen gegebenen Winkel.
Akteure	Spieler
Auslöser	Eine Drehung des Körpers ist notwendig, um einen Schritt zu machen, sich zum Ball oder für einen Wurf auszurichten.
Vorbedingungen	• Der Kopf des Agenten ist in eine Richtung ausgerichtet, von der aus er die Zielrichtung erreichen kann.
Nachbedingungen	• Der Spieler blickt in die gewünschte Richtung.
Standardablauf	1. Es wird geprüft, ob eine Drehung um den gewünschten relativen Winkel möglich ist. 2. Der Kopf des Agenten wird um den gewünschten Winkel, oder den Maximalwinkel gedreht, sollte dieser überstiegen sein.

Tabelle 17: Anwendungsfall „Den Kopf drehen“.

Name	UC-9
Beschreibung	Der Spieler (Spielmacher) sagt seinen Mitspielern an, welche Auftakthandlung ausgeführt werden soll.
Akteure	Spieler (Spielmacher)
Auslöser	Der Spielmacher entscheidet, dass nun eine Auftakthandlung folgen soll.
Vorbedingungen	• Der Spieler ist der Spielmacher.
Nachbedingungen	• Die Spieler der angreifenden Mannschaft wurden über die Auftakthandlung informiert.
Standardablauf	1. Auswählen des Spielzugs durch den Spielmacher. 2. Der Spielmacher sendet eine Nachricht, von der Art einer Auftakthandlung-Ansage an alle Agenten. 3. Die angesagte Auftakthandlung wird sich als die aktuelle gemerkt.

Tabelle 18: Anwendungsfall "Eine Auftakthandlung ansagen".

Name	UC-10
Beschreibung	Die Spieler (≠ Spielmacher) erhalten eine Ansage des aktuellen Spielmachers.
Akteure	Spieler
Auslöser	Der Spielmacher informiert über Start einer Auftakthandlung.
Vorbedingungen	• Eine Nachricht mit Ansage einer Auftakthandlung wurde erhalten.
Nachbedingungen	• Der Spieler weiß, dass nun eine bestimmte Auftakthandlung ausgeführt werden soll.
Standardablauf	1. Die Ansage wird durch den Spieler verarbeitet. 2. Die angesagte Auftakthandlung wird sich als die aktuelle gemerkt.

Tabelle 19: Anwendungsfall „Ansage über Auftakthandlung erhalten“.

Name	UC-11
Beschreibung	Der Akteur informiert andere Spieler über sein aktuelles Wissen über einen anderen Spieler.
Akteure	Spieler
Auslöser	Der Spieler erhält neue Informationen über einen Spieler, die eine akute Wichtigkeit für mindestens einen seiner Mitspieler haben.
Vorbedingungen	–
Nachbedingungen	• Eine Nachricht wurde an die anderen Spieler versandt.
Standardablauf	1. Der Spieler entscheidet, dass die neuen Informationen über einen Spieler wichtig für einen Mitspieler sind. 2. Der Spieler sendet eine Nachricht mit den aktuellen Kenntnissen an die anderen Spieler.

Tabelle 20: Anwendungsfall "Mitspieler über einen Spieler informieren".

Name	UC-12
Beschreibung	Der Spieler verarbeitet eine Nachricht von einem seiner Mitspieler.
Akteure	Spieler
Auslöser	Der Spieler hat eine Nachricht mit Informationen über einen anderen Spieler erhalten.
Vorbedingungen	• Eine Nachricht von einem Mitspieler ist eingegangen.
Nachbedingungen	• Die erhaltenen Informationen sind in dem internen Zustand gespeichert.
Standardablauf	1. Die erhaltene Nachricht wird auf ihre Art geprüft. 2. Die Informationen über den anderen Spieler werden gelesen. 3. Die Informationen werden Eintragen, wenn diese aktueller sind als die durch eigene visuelle Wahrnehmung.

Tabelle 21: Anwendungsfall "Informationen eines Mitspielers über andere Spieler erhalten".

Name	UC-13
Beschreibung	Der Schiedsrichter bewertet den aktuellen Zustand des Umfelds und prüft, ob der Ball im Tor, im Aus oder anderweitig nicht mehr spielbar ist.
Akteure	Spieler
Auslöser	Ein Simulationsschritt wurde abgeschlossen und ein neuer vorbereitet.
Vorbedingungen	–
Nachbedingungen	• Die folgende Spielsituation wurde hergestellt, wenn nötig.
Standardablauf	1. Der Schiedsrichter prüft, ob die Position des Balles außerhalb des Spielfelds, oder in dem Tor liegt. 2. Der Schiedsrichter prüft ob andere Gründe für das Herstellen einer neuen Spielsituation bestehen. 3. Wenn ja, Auslösen von UC-14.

Tabelle 22: Anwendungsfall „Erkennen einer beendeten Spielsituation“.

Name	UC-14
Beschreibung	Der Schiedsrichter stellt nach einer Unterbrechung die neue Spielsituation her.
Akteure	Spieler
Auslöser	Die Prüfung aus UC-13 hat ergeben, dass der Ball wieder in das Spiel gebracht werden, oder eine Situation unterbunden werden muss.
Vorbedingungen	• UC-13 wurde positiv ausgewertet.
Nachbedingungen	• Die neue Spielsituation wurde hergestellt.
Standardablauf	1. Es wird für jeden Spieler die jeweilige Position für die neue Spielsituation berechnet. 2. Der Spieler wird an dieser Position platziert. 3. Der als Ballführer spezifizierten Spieler wird mit dem Ball ausgestattet und der Ball ebenfalls an dieser Position platziert.

Tabelle 23: Anwendungsfall "Herstellen einer Spielsituation".

B Dateien des Systems

In diesem Anhang sind spezifische Dateien des Systems angeführt, die für die Konfiguration und Ausführung essenziell sind.

B.1 Program.cs-Datei

Bei dem hier dargestellten Quelltext handelt es sich um den Inhalt er *Program.cs*-Datei. Der Code erzeugt ein `ModelDescription`-Objekt, über das die Layers und Agenten für die Simulation spezifiziert werden. Das `SimulationStarter`-Objekt erhält dieses Objekt sowie ein `SimulationConfiguration`-Objekt (welches aus der *config.json*-Datei ausgelesen wird), um die Simulation auszuführen.

```
/* Die using-Direktive werden aus Platzgründen ausgespart */

namespace HandballMAS;

internal static class Program
{
    private static void Main()
    {
        // create the model description.
        var description = new ModelDescription();

        // add the layers.
        description.AddLayer<SystemSettingsLayer>();
        description.AddLayer<PlaysLayer>();
        description.AddLayer<DefensiveFormationLayer>();
        description.AddLayer<PlayerMindLayer>();
        description.AddLayer<PitchLayer>();
        description.AddLayer<RefereeLayer>();

        // add the entity-like agents.
        description.AddAgent<PlayerBody, PitchLayer>();
        description.AddAgent<Ball, PitchLayer>();
```

```
        // add the acting agents.
        description.AddAgent<PlanningAttackingPlayerMind, Player-
MindLayer>();
        description.AddAgent<PlanningDefendingPlayerMind, Player-
MindLayer>();
        description.AddAgent<PlanningGoalkeeperPlayerMind, Player-
MindLayer>();

        // specify JSON configuration file here
        var file = File.ReadAllText("config.json");
        var config = SimulationConfig.Deserialize(file);

        var starter = SimulationStarter.Start(description, config);
        var handle = starter.Run();
        Logger.WritePersistentLogs();
        Logger.WritePlanLogs();
        Logger.WriteRefereeLogs();
        Logger.WriteBlockHistory();
        Logger.Log($"Successfully executed iterations: {handle.Itera-
tions}");
        starter.Dispose();
    }
}
```

Listing 1: Die Inhalte der *Program.cs*-Datei.

B.2 config.json-Datei

Hier wird die Konfigurationsdatei *config.json* bereitgestellt, welche die Konfigurationsdaten für das System enthält. Sie definiert, wie viele Ticks in der Simulation ausgeführt werden sollen und spezifiziert Initialisierungsdaten und Konfigurationsdateien für bestimmte Layers und Agententypen der Simulation.

```
{
  "globals": {
    "steps": 160,
    "output": "csv",
```

```
    "options": {
      "delimiter": ";",
      "numberFormat": "G",
      "culture": "en-EN"
    }
  },
  "layers": [
    {
      "name": "PitchLayer",
      "file": "Resources/HandballPitch_10x10.csv",
      "dimensionx": 440,
      "dimensiony": 240
    },
    {
      "name": "PlaysLayer",
      "file": "Resources/plays.json"
    },
    {
      "name": "DefensiveFormationLayer",
      "file": "Resources/defensive_formation.csv"
    },
    {
      "name": "SystemSettingsLayer",
      "file": "Resources/system_settings.json"
    }

  ],
  "agents": [
    {
      "name": "PlayerBody",
      "count": 14,
      "file": "Resources/player_config.csv"
    },
    {
      "name": "Ball",
      "count": 1,
      "file": "Resources/ball_config.csv"
    }
```

```
  ]
}
```

Listing 2: Die Inhalte der *config.json*-Datei.

B.3 plays.json-Datei

Hier wird eine beispielhafte *plays.json*-Datei dargestellt. Diese Datei dient als Schnittstelle um Anweisungen für eine oder mehrere Auftakthandlungen durch einen Anwender in das System eingeben zu können. Diese werden durch die `PlaysLayer` deserialisiert und so den `PlayerMind`-Agenten verfügbar gemacht. Die hier zu sehende Datei enthält die Beschreibung der Auftakthandlung, die für das Experiment 2 genutzt wurde.

```
{
  "Play": {
    "Name": "Play",
    "_phases": {
      "1": [
        {
          "Role": "LP",
          "StartSignal": "None",
          "InstructionOptions": [
            {
              "Type": "Block",
              "Instruction": ["True"],
              "DefensivePosition": "Hole4"
            }
          ]
        }
      ],
      "2": [
        {
          "Role": "RB",
          "StartSignal": "None",
          "InstructionOptions": [
            {
              "Type": "CrossWith",
              "Instruction": [
```

```
                "CB"
              ],
              "DefensivePosition": "Hole4"
            }
          ]
        },
        {
          "Role": "CB",
          "StartSignal": "None",
          "InstructionOptions": [
            {
              "Type": "CrossFor",
              "Instruction": ["RB"],
              "DefensivePosition": "Hole4"
            },
            {
              "Type": "Pass",
              "Instruction": ["LB"],
              "DefensivePosition": "None"
            }
          ]
        }
      ],
      "3": [
        {
          "Role": "RB",
          "StartSignal": "None",
          "InstructionOptions": [
            {
              "Type": "Throw",
              "Instruction": [],
              "DefensivePosition": "Hole4"
            }
          ]
        },
        {
          "Role": "LB",
          "StartSignal": "None",
```

```
          "InstructionOptions": [
            {
              "Type": "Throw",
              "Instruction": [],
              "DefensivePosition": "Hole2"
            }
          ]
        }
      ]
    }
  }
}
```

Listing 3: Beispielhafte Inhalte einer *plays.json*-Datei.

C GOAP-Aktionen der Angreifer

Hier wird in Tabelle 24 die gesamte Menge der GOAP-Aktionen aufgeführt, die einem `PlanningAttackingPlayerMind`-Agenten zur Verfügung stehen. Für jede dieser Aktionen sind zudem die Vorbedingungen und Effekte aufgelistet.

Tabelle 24: Die GOAP-Aktionen der `PlanningAttackingPlayerMind`-Agenten mit den Vorbedingungen und Nachbedingungen.

Aktion	***Vorbedingungen***	***Effekte***
BlockFor	HasBall == False IsCloseToDefence == True CanBlock == True	HasTeammateScoringOpportunity == True IsTeammateOpen == True
CallPlay	IsPlayCalled == False IsPlaymaker == True KnowsPlays == True	IsPlayCalled == True HasInstructionInPlay == True
ExecutePlay-Instruction	KnowsPlays == True IsPlayCalled == True HasInstructionInPlay == True IsGameRunning == True	IsTeamScoringAGoal == True HasTeammateScoringOpportunity == True HasScoringOpportunity == True
GetOpen	HasBall == False	IsOpen == True IsPosingThreatToDefence == True
Jump	HasBall == True IsInThrowingRange == True IsGameRunning == True IsJumping == False	HasScoringOpportunity == True IsJumping == True IsTalleThanDefender == True

LookForBall	*HasBall == True*	*IsSeeingBall == True* *IsGameRunning == True*
LookForPlayer	IsGameRunning == True	HasCurrentInfoOnOutfieders == True IsSeeingOpenTeammate == True
MoveTowards-DefensiveHole	IsGameRunning == True	IsCloseToDefence == True HasScoringOpportunity == True IsPosingThreatToDefence == True HasBeelineToGoal == True HasHighXGoals == True IsInThrowingRange == True
Pass	HasBall == True IsTeammateOpen == True IsSeeingOpenTeammate == True	HasBall == False IsGameRunning == True
PickUpBall	IsSeeingBall == True IsBallFree == True IsClosestTeammateToBall == True	IsBallFree == False HasBall == True
Throw	HasBall == True HasScoringOpportunity == True IsFacingGoal == True IsInThrowingRange == True	HasBall = False IsTeamScoringAGoal == True
WaitToThrow	HasBall == True IsGameRunning == True IsJumping == True	HasScoringOpportunity == True

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original