Master's Thesis

Decision Making in a Multi-Agent System - Planning versus Learning

Monika Schubert

Graz, January 2008

Supervisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa
Evaluator: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa

Institute for Softwaretechnology
Graz, University of Technology

Abstract

Within the research field of autonomous agents, decision making is a central topic. In this thesis two different approaches - one by planning and an alternative one via learning - are described in their application. While the current implementation of the planning system is based on a Prolog backend, the learning approach is implemented with a reinforcement Q-learning algorithm. The environment of the application is the system of *KickOff-TUG*, the RoboCup 2D Soccer Simulation League team of Graz, University of Technology.

Within this work both approaches are compared and evaluated to discover their strengths and weaknesses. It is shown that the main advantage of the planning system is that the plans are built and reviewed by humans. Unfortunately the dependency on the human is the biggest problem of this system as well. On the other hand the learning system is more adaptable, but its main disadvantage lies in the specification of the target function and the creation of training examples.

This work shows that both algorithms, planning and learning, can be applied to the decision making process, as they are equally appropriate.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a certain environment agents have to cooperate in order to reach a certain goal. The ways how agents can cooperate are different. For example one agent can be charged with the responsibility to coordinate the other agents, and the other agents just do whatever the planning agent tells them to do. This is called a centralised way of cooperation. The centralised cooperation is quite easy to implement but the disadvantage is that if this central agent fails, everything fails.

Another possibility to implement coordination is to give every agent a single decision making instance to plan its next action. This has the advantage, that the common goal does not depend on one single agent. But the problem is that the agents have to have a possibility to communicate with each other. This communication can be verbal with a protocol that is transmitted between the agents or non-verbal. In a non-verbal communication the agents have to interpret the actions of the other agents. The finding of the next action or actions is called *decision making*. This *decision making* can be implemented in different ways for example through planning or learning.

1.1 Goal

The goal of this work is to find out what way of decision making is better suited for an agent - planning or learning. Therefore, a highly dynamic environment, the RoboCup Soccer Simulation League, is chosen. In this environment it is not possible to implement a centralised decision making process because the communication is limited. The RoboCup including the different leagues, especially the RoboCup 2D Simulation League and the team *KickOffTUG* are described in Chapter 2.

The system of the RoboCup 2D Soccer Simulation League team *KickOff-TUG* has currently a decision making based on a planning algorithm. The basic knowledge about logic, logical closuring, planning and the current implementation of the planning system are described in Chapter 3. The system, which should be compared with the planning one is based on a learning algorithm. This learning approach is implemented during the master's thesis course. To have a basic for the implementation, the different learning methods and critical design issues are outlined in Chapter 4.

In Chapter 5 the different analysing methods and their outcomes are described. These methods include a comparison of the different systems through games and special situations. Furthermore the systems are compared according to their types and finally the different approaches to influencing factors are evaluated. This analysis should show which system is better suited for the decision making process.

1.2 Multi-Agent Systems

The term *agent* was already used, but what exactly does it mean? Woolridge defines an agent as:

> *"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." ([Woo02], p. 15)*

This means an agent can perform some action in an environment and furthermore every agent has a goal. In this context the term *agent* is used as an instance of a software program. The environment is the RoboCup Soccer Simulation League, for a detailed description about the environment and the actions that can be performed take a look at Section 2.3. The objective of a soccer playing agent is obviously scoring goals to win the game.

It is evident that the environment and the agent have to have some kind of interaction, as Figure 1.1 shows. For this interaction the agent has some sensors to observe the environment and some actuators to perform actions.

As already pointed out there usually is more than one single agent in an environment. An environment where more than one agent is acting is also called a multi-agent system. A clearer definition is given by Woolridge:

> *"The system contains a number of agents, which interact with one another through communication. The agents are able to act in an environment; different agents have different 'spheres of influence', in the sense that they will have control over - or at*

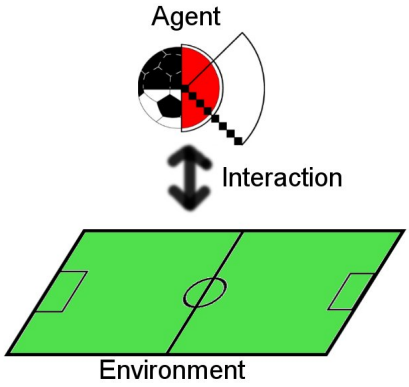

Figure 1.1: Interaction between an agent and the environment

> *least be able to influence - different parts of the environment." ([Woo02], p. 105)*

The most important aspect of a multi-agent system is, that there is more than one agent acting. There are three main elements that influence the interaction between agents [Fer01]: the agent's goals, the accessibility of resources and the capability of the agents. If all permutations are considered then there are eight different classes of interaction types. The main distinction depends on the compatibility of the goals of the agents. If the goals are compatible, this depends on the other influences like the resources or the agents' capabilities, the agents can work alone or together. Otherwise, if the goals are not compatible, there will be a competition between the agents.

In Figure 1.2 there are agents that act in an environment. The elliptic middle green round areas on the field are the spheres of influence of the agents. The sphere of influence of the agent on the left hand side has no intersection with another sphere of influence of an agent in the environment. Thus this agent has no interaction with others and works on his own. The other red agents have an intersection with the yellow agents. Thus it is called their *interaction*. It is marked as dark green area.

1.3 Decision Making

Decision making is the process of the derivation of an action from the current situation. There are many different ways of how to derive an action - it can be done either by probabilistic methods, potential fields methods, planning or learning. In all cases the basics for a good decision is the knowledge

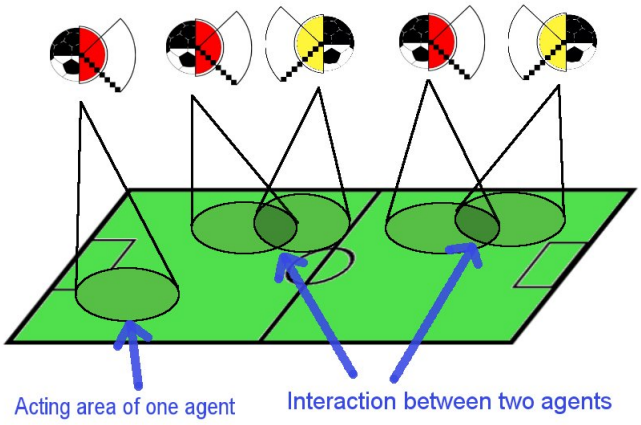

Figure 1.2: A multi-agent system with agents interacting

about the environment and the knowledge representation. This is the reason why artificial intelligence systems are often classified by their knowledge representation [Sav85].

The knowledge that is represented is built up from the environment. This environment can either be fully visible or partly visible to the agent. In a fully visible environment it is easy to model the environment because every information is available. In a partly visible environment the agent has to create his own view of the environment, that can contain unknown or uncertain information. But no matter how much information is available this information has to be stored and represented. Retti [Ret86] distinguishes two methods of knowledge representation [Ret86]: declarative and procedural. In the declarative representation the knowledge consists of a set of facts. This allows logical closures and is problem orientated. But it says nothing about how the knowledge should be applied. In the procedural method the knowledge is no longer represented as elements of the world but as an application of the knowledge. The knowledge base consists a set of procedures that have the capability to apply the knowledge. Minsky [Min74] has developed the frames as a third system of knowledge representation. This approach comes from image processing. Every frame has some typical slots and every element of the world can be assigned to such a slot. This third representation is a combination of the declarative and the procedural knowledge representation.

Chapter 2

RoboCup

In this chapter the RoboCup is presented. The different leagues of this International Federation will be outlined and the 2D Soccer Simulation League will be described in more details. Beside that the team *KickOffTUG* of the University of Technology, Graz, is introduced.

2.1 Introducing the RoboCup

The word RoboCup derives from the *Robot World Cup Initiative* and is an international project that supports the research on artificial intelligence and robotics. Therefore conferences and competitions are arranged between teams from different universities and countries. The original goal was to introduce a new standard environment for researchers. The operation field is a soccer game. One reason for choosing a soccer game is that it includes more than one agent. The agents have to interact and cooperate with each other in order to reach their common goal. Furthermore a soccer game is a highly dynamic environment. During the last two years new leagues beside the soccer field have been introduced. But everything is still under the vision of the RoboCup and under the corporate identity. Figure 2.1 shows the logo and thus the visual identity of the RoboCup.

The vision and the declared official goal of the RoboCup is:

> *"By the year 2050 develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team." [Rob07a]*

Today the goal still seems to be far away, but every year the robots improve their performance. Apart from that researches distributed all over the world work on the simulators, the leagues and try to improve the rules every year.

The number of leagues increased in the last years. This continuous progress is the reason why more people get involved into the research of autonomous agents.

Figure 2.1: The official logo of the RoboCup [Rob07a]

2.2 The Different Leagues

A robot consists of a lot of different components and techniques. In order to focus on only one main part of these components and techniques RoboCup introduced different leagues. Every league emphasises a different main point, although the basic questions are the same. Not every league deals with soccer, some of them deal with the usage of robots in another environment. An overview over the different leagues is given here. Further information can be found on the official website of the RoboCup [Rob07a].

2.2.1 RoboCup Soccer

In these leagues the environment is a soccer field. Most of these leagues were introduced at the start of the RoboCup. These leagues are called historic leagues.

Middle Size League:
This is the top class of all leagues. The prerequisition to be able to take part and be competitive in this league is a well working combination of hardware and software. But in these days more than this is needed - a high level of artificial intelligence and cooperation between the robots are necessary as well. Thus a good integration of all parts of the robots is the goal of the Middle Size League. In this league two teams are playing and each team consists of six players.

Small Size League:
This league focuses on the coordination and controlling of a multi-agent system. The robots are quite small (max. 18 cm diameter) and they have only a few sensors. To simplify the environment two external cameras are observing the playfield and send the information to an external computer. This

computer calculates the tactics and other things and sends the commands to the players.

Four Legged League:
In this league all participating teams have the same hardware. Until now this was the artificial Sony AIBO dog. As a result of the dismissal of the production of these dogs the new common hardware is the Aldebaran Nao. The main goal of the Four Legged League is to develop and apply software components on a well working hardware.

Humanoid League:
Robots like humans are the agents in the Humanoid League. The focus of this league lies on dynamic walking and kicking the ball while maintaining balance. The league exists in two types: the KidSize (30-60cm height) and the TeenSize (80-130cm height).

Soccer Simulation:
In the Simulation League there are no real robots present. The goal is to provide a research platform that is nearly independent of the hardware. The focus lies on the artificial intelligence and on the cooperation and coordination between the agents. The league has two facets: the 2D and the 3D Simulation League. The 2D Simulation League is described in more detail in Section 2.3.

2.2.2 RoboCup Rescue

RoboCup Rescue focuses on the rescue of people after a disaster. To achieve this many heterogeneous agents have to work together - physical robots to search and rescue people, information agents, that collect all information and decision support systems. The different leagues in the rescue domain are:

- Standard Robot-League
- RoboCupRescue Robot League
- Rescue Agent Simulation
- Rescue Virtual Robot Simulation

Further information about the rescue systems can be found at [Rob07b].

2.2.3 RoboCup@Home

This league has been introduced in 2006 and focuses on the application in a real-world environment. This includes self-localisation in the environment

and the human-computer interaction. The goal of this league is to create robots that can help people in everyday life. Further information about this League can be found online at [Rob07c].

2.2.4 RoboCup Junior

The RoboCup Junior is for pupils up to the age of 19. It supports the education and the understanding of new technologies. The students can participate in three different leagues:

- Soccer: This league takes part in two types of tournaments, the one-against-one tournament and the two-against-two tournament.
- Rescue: This follows the model of the RoboCup Rescue. The student's robots have to follow a path through a special environment representing a one-family house and find victims.
- Dance: Beside the technical skills creativity is requested, when the robots perform the selected dance performance. Beside the technical challenge the aim is to enhance female students enthusiasm about technical issues and studies.

2.3 2D Simulation League

In the Soccer Simulation leagues there are no real robots competing with each other, but programs. The Simulation league implements only on the essentials of the physical features, which are represented by the soccer server. Thus the planning and the strategies are the main focus. In the future the programs and algorithms developed in the simulation leagues should be transported to the real robots in order to improve their cooperation. The problem that occurs with the translation is the adaptation for the real environment. A main advantage of the Soccer Simulation league is that limited hardware is needed. The hardware of a robot is expensive and this league opens the research area to a broader field of students. To perform a game of the simulation league one personal computer is sufficient.

2.3.1 The environment

The environment of the 2D Soccer Simulation League is based on a simulator called soccer server, and two teams. Each team consists of eleven independent programs. Each program connects itself via UDP to the soccer

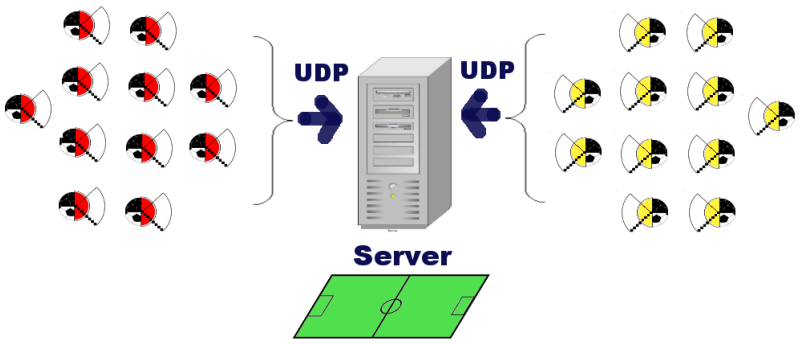

Figure 2.2: Interaction between the soccer server and the two teams competing

server that provides the physical environment. The setup of a game can be seen in Figure 2.2.

The interaction between an agent and the server is defined by a certain protocol. This protocol consists of basic messages for connecting, setting the team names and so on. But the important commands especially for the game that can be sent are catch, dash, kick, say, turn, turn-neck, change-view, hear, see and sense. A more detailed description of these actions can be found in Section 2.3.2. Another point to mention is that if the agent sends a command to the server then the server applies this action to the environment, if the command is valid. Otherwise the command is skipped.

In order to follow a game it can be displayed with a special monitor running on a personal computer that is connected to the soccer server. Games can also be stored and reviewed. This is very important for the analysis of the games.

2.3.2 Actions

Everything about the interaction between an agent and the server and all possible actions can be found in the official soccer server manual [CFH+02]. The basic actions an agent can perform on the environment (server) are: catch, dash, kick, say, turn, turn-neck and change-view. The main messages between the agent and the server can be seen in Figure 2.3. These actions are now described:

Catch:
Catch can only be performed by the goalie. The parameter for this command is a direction. The ball is caught if it is in the catchable area this cycle. The

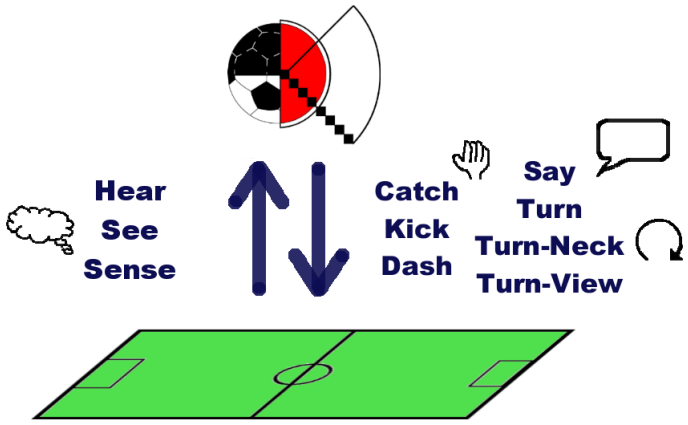

Figure 2.3: Actions that can be performed by an agent and senses which an agent can use to observe the environment.

area in which the ball can be caught is defined by a rectangle with a standard length and width.

Dash:
The command *dash* moves the agent with a certain power that is given as a parameter. The agent moves into the direction of its body, or backwards if the power is negative. The dashing uses a certain amount of the stamina of an agent. Every agent has a stamina that can be used for dashing and that recovers over time. The effective dash power with which the agent is moved is calculated by a multiplication of the effort and the stamina management of the player, with the dash power rate and the power.

Kick:
With the *kick* command a player can kick the ball into a certain direction with a certain power. The direction specified by an angle and the power are given as parameters. The command is only performed if the ball is within a limited distance to the player. Otherwise it has no effect.

Say:
Every player can distribute *say* messages to other players. The size of every message is limited. The characters a message can include are numbers, letters and some special characters like "+". The broadcast messages can be heard by players and the coaches of both teams.

Turn:
The *turn* command turns an agent's body by the agents movement that is given as a parameter. The resulting angle the agent actually turns depends not only on the movement, but also on the current agent's speed. If the

agent is moving it is more difficult to turn than when the agent is standing.

Turn-neck:
Every agent can be seen as splitable into a body part and a head part. These parts can be turned independently of each other. With the turn-neck command, the agent can turn its head part relative to its body. The parameter angle gives the relation between the body and the head.

Change-view:
The *Change-view* command can change the vision of an agent. It can be decided on the quality, the width and the frequency. While the width and the quality are given as parameters, the frequency is calculated by the server. With this frequency *see* messages are sent to the agent.

The agent observes the current state of the environment by getting messages from the server. These messages are hear, see and sense.

Hear:
With the aural sensor all messages that are sent by other players, the coaches or the referee are heard. Whether a message from another player is received depends on the distance between these two agents. When a message is heard the receiver gets information about the time, the sender and the message itself. The time is given in cycles of the game. The sender can either be the agent itself, a player of the own team, a player of the opponent team, one of the coaches or the referee.

See:
The *see* message represents the vision sensor. With that sensor the agent can observe every object that is on the field. These objects can be flags, other players or the ball. Flags are certain points on the field that facilitate the agent to calculate its own position. The maximum information that can be observed from one object are the name, the distance and direction relative to the agent, the changing of the distance and the direction and the body facing direction and the head facing direction. Whether all this information is transferred depends on the distance of the object and the object itself. The vision of an agent can be seen in Figure 2.4. In this figure there is one yellow player that is close to the red player. From this player all information is transferred to the red agent. From the other yellow player in sight of the red player the information is limited, there is no facing direction and probably no changing of the distance and direction. The yellow player which is below the red player is not in sight of the agent, thus no information about this agent is transferred him. In addition to all informations about the objects the time is transferred.

Sense:
With the *sense* command the agent can observe its own *physical* status. This includes the view mode, the stamina, the speed, the head angle and

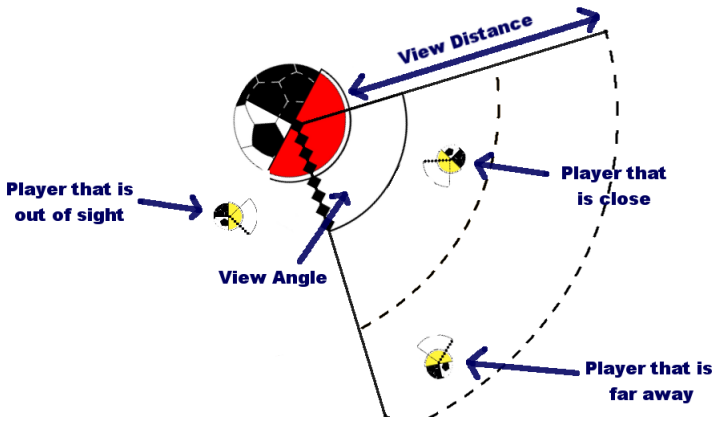

Figure 2.4: The agents vision sensor model

some counters. The view mode indicates the distance, the quality and the frequency of the *see* commands. The stamina represents the power condition of an agent by the current stamina and the effort. The speed gives an approximation of the current velocity and the direction. The head angle is the relative angle of the head in relation to the body. The counters are variables that indicate how often a command has been performed. Thus there is a counter for every command.

In order to know what action to perform the agent needs information about the environment and about itself. A characteristic of the Soccer Simulation league is that every agent can only partly observe the environment. In the agent's view there are informations about the ball, the own players and the opponent players. If a player is close enough to the agent the uniform number and the team can be identified. But if the player is far away it can be only identified as a player.

2.3.3 Cooperation in a Team

The cooperation between the agents in a team is essential, especially in the 2D Soccer Simulation League. The cooperation includes that not all agents from one team should go towards the ball at the same time. But what is the optimal position and behaviour of an agent? When should an agent dribble to the goal, when play a pass?

These questions are answered by the type of the decision making that derives the next action for an agent. The decision making algorithm has most of the time a reactive component that reacts on certain inputs. But it also has

a strategic component to improve the reactivity and reach a more complex behaviour.

2.3.4 Common Approaches

There are a lot of different approaches how the setting-up of a team can be fulfilled. In this part different strategies of several teams are outlined.

Brainstormers:
The main motivation of the team of the University of Osnabrück is on reinforcement learning. They try to develop new variants and practical algorithms for a complex domain. The reason for using reinforcement learning is that the success or failure of the agent is provided by the environment. The long term goal of the Brainstormers is to have a team of learning agents, where the only directive is "win the match" and the agents learn the appropriate behaviour. [RG08]

Virtual Werder Bremen:
The main focus of the students of Bremen was the 2D Simulation League until 2004. Since 2005 they focus only on the development of their 3D team. Nevertheless they had an interesting approach in the 2D Soccer Simulation League. They implemented the decision making process with potential fields. Thus every action of an agent was based on the position of players of the own team, the position of the players of the opponent team and the ball. All these influencing factors were represented by a potential field. [MAS+07]

UvA Trilearn:
In the early stages the goal of the team of the University of Amsterdam was to build a good foundation. After this huge step the research was focused on the interaction between the agents. The base of UvA Trilearn is still used by top-teams, like Dainamite [EWK+08]. Especially for young teams who build up everything from scratch the thesis *"The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team"* is a good start [BK02].

2.4 KickOffTUG

KickOffTUG is the RoboCup 2D Soccer Simulation League team of Graz, University of Technology. It is one of the few RoboCup teams in Austria with international experience. The last international competition the team participated in was the RoboCup World Championship in Atlanta, USA, in July 2007. Everything that is implemented in order to be analysed in this document is part of the team KickOffTUG.

2.4.1 The Team History

The team was founded in 2004 by three students named Stephan Gspandl, Michael Reip and Monika Schubert. These students implemented the basic skills of the team as their bachelor project. The students wanted to combine their acquired skills during their studies of *Software-Development and Knowledge Management* in a project. The goal at that time was to create a conceptual design and to implement it after the design phase. The focus was to build up the team from scratch in order to gain a better understanding.

The resources of these three students were not enough and thus new students were acquired. The new group of students divided the work into projects like revision of the world model, introducing a communication system, performing automatic testing, working on public relations and others. Since the beginning the team has grown and now it incorporates about 10 students from Graz, University of Technology. These students are in different stages of their studies. By now it reaches from optional courses and projects to bachelor and master's theses. The track record reaches from the first international participation at the RoboCup German Open in 2005 to a lot of local activities like workshops, recruiting days and so on. Another participation at the RoboCup German Open followed in 2007 and the highlight until now was the RoboCup World Championship in 2007.

2.4.2 Technical Details

The programming language of the team is Java. Java is an object-oriented programming language that is compiled to a byte code. This byte code runs on a virtual machine which makes it platform independent. The tool for the decision making process is *yprolog* [Sch07] which is integrated into the java code. The coding standard used for the implementation in Java is mainly influenced by the coding standard described and used in the book [Sch01]. Beside that a system to control the versions of the source code is used. This versioning system is integrated into the project management tool Trac [Tra07]. This tool also has the possibility to assign tickets, enter bug reports, support the communication in the team via a WikiWeb and manage working times of team members. Furthermore the tool Cruise Control [Cru07] is used for automatic builds and the automatic execution of JUnit tests. For the external representation of the team we have a homepage [kic07] running with the content management system Joomla [Joo07].

2.4.3 The Software Architecture

The software architecture of the team can be divided into three different layers: the basic, the midlevel and the abstraction layer. The basic layer is responsible for the communication with the server, which runs via UDP. When the server sends commands to the agent they are received by the *ServerInterface*. The useful information is passed from the *ServerInterface* to the *WorldModel*. This *WorldModel* is the representation of the environment for the agent. It calculates the own position with filters and stores the information about the other players and the ball on the field. Because of the included filters and the calculation the *WorldModel* is situated in the midlevel layer.

The abstraction layer consists of the *DecisionMaking* part, provided by the included Prolog and the *Strategy* modul. The *Strategy* covers the communication between the players, the positioning of each player depending on the other players and the models. This domain has implemented parts, but also a high abstraction. This is the reason why it is situated between the mid-level and the abstraction layer. Furthermore, it passes the basic actions that should be sent to the server in order to realize the strategy to the *Synchronisation*. In the *DecisionMaking* module the logical closure takes place. It uses the *Conditions* for the representation of the world. With these *Conditions* and some facts about the world the *DecisionMaking* derives an *Action* that should be executed. This selected *Action* is then passed to the *Execution*. This implements the action and generates the sequence of basic actions that should be executed on the server. These actions are forwarded to the *Synchronisation*. The *Synchronisation* builds the command that is sent to the server by the *ServerInterface*. An overview of the design can be seen in Figure 2.5.

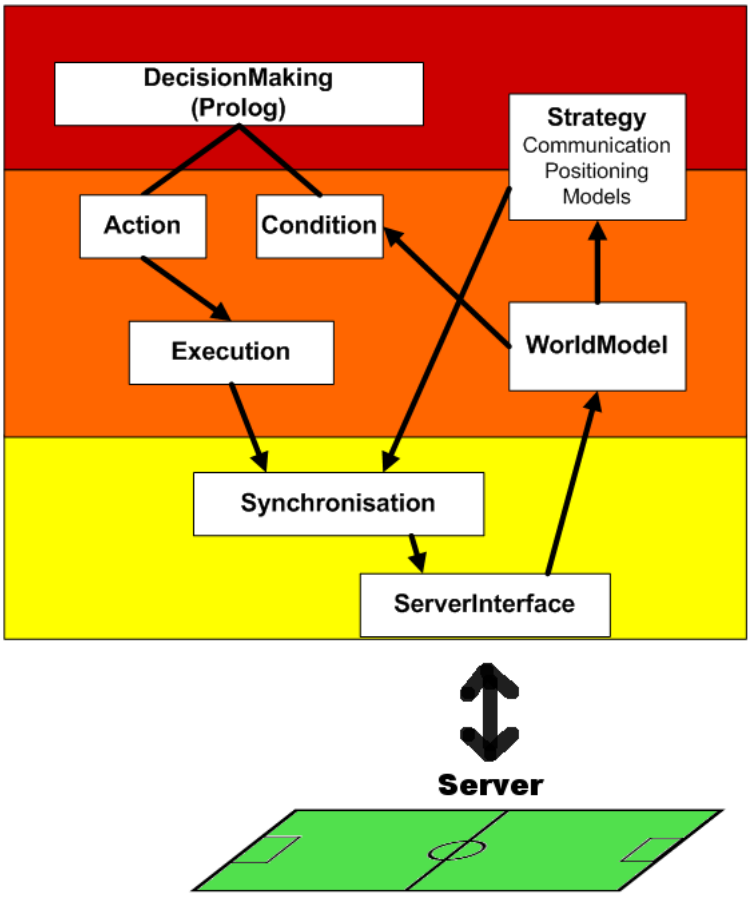

Figure 2.5: An overview of the software architecture of the team KickOff-TUG

Chapter 3

Planning

Like machine learning, planning is a part of artificial intelligence. There are lots of old stories based on artificial life, but the real beginning of the artificial intelligence was in the year 1956 with the Dartmouth-Conference, where a first definition of *Artificial Intelligence* (AI) was given. One of the researches that took part in this conference was McCarthy. He defines artificial intelligence on his homepage as follows:

> *"It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable." [McC04]*

Another definition more specific for robots says, that AI is the possibility to gain useful information out of raw sensor data [Hau07]. This intelligence can be implemented with a knowledge representation and a decision making algorithm. How such an implementation can work is described in this section of the work. Another possibility to implement intelligence into a robot is with a learning method, where the robot learns the knowledge through training examples or through acting in the environment and gaining a feedback (try and error). The learning methods as well as the implementation of the learning system are described in Chapter 4.

3.1 Definition

In this section a definition of and description about planning is given. A simple, but satisfying definition is given by Russel and Norvig:

> *"The task of coming up with a sequence of actions that will achieve a goal is called **planning**." ([RN03], p. 375)*

The planning is performed by a planning system which is part of an agent. Every agent can perform actions that change the environment. The selection of the next action the agent performs is done by the planning instance and specified by the agent's goal. When this goal is reached, the agent has nothing to do until it defines another goal. The sequence of actions or the action trace is built up of single actions. How these actions are chosen and the action trace is built up is defined by the planning system and the implemented planning algorithm.

3.2 Logic

Logic is a main part of artificial intelligence. There are three main fields of applications that logic is used for. The first one is as an analytic tool for checking if anything is true or false. Beside that logic can be used for knowledge representation and as a reasoning system. With a reasoning system new requests can be handled and an answer based on logical closure is derived. Most of the applications used as reasoning systems are based on the work of Nilsson ([Nil86] and [Nil71]). The third field is to apply logic as a programming language [Moo95]. The most common programming languages in the field of logic are *Prolog* and *Lisp*.

a	b	a $\wedge$ b	a $\vee$ b	a $\Rightarrow$ b
true	true	true	true	true
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Table 3.1: Example of a truth-table.

Logic in the way of reasoning was first introduced by Aristoteles. This classical logic was a two value logic with the values *true* and *false*. Nowadays the starting point for students to apply logic are truth-tables. These tables date back to Philo of Megara [San89]. In Table 3.1 an example is shown. In the first two columns all possible variations of the symbols are listed. The third one shows the boolean *AND* operator and the forth one the boolean *OR*. The output of two variables that are combined by an *AND* is only true if both are true. The operator *OR* is true, if at least one symbol is set to true. About the implication that is used in the last column Hilbert says:

> *"$X \Rightarrow Y$ is always true if X is false and also if Y is true" [HA50]*

The expansion of this propositional logic is first-order logic. This is described in Section 3.2.1.

3.2.1 First-Order Logic

First-order logic is, as well as classic logic based, on the boolean values *true* and *false*. While in the propositional logic every symbol represents a complete statement, in first-order logic predicates can be used [Lug02]. A symbol in the propositional logic could be the phrase *"the ball is in the right goal"*. This statement can be either true or false. The same sentence can be represented in first-order logic by the predicate *ballInGoal(right)*. The more abstracted predicate is *ballInGoal(X)*, where the parameter X stands for the goal. In a soccer match this would be *right* or *left*. In this example the syntax of first-order logic has already been used. The complete syntax consists of [EET01]:

- **Constants:** a constant is a defined object
- **Variables:** a variable stands for an object, that has not yet been defined
- **Functions:** a function specifies an object determined by the function out of all parameters $o = f(o_1, o_2, ..., o_n)$
- **Predicates:** a predicate with one parameter specifies an attribute; if it has more parameters it represents a relation between the objects

With this syntax terms, facts, literals and formulas can be described. In order to prove if a sentence is true or false, the modus-ponens rule

$$\frac{A \wedge A \rightarrow B}{B}$$

can be applied. This rule says if A is true and the implication from A to B is true then B is true as well. Another possibility to prove the truth is the *resolution*, which has been first introduced by Robinson [Rob65]. An example of a proof with resolution can be found in Figure 3.1. The goal of this example is to find out if another player is a possible partner for a pass. The predicates that will be used to describe the situation are:

- **HasBall(X):** describes if the player X has the ball
- **Member(X, T):** describes if the player X is from team T
- **SameTeam(T1, T2):** describes if team T1 is same as team T2
- **PossiblePassPartner(X, Y):** describes if the player Y is a possible pass partner for X

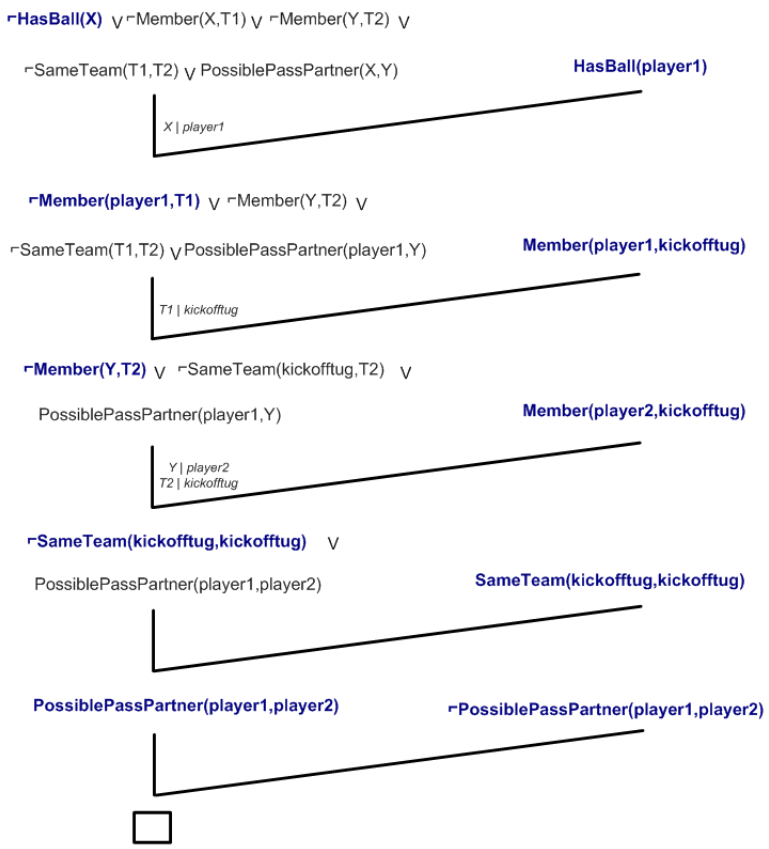

Figure 3.1: Interaction between the environment and the agent

These are just abstracted predicates. In order to perform the resolution a knowledge base is needed. All facts about the environment are stored in the knowledge base. Furthermore it can contain informations about the relationships between objects. The first entry in the knowledge base is the rule, that if the player x has the ball and if he is from the same team as the player y, then the player y is a possible pass partner for the player x. This is just a simple definition. In a real game for example it would be checked if the pass partner is in an area that is close enough for passing and the player should not be covered by an opponent player and so on. Other entries in the knowledge base are describing the situation. For example that the player *player1* has the ball and that both the players *player1* and *player2* are from the team *kickofftug*. Beside that the predicate SameTeam is used to specify that the team *kickofftug* is the same as *kickofftug*. This is just a formalisation that is needed. The complete knowledge base of our example is:

- $\forall X \forall Y \forall T1 \forall T2$ (HasBall(X) $\wedge$ Member(X, T1) $\wedge$ Member(Y, T2) $\wedge$ SameTeam(T1, T2)) $\rightarrow$ PossiblePassPartner(X,Y)
- HasBall(player1)
- Member(player1, kickofftug)
- Member(player2, kickofftug)
- SameTeam(kickofftug, kickofftug)

The goal of the resolution is to find out if the player *player2* is a possible pass partner for the *player1*. This is formalised by PossiblePassPartner(player1, player2). For the resolution this theorem is negated. The complete resolution can be found in Figure 3.1.

3.2.2 Fuzzy Logic

The main difference between fuzzy logic and all other types of logic is that in fuzzy logic not only the boolean values *false* (0) and *true* (1) are available, but also values that lie in between. Thus it is possible to express statements like *in between* (0.5), *nearly true* (0.75) or *nearly false* (0.25). This makes fuzzy logic more complicated to understand, but increases the expressiveness considerably. Beside that it is closer to human thinking than any other logic.

An example, that shows why fuzzy logic is needed is the sentence: *"I am lying"*. Can this sentence be true? If it would be true, the person would be lying, and thus implicates that the sentence is false. This is not possible. On the other hand if the sentence would be false, the person tells the truth. But the statement can never be true. Thus boolean logic is not expressive enough for this example.

As already mentioned fuzzy logic is an extension of classical boolean logic, which uses the values *true* and *false*. A fuzzy set is described over a set X, which is a subset of $X \times [0, 1]$. This fuzzy set can be described through the function [LC01]:

$$\mu \rightarrow [0, 1]$$

Figure 3.2 shows an example for two fuzzy functions. On the x-axes the temperature is assigned and on the vertical y-axes the resulting fuzzy values can be read. The function drawn in *blue* symbolises a function for the attribute cold and the *red* one for hot. The relative values can be seen in both functions as a changeover between the states 0 and 1. Functions like the ones in Figure 3.2 are common for fuzzy attributes.

Although the logical closuring can be applied for fuzzy logic, the rules are different from the ones used for boolean logic. The rules for logic closuring

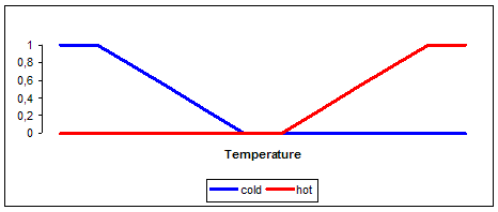

Figure 3.2: Example for fuzzy functions

of fuzzy logic are:

$$T(a \wedge b) = min(T(a), T(b))$$

$$T(a \vee b) = max(T(a), T(b))$$

$$T(\neg a) = 1 - T(a)$$

where T is the fuzzy-truth of a complex sentence [RN03]. The problem with fuzzy logic is that the correlations between the two sentences combined can not be represented. A suitable application of fuzzy controls lies in automatic gearboxes and video cameras [Elk93].

3.3 Current Implementation

In the current system of KickOffTUG the decision making is implemented with a planning variant. This planning system and everything combined with it is described in this section.

3.3.1 System Overview

In Section 2.4.3 the architecture of the team *KickOffTUG* has been described. Now the focus of the system description lies on how the artificial intelligence is implemented. The algorithm to derive an action from the current state of environment is based on a planning algorithm. More about this algorithm can be found in Section 3.3.4. The planning system consists of the AI-Level and a backend. This backend is implemented in Prolog and is responsible for the resolution of decisions. The knowledge that is needed for the logic closures is provided by the AI-Level. This AI-Level consists of actions, conditions and plans. A plan itself is based on actions and conditions. The conditions provide the knowledge to choose an action. This abstracted knowledge about the environment is derived from the information provided by the *WorldModel* which is a part of the system level. The

pool of all actions provides all possible actions that can be performed in the environment. The actions as well as the conditions are in an abstract form to ensure portability. Thus the real implementation of the actions lies in the *Execution* which is a part of the system level, too. An overview of the system can be seen in Figure 3.3.

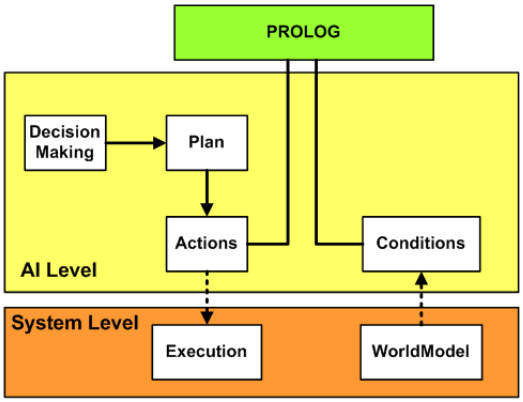

Figure 3.3: Interaction between the environment and the agent

3.3.2 Knowledge Representation

The knowledge agents needs for a closure consists of facts and conditions. Facts are statements that are always true in the environment. Conditions describe a certain state of the environment. For example the condition *hasBall* is true, when the agent has the ball in the current state, otherwise it is false. Thus the value of a condition changes over time.

Every Prolog action can have preconditions and postconditions. Preconditions are conditions which have to be true before the action can take place. A post condition is a condition indicating something about the state after the action was performed. For example in order to perform the action *kickBall* the agent has to have the ball - *hasBall* is the precondition. If this precondition is not true, the agent is not able to execute the action successfully. After the action has taken place the ball is somewhere else than in the area of the agent - thus the post condition is *notHasBall*.

3.3.3 Plans

The planning algorithm derives its decision based on static plans. Every plan consists of actions and every action consists of preconditions and postconditions. An example plan can be found in Figure 3.4. This plan is from

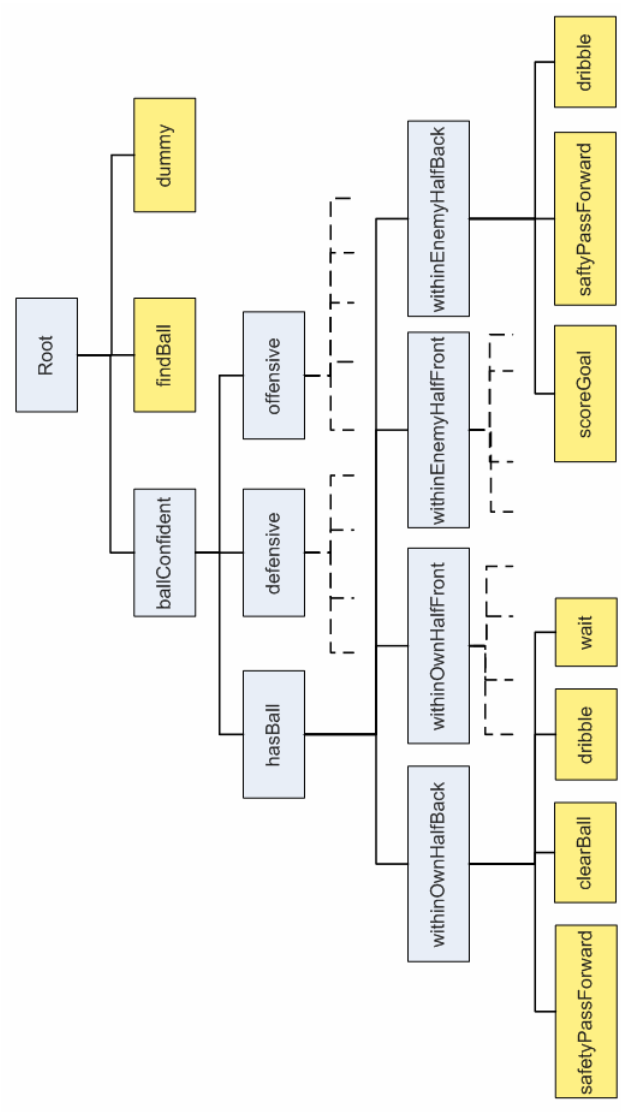

Figure 3.4: Plan of a striker for the regular game

a striker for a regular game. Every plan is built up like a tree. Every node in the tree holds an action. Every leaf is an action as well, but beside that it is a goal. As already mentioned we have more than one plan. There are four different player types: goalie, defender, midfield and striker. Every player type has a plan for the regular game (regular plan) and a plan for standard situations (standard plan). As the midfield is divided into a central midfield and a wing midfield, there are plans for:

- Goalie
  - Regular plan
  - Standard plan
- Defender
  - Regular plan
  - Standard plan
- Central midfield
  - Regular plan
  - Standard plan
- Wing midfield
  - Regular plan
  - Standard plan
- Striker
  - Regular plan
  - Standard plan

It is obvious that the plan of a striker is much more offensive than the one of a defender. Beside that the main focus of a striker is to get the ball and score a goal, while a defender should cover the opponent or close open space. The defenders are built up in a straight four-man backfield so it does not make sense to split the defenders into a centre and a wing. All players should try to stay in a line which makes the plans similar.

3.3.4 Decision Making Algorithm

The decision making algorithm describes how an action is derived from a plan. This algorithm is based on a combination of the *STRIPS* (Standford Research Institute Problem Solver) [NF71] and the Teleo-Reactive-Formalism by Nilsson [Nil94]. *STRIPS* is a planner that uses a set of

actions to decide upon. Every action consists of preconditions and postconditions. The *STRIPS* algorithm chooses an action for which the preconditions are true and postconditions are false. The idea of the Teleo-Reactive-Formalism is used to assign more flexibility to the agents. Every Teleo-Reactive-Formalism has a goal (teleo) which should be reached under dynamically changing conditions. The agent has to react on this changing environment (reactive). In the current implementation the observation of the environment is done by the conditions, which are evaluated every time a decision should be taken.

The plans on which this decision making depends on have already been described in Section 3.3.3. The sequence of the elements in these plans are crucial as the tree is traversed by a depth-first search. This makes an assignment of a reward, as proposed in [BN95] redundant. Although the algorithm of implementation has already been described in [Gsp07] and [Rei07], it is fundamental to describe it here again:

```
decide(root_node)

FOR all children of root_node
  retrieve action from actual child
  IF postcondition of action is true
    IF root_node is a leaf node return,
       because a goal has been reached
    ELSE decide(actual child)
  ELSE
    IF precondition of action is true
      execute the action
```

An example how this could work in a tree is shown in Figure 3.5. The assumption of this example is that in the current state the predicates have the values listed in Table 3.2.

Predicate	Value
seeBall	true
hasBall	true
freeShotPossible	false
closeToGoal	true
goalScored	false

Table 3.2: Predicates and their values for a sample state of environment

The starting point for the algorithm is the root node. The first child (*Intercept*) is considered. The action *Intercept* has *seeBall* as a precondition and *hasBall* as a post condition. As the post condition is true, the current

node (*Intercept*) is the new root node and the nodes in the next level are examined. Now the post condition of the action *ScoreGoal* is considered. This is false and thus the preconditions *freeShotPossible* and *closeToGoal* are checked. One of these is false and thus the next action in the same level is the current one. This action (*Dribble*) does not have any post condition. By default the post condition is always false, because otherwise the action would never ever be considered. This stands in contrast to the preconditions. If a precondition is not set, it means that it is true, thus the action can always be performed. The precondition of *Dribble* is obviously true, because otherwise we would not be in this branch. For this plan and the specified state of environment the action *Dribble* is derived.

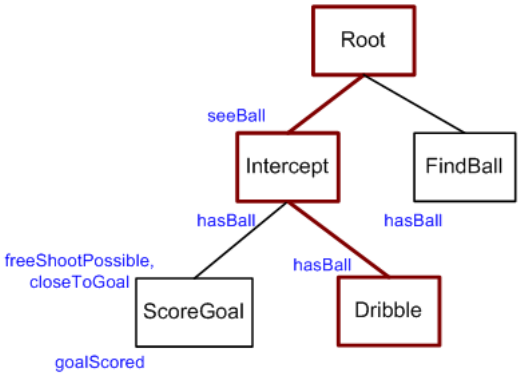

Figure 3.5: Example for the derivation of an action from a plan

The logical closure of the system is implemented in Prolog. A program written in Prolog is based on the Horn clauses. In comparison to this a Lisp program is based on the Lambda calculus. Prolog was chosen because it is expressive and flexible enough to correspond to the requirements of the system.

Chapter 4

Learning

In comparison to planning, which is based on logical closure, the field of machine learning is founded on probabilities. The main research goals of the machine learning community is to analyse the different principles of machine learning in order to apply them in many different fields. The application fields of machine learning reach from data mining programs to information filtering systems that learn the users preferences, to autonomous robots or vehicles [Mit97]. RoboCup is an environment created to enforce the research of machine learning. Researchers are applying and analysing algorithms in order to improve them and to gain more information about the test and learning environment.

In this chapter a definition of machine learning is given. Afterwards the main principles of design decisions that need to be taken when applying an algorithm to a specific problem are described. This leads to the design of the implementation of a learning algorithmn the team *KickOffTUG*. A fundamental question of a learning system concerns different learning methods. These are presented and analysed in this chapter.

4.1 Definition

Learning is one of the main differences between a creature and a machine. The combination of increasing computing power and different learning algorithms closes the gap between machine and creature. But the term learning is quite huge if the different applications are considered. The different learning classes are [Maa07]:

- **Classification learning:** The input data has to be classified into different categories. An example is to learn the recognition of faces of different people or to learn the understanding of a language.

- **Regression:** This deals with the prediction of the future, for example how long will the rain remain or how long will it take until a patient gets well again.
- **Unsupervised learning:** Knowing something that has not been taught. For example if a car makes strange noises, human will recognize them without having heard them before.
- **Reinforcement learning:** Learning a strategy in order to achieve a predefined goal or to survive.
- **Adaptive control, motor learning:** Learn how to control different parts of a machine for example in order to stand up.
- **Memory management:** For a machine this deals mainly with sorting algorithms, but for humans it is much harder to recognize learned entities such as vocabularies.
- **Imitation learning:** Learn to imitate another person or another machine.

Although the different types of learning have already been listed, the term *learning* has not yet been defined. Mitchell has a good definition for a learning task. He says:

> *"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." ([Mit97], p. 3)*

This means every learning task consists of three main features. First of all the task: In the studies for this work the task will be *playing soccer*. This is just an abstract description. The task is much more complex, but in order to give an example it is enough. The second feature is the measurement of the performance. The performance can be measured in games won or goals shot for example. The last but not least feature is the training experience. It describes how the training is situated. In this paper it will be games played against another team.

4.2 Design Decisions

The design of how the learning is implemented to an agent is crucial for the success. There are different design decisions to make. These are described in this section. For further informations take a look at [Mit97].

4.2.1 Type of Training Experience

The type of training experiences deals mainly with the feedback that is provided by the environment. This feedback can either be direct or indirect. If the environment provides direct feedback every turn can be analysed. If only indirect feedback can be returned to the agent, the agent has to analyse a sequence of turns leading it to success or failure. The indirect feedback is harder to analyse because it can contain optimal movements, but still having a negative outcome due to other non-optimal movements performed in the same period. Therefore direct feedback is easier to handle.

An alternative to distinguish the training is by availability of a trainer or a master. Basically there are three different types of trainings: supervised, self-supervised and unsupervised [Neh03]. Supervised learning means that the training information is provided externally. This external trainer can be a human. The supervised learning mechanism is often applied with neural nets. In case of the self-supervised training the learning mechanism stays the same, but the external feedback is replaced by an internal feedback. The training agent needs a control structure to generate that internal feedback from the state of the environment. Reinforcement learning is a method that uses mostly self-supervised control. Unsupervised learning is not based on input-output pairs, but by exploring and analysing the underlying structure. This is mainly used for classification or self-organising feature maps [Koh88].

Another important issue is how the training is situated. There can be a teacher controlling the training. For example the trainer controls the optimal sequence of states. It can also provide the optimal action for every state of the environment. In the Soccer Simulation League this is infeasible, because there are too many states and furthermore for every state the best action is not unambiguous. If there is no omniscient trainer available, the agent can perform the training against another team. This opponent team can be either an instance of the own team or another team. No matter which team is chosen, it must be considered that the learned behaviour is nearly independent of the other team. Otherwise the learned team will just be specified to win against the team of the training. This leads to another crucial aspect of the machine learning design process - the distribution of the training examples. This distribution is responsible to learn an abstract behaviour, thus it has a high impact on the final outcome.

The training is built up in a way that the agent performs an action on the environment. The environment changes the state and with that state the agent can analyse if the action that it performed was good or not. Thus the agent is experimenting with the states of the environment. The interaction of the environment and the agent is shown in Figure 4.1.

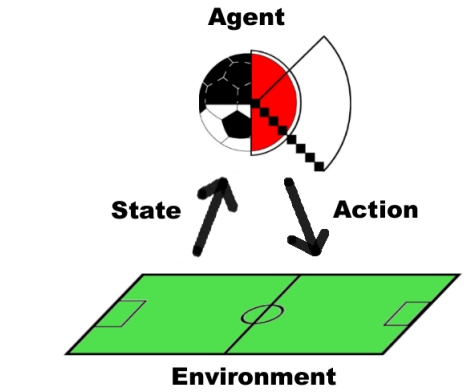

Figure 4.1: Interaction between the environment and the agent

4.2.2 Target Function

The target function tells exactly what type of knowledge will be learned and how it will be used by the performance measurement tool. The knowledge can be a mapping of the environmental state to the best action: $T : S \rightarrow A$. Another approach says that the target function assigns a real value to every state of the environment: $T : S \rightarrow \Re$. This real value describes how good the state is [Mit97]. In all cases the target function tries to find the best legal action or move for the agent.

The considered state of the environment is important for the performance of the learning application. In a real world environment this can be very complex. Therefore the observation of the environment has to be simplified. This is reached by a reduction of the observed parameters, meaning not the whole world needs to be modeled, but only the area that can be influenced by the agent.

4.2.3 Representation of the Target Function

The target function can be represented in different ways. Nearly every mathematical function that takes a state of an environment as an input can serve as a target function. The most popular functions are linear functions, polynomial functions and artificial neural networks. The input can either be a single value or a vector describing the environment. In the formulas described in this context x is the input. Beside that every function has some weights represented by w. The linear function can look like

$$T(x) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_4 * x_4 \tag{4.1}$$

The advantage of a linear function is that it is easily approximated, because it is quite simple. But the disadvantage is that its expression is limited. The polynomial function like

$$T(x) = w_0 + w_1 * x + w_2 * x^2 + w_3 * x^3 + w_4 * x^4 \tag{4.2}$$

is more expressive then the linear one. Bishop shows the expressiveness of different functions in [Bis02]. In his example a noisy $\sin(2\pi x)$ function is approximated with different polynomial target functions. While the linear function was not expressive enough the third order polynomial worked out fine in the training set as well as in the latter behaviour. The problem with higher polynomial was that the function was too accurate and thus it was only appropriate for the training set, but not for the final performance. This is called *overfitting*.

Another possibility is to represent the target function as an artificial neural network. A single neuron can be modeled by the Heaviside-Function:

$$T(x) = HF(w^T * x - \Theta) = \begin{cases} 1 & : \quad w^T * x - \Theta \\ 0 & : \quad else \end{cases} \tag{4.3}$$

This function checks if the input vector multiplied by the weights is greater than a certain threshold Θ. If the condition is true, the output is 1 otherwise the output of the neuron is 0. Neural networks can be very expressive depending on the layers. These networks are described in more detail in Section 4.3.3. In order to get the perfect target function a very expressive representation is needed. But the more expressive a representation is, the more training examples are needed to choose among the hypotheses.

4.2.4 Choosing a Learning Algorithm

The learning algorithm describes the approximation of the target function. This is done by a set of training examples that are performed. After each training session the weights of the target function are adjusted. Usually this is done by minimizing an error function [Bis02]. Such an error function normally measures the difference between the function and the training example. An error function for this example would be:

$$E(x) = (V_{Train}(x) - V_{Predict}(x))^2 \tag{4.4}$$

where x is the training value. V_{Train} is the value of the training set and $V_{Predict}$ is the predicted value of the algorithm. Since the error should always be positive the resulting value is squared.

It is nearly impossible to learn a perfect target function. Usually the target function is an approximation. This approximation is easier to reach and serves the purpose as well, sometimes better. A too accurate target function can be specified to the training examples, but does not have the flexibility to serve the real environment. This problem is called *overfitting*. A more detailed description about the different learning algorithms and approximation methods can be found in Section 4.3.

4.2.5 The Resulting Design Models

The design of the learning system consists of four different modules. These modules are the *Performance System*, the *Critics* or *Analyser*, the *Generaliser* and the *Experiment Generator*. The interaction between these modules is shown in Figure 4.2.

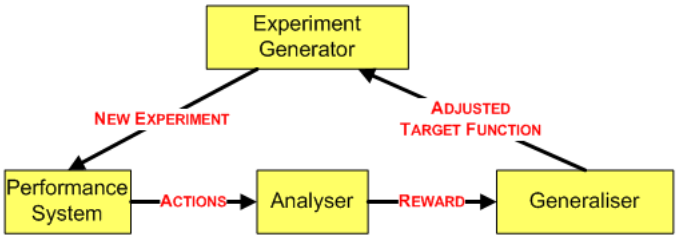

Figure 4.2: The interaction of the program modules of a learning system

Performance System:
The performance system implements the task. It takes a new experiment as an input and produces a single action or a trace of actions as an output. The mapping from the input state to the output is done by the target function.

Analyser:
The analyser gets the action trace from the performance system as an input and analyses the impact of these actions on the environment. With the new state of the environment it generates a reward as an output.

Generaliser:
The generaliser implements the main part of the learning algorithm. It takes the reward of the analyser and adjusts the weights of the target function for the whole learning system. This is often done by an algorithm which is

based on the gradient decent approach of the target function. The adapted target function is forwarded to the experiment generator.

Experiment Generator:
This module generates a new experiment and starts the learning cycle again. In every cycle the target function is updated and thus more trained.

4.3 Learning Methods

In this section different learning methods are described and analysed. It gives an overview of the most common techniques used.

4.3.1 Decision Trees

The input of a decision tree algorithm is a vector describing an object or situation. The elements of this vector are also called attributes. The output is a decision that is derived after performing a sequence of tests. An internal test is represented by an internal tree node and checks the value of one of the properties of an attribute. [RN03]

An example decision tree can be found in Figure 4.3. The input vector enters the decision tree in the root node and is tested there. Every internal tree node represented by a yellow circle is a test case. As a result of this test regarding to the input vector one of the edges is taken. A link between the nodes can be interpreted as a test result. These test results can either be a numerical split or a nominal split. The tree is traversed by performing the internal tests until a leaf is reached. This leaf represents the decision or the class an input vector can be classified into.

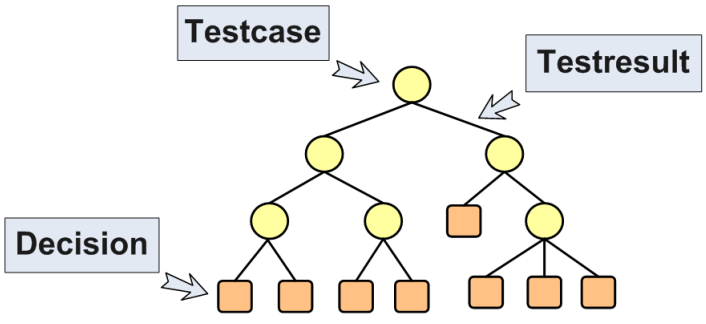

Figure 4.3: The representation of a decision tree with test cases, test results and decisions

Example of an Algorithm

There are some different learning algorithms for decision trees. The basic one is ID3 [Lug02]. It is the basis of more expand ones like C4.5 [Qui03]. Therefore the ID3 algorithm is described in pseudo code here:

```
Function ID3 (Examples, Target_Attribute, Attributes):

Create a root node for the tree;
IF all Examples are positive
  RETURN the single-node tree root with a positive label;
IF all Examples are negative
  RETURN the single-node tree root with a negative label;
IF Attributes is empty
  RETURN the single-node tree root with the label of the
    most common value of the Target_Attribute;

BEGIN
  A = an attribute that best classifies the examples
  Decision tree attribute for root = A
  FOR EACH value vi of A
    add a new tree branch below root,
      corresponding to the test A = vi
    let Examples(vi) be the subset of Examples
      that have the value vi for A
    IF Examples(vi) is empty
      below this new branch add a leaf node with label = most common
        value of the Target_Attribute in Example
    ELSE
      below this new branch add the subtree
        ID3(Examples(vi), Target_Attribute, Attributes-{A})
END
```

This algorithm builds decision trees using a top-down technique. All the training examples have to be classified. These classes are called *decision* in Figure 4.3. This classification is done by choosing an attribute of the training example that splits all examples into two classes. If all examples are in one class the algorithm terminates. Otherwise a new subtree with another splitting test is added. Finally the algorithm builds a concept by eliminating candidates out of the training examples. The main advantage of this algorithm is the representation in decision trees. These trees can easily be reviewed by a human. Furthermore the classification of the decision is suitable if the incoming data is noisy. [Lug02]

A problem occurs if the resulting tree gets big, because in a huge tree it is hard to preserve the overview. A possibility to shrink a decision tree is pruning. Pruning compares subtrees and eliminates redundant entries. The pruning algorithm can either be applied during the training phase or after the complete training.

Application in the RoboCup

Virtual Werder, a RoboCup 2D Simulation League team, used decision trees to analyse the behaviour of the opponent players. The research group implemented the learning algorithm into the online coach. With this algorithm the coach evaluates the formation, the preferred defensive behaviour and the offensive play of the opponent team over a certain time. The values recorded over the interval are the input for the decision tree algorithm. This algorithm generates rules and out of these rules the coach sends commands regarding the opponent behaviour to the players. With this information the players can adapt and improve their behaviour. In order to get better rules for the learning algorithm the decision tree is evaluated after the game by a human. [DHVW02]

Evaluation

A simple example of how the decision tree learning can be introduced in the soccer domain is shown in Figure 4.4. In this example the internal tree nodes represent some kind of condition. In most cases this condition has two outcomes, either it is fulfilled or not. Only the distance of the ball has more possible solutions: *nearby, far* and *too far*. After performing the tests a decision is derived. Every leaf node represents an action that can be performed. The possible actions in this example are: *Find Ball*, *Dribble*, *Run to Strategic Position*, *Run to Pass Position* and *Cover Enemy*.

As every internal test node represents a condition, this learning method is appropriate to evaluate the conditions and how they are suited for the actions. For this evaluation some conditions can be combined in such a way that their outcome is not boolean anymore. The path from the root node to the leaf defines a conjunction of conditions under which the action should be executed.

The main problem of the decision tree is its size. If a lot of different actions and a high number of conditions are used in the training examples the decision tree gets huge. Another problem is that the algorithm needs training examples. Because it is hard to automatically create these training

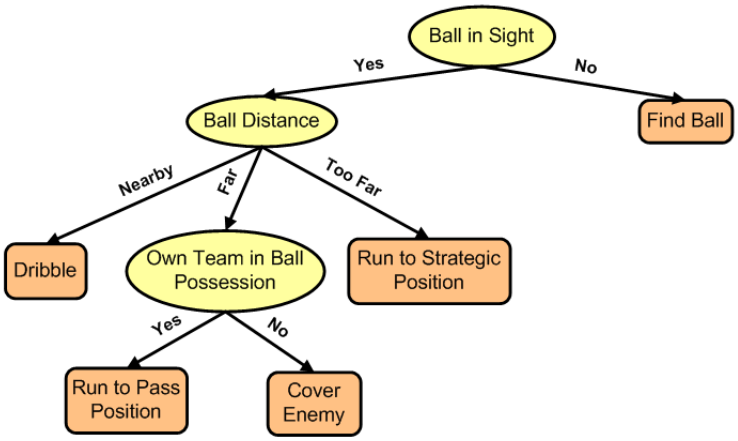

Figure 4.4: Example for a simple application of a decision tree.

examples all the training examples have to be built by hand. This is not applicable in the team KickOffTUG, thus this learning method is not used.

4.3.2 Genetic Learning

The genetic learning method is inspired by biological evolution and natural selection. The idea behind is, that there exists a population of hypotheses and these hypotheses are updated by crossover and mutation at each step. The goal is to find the best hypothesis over a given search space. How appropriate a hypothesis is for a given task is determined by the fitness function. The fitness depends on how the chromosome behaves according to the problem [Lug02]. In this context a chromosome is a string containing important information coded into bits.

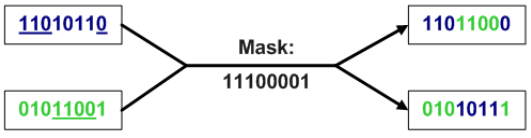

Figure 4.5: Example for the crossover of a string with a crossover mask of 11100001

One advantage of this learning method is that the evolution on which it is based has been a successful and robust method within biological systems. Another one is that the hypotheses can contain complex and interacting data that are difficult to model. This connection between the different parts

of the data can be automatically improved by the algorithm. Beside that it is easy to parallelize the calculating and therefore increase the resulting computing power [Mit97].

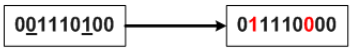

Figure 4.6: Example for the mutation of a string with the mutation rate of two.

The different genetic learning implementations vary in details, but the basic concept is the same. The algorithm deals with a pool of hypotheses. Out of this pool some are selected and used for reproduction. These selected ones are used for the new generation which is produced by crossover and mutation. Crossover is the exchange of substrings. How the substring is created depends on the crossover mask. An example is shown in Figure 4.5. A mutation is the invertion of single bits. The amount of these bits is called mutation rate and it is usually quite small to ensure consistency in the population. An example of mutation is shown in Figure 4.6.

Example of an Algorithm

The algorithm is based on the main steps of evolution: selection, crossover and mutation. The input data for the algorithm on one hand is the fitness function that assigns a value to a given hypothesis. On the other hand values that specify the learning have to be defined. First of all a threshold that is responsible for the termination. Beside that the algorithm takes a parameter that declares the number of hypotheses, another one for the fraction that should be replaced by the crossover at each step and the mutation rate. Another important issue of the algorithm is the probability criteria for the selection. This is given by

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)} \tag{4.5}$$

This formula calculates the probability that a hypothesis h_i is selected out of all given hypotheses.

```
GeneticAlgorithm(fitness, threshold, p, r, m)
  P <- Generate p hypotheses at random     // init population
  FOR EACH h in P compute fitness(h)       // evaluate

  WHILE(max fitness(h) < threshold) do
```

```
create a new generation Ps
// Selection
probabilistically select (1-r)p members of P to add
  to Ps the probability of the selection is given
  by Pr(hi)
// Crossover
probabilistically select r*p/2 pairs of hypotheses
  from P, according Pr(hi)
  for each pair <h1,h2>, produce two offspring by
  applying the crossover operator
  add all offspring to Ps
// Mutation
choose m percent of members of Ps with uniform
  probability for each invert one randomly
  selected bit
// Update
P <- Ps
// Evaluate
FOR EACH h in P compute fitness(h)

RETURN the hypothesis from P that has the highest fitness
```

Each iteration in this algorithm is called generation. The set of performed generations is called a run. After a run there should be fitter chromosomes in the population. But the problem is that this algorithm highly depends on the random function. Thus it can happen, that the chromosomes are less fit after a run than before. Another aspect of the random function is that two runs with different random functions produce a different behaviour.

Application in the RoboCup

One of the first applications with genetic programming in the RoboCup was done by Sean Luke ([LHF+97], [Luk98a] and [Luk98b]). In his experiments he found out that homogeneous players, all players have the same source code, performed better than heterogeneous teams. These heterogeneous teams had almost the same implementation, just the sub trees differed. Luke figured out that the problem was the complexity. If more computation time would have been available, the heterogeneous teams would have performed better.

Evaluation

Klepper [Kle99] already implemented a decision making algorithm with genetic programming. This decision making was based on high level actions, which should be selected by the genetic algorithm. The problem he experienced in his experiments was that it was hard for the agents to learn any defensive behaviours. Klepper also analysed the performance of low level actions in comparison to high level actions and he figured out that the usage of high level actions was always better than the low level ones.

The genetic learning method is a general approach that has a satisfying outcome if the period of searching is long enough. But the problem is that the algorithm strongly depends on the accuracy of the fitness function and on the encoding of the problem into chromosomes. Finding these two parameters can be very time consuming especially in a complex environment.

4.3.3 Neural Nets

Neural nets are inspired by biology, like the genetic learning method. In comparison to the chromosomes of the genetic learning method the neural nets are based on the human brain. Our brain consists of millions of connected neurons. A schematic of a single neuron can be seen in Figure 4.7. This neuron consists of the input values x_i and the weights w_i. In the neuron itself the Heaviside function (Formula 4.3 on page 32) is implemented. The output of a neuron is a single value. The problem with a single neuron is that it is not expressive. It can be used to represent linear functions like the boolean AND operator (the example can be found in [Wal05], page 2).

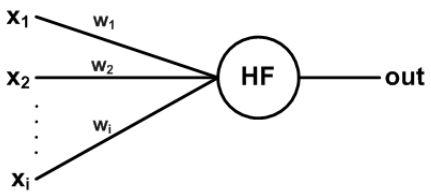

Figure 4.7: Example of a single neuron

For more complex applications a net of neurons is used. An example neural net can be seen in Figure 4.8. This neural net consists of three layers: the input layer, one hidden layer and the output layer. Every neuron that takes one or more input values is part of the input layer. The hidden layer has neither direct input nor direct output, thus it is not visible to the application. The optimal number of hidden layers is an unsolved problem. Bailey and

Thompson [BT90] created a "thumb"-rule that recommends that the number of hidden neurons should not exceed 75% of the number of input neurons. The output layer incorporates all neurons that produce a result for the input values. More complex architectures of neural nets can be found in [LC01].

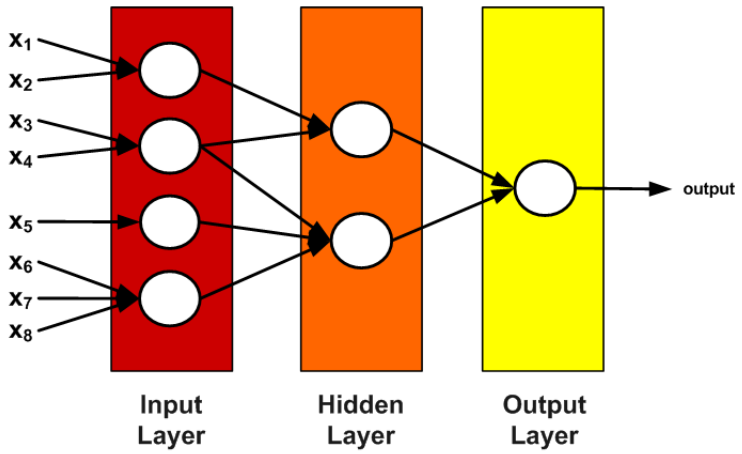

Figure 4.8: Example of a neural net with an input layer, one hidden layer and an output layer

The main advantage of neural nets is that it is possible to learn complex patterns without learning the rules. The benefit of a neural net compared to traditional probabilistic methods is that the nonlinearity in the dataset can be used. Another strength of the neural net is that they are fault-tolerant and still applicable if the data is inconsistent. On the other hand neural nets are not appropriate for applications that are based on structured problems with corresponding theories [Wal05]. This is based on the fact that there are better learning methods if a model is available.

Example of an Algorithm

The most common algorithm to train neural nets is the backpropagation algorithm. This algorithm is based on the gradient descent. The gradient descent is an iterative method for finding the local minimum in a certain space. The backpropagation algorithm is suitable for multilayer networks with a fixed size of input, output and hidden neurons. Furthermore the connections between the neurons must be fixed in advance. The weights of the net are adjusted by the squared error between the output value and the target value. Due to a higher number of outputs the sum over all these

output values must be taken for calculating the error. The resulting formula for the error is:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \tag{4.6}$$

The input for the backpropagation algorithm is a set of training examples. Each training example is a pair of the input vector $\vec{x}$ and the target vector $\vec{t}$. The learning rate specifies how much each weight will be adapted. As already mentioned the algorithm needs to be specified by the number of neurons and connections between these neurons. The following algorithm defines the neural network by the number of inputs n_{in}, the number of hidden neurons n_{hidden} and the number of output units n_{out}.

```
Backpropagation
(training_examples, learning_rate, n_in, n_out, n_hidden)

  create a feed-forward network with n_in inputs,
    n_hidden hidden neurons and n_out output units
  init the network with small random weights

  until the termination condition is met
    for each <x,t> in training_examples

      //feed forward
      input the instance x to the net and
        calculate the output o_u for every unit u

      //propagate backwards
      for each network output unit k calculate the error
      for each hidden unit h calculate the error
      update each network weight w_ji
```

The error of the output units is calculated with the formula:

$$\delta_k = o_k(1 - o_k)(t_k - o_k) \tag{4.7}$$

where o_k is the resulting output by the feed forward and t_k is the target output of the training examples. The error of the hidden units differs, because all outputs have to be taken into consideration. The formula for the hidden units is:

$$\delta_k = o_k(1 - o_k) \sum_{k \in outputs} (w_{kh}\delta_k) \tag{4.8}$$

The weights w_{ji} are updated by

$$w_{ji} = w_{ji} + \Delta w_{ji} = w_{ji} + \eta \delta_j x_{ji} \tag{4.9}$$

where η is the learning rate.

Application in the RoboCup

The neural networks have different applications in the RoboCup community. For example they are used in the Sony Aibo League to control the motion on a given trajectory close to the physical limits [GSS07]. In this paper it is described how they use one neural network for the whole motion control and how they use one neural network for each hinge.

Rojas and Atkinson describe in [RA08] how they use a neural network for the game strategies. The game is classified with parameters, so new game situations can be appointed. The experiments were performed in 10-minutes games and every 30 seconds the game was stopped and an expert had to evaluate the situation and assign a fitness value. The algorithm Rojas and Atkinson are using is Backpropagation.

Evaluation

Artificial neural nets are powerful and the advantages have already been pointed out in the description. The main problem with the neural nets is that a set of training examples is needed. These training examples depend on the specific usage of the algorithm. For example neural nets can be used for a mapping from every state of the environment to a value giving an evaluation of the current situation. The goal of this work is to reimplement the decision making algorithm in order to compare it with the existing one. This can be done by a mapping of a state of the environment to a single action that fits best this current state. But these training examples have to be created by a human. And neither the rating of the environment nor the choice of the best action is trivial or well defined.

4.3.4 Reinforcement Learning

The main difference of reinforcement learning compared to other learning methods is that the agents learn through a reward. This reward can be given by a trainer that evaluates the state of the environment after an action or a sequence of actions have been performed. Another possibility to estimate the reward is by the agent itself. In many applications the reward is granted for a sequence of actions and tells if the goal has been reached or not. In any kind of games this goal is to win the game. Thus if the game is won the reward will be positive and if the game is lost the reward will be negative. In a draw it will be zero. The reward is the reason why...

> *"...reinforcement learning is much more focused on goal-directed learning from interaction than other approaches to machine learning." ([SB98], p. 3)*

The main difference between the reinforcement learning and supervised learning is correctness of the input. In the supervised training examples it is assured that the training examples have a correct training output for every training input. But in the reinforcement algorithm there can be actions that are not optimal for the global goal, but have a positive reward, because the state of the environment or the current outcome is good.

One general problem of reinforcement learning is that the actions an agent performs on the environment do not determine an immediate reward. Therefore the agent needs a model of the environment in order to predict the reward for the next state of the environment. A common method to model the environment is the Markov decision process (MDP). As described in [KLM96], the MDP consists of:

- a set of states S
- a set of actions A
- a reward function $R : S \times A \rightarrow \Re$ and
- a state transition function $TS \times A \rightarrow \Pi(S)$ and $\Pi(S)$ is the probability distribution of S

The most probable next state of the environment is described by the state transition function depending on the current state. This function is often used to estimate the maximum reward that can be gained after performing actions.

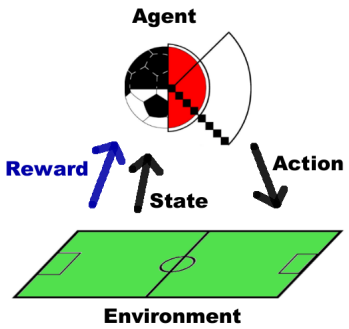

Figure 4.9: Interaction between an agent and the environment with reinforcement learning

An overview how the reinforcement learning is working can be seen in Figure 4.9. This figure shows the agent, performing actions in an environment. This

environment is observed by the agent, giving the agent a current state of the environment. The reward is a value that is assigned according to the current state of the environment. In this figure it is drawn as a number given by the environment, which is not always correct.

Example of an Algorithm

In the field of reinforcement learning three learning approaches can be identified. The dynamic programming ideas, Monte Carlo and temporal-difference (TD). The dynamic programming methods use the calculated states after the performance for the next decision. This following decision is based on a value function. The dynamic programming approach needs a model of all states for its performance. In comparison to that the Monte Carlo method does not need a complete model, because it learns through the whole trace of states. Thus experience is used for the update of the value function. [Lug02]

The TD approach is based on a combination of the Monte Carlo and the dynamic programming approach. Like Monte Carlo, TD can learn from raw experience and does not need a model of the environment. The analogy between TD and dynamic programming is that their estimate is based on other learning estimates without waiting for the final outcome [SB98]. The main character of the TD algorithm is that it is based on the difference between the current state of the environment and the next state. For every state of the environment an action is selected and performed in the environment. After this the reward is observed. Finally the function $V(s)$ is updated with the formula

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{4.10}$$

whereas α is the learning rate and γ is the discount factor. Both parameters are chosen in the interval $[0; 1]$. The learning rate is accountable for how much the current function V is adjusted. If the learning rate is low it takes more time to learn, but it is more accurate. The discount factor models the fact that the next state of the environment is worth less than the current state. The algorithm for the temporal difference is:

```
TD
  Init V(s) arbitrarily, P to the policy to be evaluated
  For each episode
    Init s
    For each step in the episode
      a = action given by P for s
      Perform action a
      Observe the reward and the next state s'
      V(s) = V(s) + alpha [r + gamma V(s') - V(s)]
      s = s'
```

This algorithm is an on-line learning algorithm which can be implemented in a fully incremental fashion. Other algorithms that are based on the TD are Sarsa and Q-Learning [SB98]. A design issue which is not covered by this algorithm is the dilemma of exploration. On one hand actions that had performed well in the past should be used in the present. But on the other hand new actions should be chosen, to find out if they are better.

Application in the RoboCup

Reinforcement learning is probably the most commonly used learning technique in the RoboCup. The reason is that agents can gain experience through the environment. One application of the Q-Learning algorithm is described in [KSL08]. It figures out how model-based reinforcement learning can be used in a complex domain. The domain is the simulated RoboCup soccer, especially the subtask of keeping the ball from the opponent.

The teams at the University of Osnabrück are focusing on reinforcement learning. They are developing new variants and practical algorithms for a complex domain in the simulation league as well as in the middle size league. Especially for the simulation league the reason for reinforcement learning is that the success or failure of the agent is provided by the environment. The long term goal of the Brainstormers - the 2D Simulation league team of the University of Osnabrück - is to have a team of learning agents, where the only parameter is "win the match" and the agents learn the appropriate behaviour. [RG08]

Evaluation

As already pointed out the reinforcement learning method is the only one that uses the interaction with the environment for adjusting the target function. The only problem that occurs during the learning phase is that the learning process takes a lot of runs. Tesauro [Tes95] described a backgammon game, trained with the temporal-difference (TD) algorithm. The goal of this trained game was to become a world-class player. For the training 1.5 million self-generated games where used until it was competitive with the best human players. The target function of this learning algorithm is a multilayer perceptron. A perceptron is an artificial neural net, which was first introduced by Rosenblatt [Ros88]. The complexity of this multilayer perceptron is another reason for the high number of trainings used.

4.4 The Resulting System

It is hard to find the most appropriate learning method for the RoboCup Soccer Simulation, because it is a platform that allows a lot of different learning approaches. Nevertheless a design for the implementation of the learning system is needed.

4.4.1 Target Function

The target function describes what will be learned through the experiments. In this work the accuracy of a single action for a certain state of environment is learned. This is realised by implementing a target function in every single action. This function is a polynomial function of third order, like:

$$f(x) = w_0 + w_1 * x + w_2 * x^2 + w_3 * x^3 \tag{4.11}$$

where x is the vector of the considered state of the environment. w_0, w_1, w_2 and w_3 are the weights that are adjusted during the training phase. The polynomial of three is chosen as this function has the possibility for an inflection point and extreme points. The advantage of this polynomial function compared to a linear function is the higher expressiveness. In comparison to a neural network the polynomial function has a shorter training time in order to get satisfying results.

The best action for a certain state of the environment is chosen by the *ActionSelector*. This *ActionSelector* has different pools of actions it chooses from. These pools are distinguished by the player types into the action pool for the goalie and the one for the other players. The pools for the field players are divided according to the different play modes, into the pool with the actions for the regular game with the ball, the one for the regular activities without the ball, the one for the standard situations in which the own team has the ball and the pool with the actions for defensive standard situations in which the opponent team has the ball. The different pools and the interaction with the *ActionSelector* is shown in Figure 4.10.

After the *ActionSelector* has chosen a pool, the state of environment is forwarded to all actions of this pool. For every action the approximated reward is calculated by the target function. The action with the highest value is chosen and executed. After this execution the weights of this action are adopted.

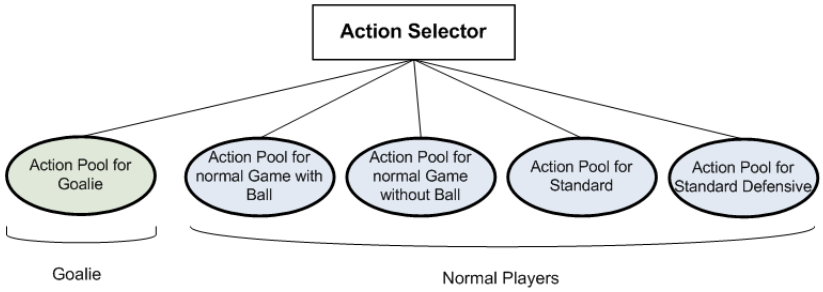

Figure 4.10: Different action pools for the players of the learning system

4.4.2 Learning Method

The advantages and disadvantages which have been already described in Section 4.3 are summarized in Table 4.1.

All described and analysed learning methods can be applied in the RoboCup Soccer Simulation League. But not all methods are equal useful for a certain application. The ambition of the implementation was to reimplement the decision making. As there are no current training examples available, a learning method without training examples was chosen. The available methods with this restriction are: genetic learning and reinforcement learning. Because the reinforcement learning has a better performance compared to the genetic learning it was decided to implement reinforcement learning. The main problem with reinforcement learning is the observation and evaluation of the environment. Since this can be done by the trainer who has a complete world model the disadvantage vanishes.

Environment

The simulation is set up as shown in Figure 4.11. In this figure the right part shows the agent that observes the environment and builds up its own incomplete *WorldModel*. From this *WorldModel* the current state of the environment is passed to the *LearningAlgorithm*. This *LearningAlgorithm* holds the value function that selects an action depending on the state of the environment. This action is passed to the *Execution*, that generates the performance of the action in the environment. For the reinforcement learning it is essential that the *LearningAlgorithm* gets a reward from the environment. As the RoboCup soccer server does not provide a reward by its own, a trainer is used. This trainer observes the environment and builds its *WorldModel*. This is a complete one, because the trainer can see everything

Learning Method	**Advantages**	**Disadvantages**
Decision Trees	understandable representation of the trees; good classification if the incoming data is noisy	it is hard to get a general idea of a big decision tree; training examples are needed
Genetic Learning	suitable for complex and interacting data	high computational power is needed; depends on the accuracy of the fitness function
Neural Nets	provides a possibility to learn complex patterns without learning the rules	not applicable for structured problems; training examples are needed
Reinforcement Learning	learns through an interaction with the environment	a possibility for the observation and evaluation has to exist

Table 4.1: Advantages and disadvantages of different learning methods

that takes place in the environment. With this *WorldModel* a reward is calculated by the trainer and passed to the agent.

The most important objects of the environment of the RoboCup soccer server are eleven agents for each team and the ball. Every object has a position and a velocity. Beside that every agent has a facing direction of its head and body. Another important information about each agent includes the view mode, the current speed and the available stamina. One possibility for the learning algorithm is to take all this information as an input. This is not feasible, because the model of the world for an agent is incomplete. If all elements of the world are considered the parameters of the input would explode. Another problem is that a higher number of input values has a high impact on the runtime of the training. Thus a simplification of the environment is used. This simplification includes the position of the agent itself, its velocity and its stamina. The ball can be seen as the main object in the soccer environment. Thus in the simplified state of the environment only the relative position of the ball and its velocity to the agent are considered. In the area around the agent the four closest own players and the two closest opponent players are considered. This simplified environment still has 22 different parameters to be described.

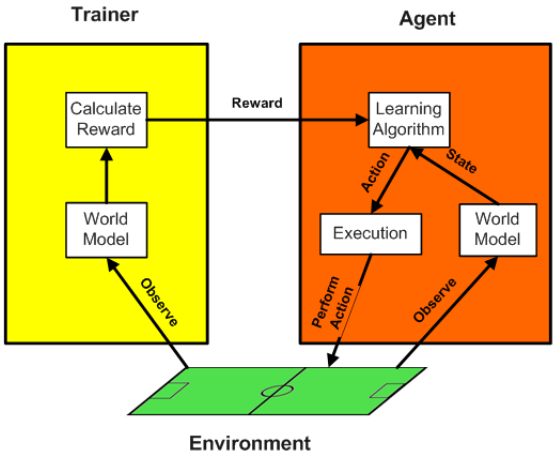

Figure 4.11: Interaction between the agent, the trainer and the environment

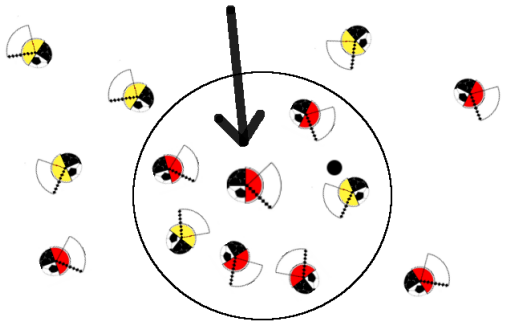

Figure 4.12: Simplified environment of the agent

In Figure 4.12 a simplified environment is shown. It points out the players of the two teams, the red ones and the yellow ones. The black dot symbolises the ball. The simplified environment of the red agent, focused by the arrow, contains everything that is in the circle.

Learning Actions

The basic actions an agent can perform have already been described in Section 2.3.2. These actions are just the commands the agent can send to the server. Case studies have been made to apply a decision making algorithm to these basic actions [Kle99], but the outcome was not satisfying. In these studies the created system was based on the learning of high level actions.

This is similar to the approach that is implemented in the team *KickOff-*

TUG. The actions used in the learning decision making are high level actions. These *learning-high-level* actions are similar to the *planning-high-level* actions. Both are executed by the *Execution* instance. The main difference between a learning- and a planning-high-level action is that the learning action has a target function, while the planning action has preconditions and postconditions. In order to give the players the possibility for an offensive and a defensive behaviour actions like *RunToOwnGoal* and *RunToEnemyGoal* were introduced in the learning system.

Learning Algorithm

The learning algorithm implemented in this context is based on the reinforcement Q-learning. The algorithm introduces an action value function, that takes a state of environment and an action as an input and produces a value how adequate this action is for the state as an output. In every run the Q-function is adjusted according to the formula:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.12}$$

where α is the learning rate. It is set between 0 and 1 and describes how much the function is updated. A learning rate of 0 tells that the Q-function is never updated. γ is the discount factor, that is also chosen between 0 and 1. It considers the fact that the future value is less worth than the current. The reward of the action is taken into account with the variable r_t. Furthermore the estimation of the reward for the next possible action is calculated and assigned to max_a.

The algorithm for the Q-learning is an extension of the temporal-difference algorithm described in Section 4.3.4. The main difference between these two algorithms is the updating of the value function. The algorithm of Q-learning written in pseudo code is:

```
Q-Learning

  Init Q(s,a)
  For each episode
    Init s
    For each step in the episode
      choose an action a using Q depending on s
      Perform action a
      Observe the reward and the next state s'
      Q(s,a) = Q(s,a) + alpha [r + gamma max_a Q(s',a) - Q(s,a)]
      s = s'
```

As already mentioned the Q-function is a mapping from every state of the environment to the best action. In the implementation this Q-function is divided. Every action implements a function that takes a state of environment as an input and returns a value indicating how accurate this action is according to this input state. The Q-function asks every action that can be taken in the current state, how accurate it is and returns the one with the highest value. This selected action is performed in the environment using the *Execution*.

The reward function defines the task of the learning process. It maps the actual state of the environment to a single reward number to specify the state [SB98]. In the implementation the reward is given by the trainer, which is the only agent that has a complete model of the world and can perform a reasonable calculation of the reward. A reward is given if

- the own team has scored a goal (reward of 100)
- the opponent team has scored a goal (reward of -100)
- the own team is in ball possession (reward of 0.1)
- the opponent team is in ball possession (reward of -0.5)

The reward for the goals is high, because the achievement of a goal has a high impact on the result of the game - if it is won or lost. The problem of the reward of the goal is that a goal score occurs seldom and infrequently. This leads to the idea to evaluate the current situation of the game. This evaluation is done by the analysis which team has the ball. All values used for the reward are just the first initialization and can be easily adjusted. This is important for the different experiments which need to be taken for the analysis of the learning implementation.

4.4.3 Design Modules

In this section the different design issues mentioned in Section 4.2.5 are described in the context of this work.

Performance System:
In every cycle the agent should perform an action. For the selection of the action the Q-function is used, in order to get the best action for the current state of the environment. But depending on the parameters of the agent, this selection can also result in a random action. This is used to try new actions for a state.

Analyser:
The analysis how appropriate the action was, is done by the trainer who

passes the reward to the agent. A more detailed description of the trainer and the reward can be found in Section 4.4.2 (*Environment*).

Generaliser:
The duty of the *Generaliser* is to update the target function. The Formula 4.12 on page 51 is used for that. This goes along with the description of the algorithm which is placed in Section 4.4.2 (*Learning Algorithm*).

Experiment Generator:
The server can be seen as the experiment generator, because the server increments the cycles. Beside that it simulates the environment in which the actions are performed. Thus in each cycle the target function is updated.

Chapter 5

Analysis - Learning versus Planning

The problem in the RoboCup soccer simulation league is that there are approximately

$$346000000000000000000000000000000000000000000$$
$$000000000000000000000000000000000000000000000$$

different positions where a player or the ball can be. This value is calculated with the formula $((2 * 52) * (2 * 34))^n$. As the field has the length $2 * 52$ and the width $2 * 34$ the area for one position is one square meter. n is the number of considered instances like players and the ball. In a normal game the number for n is 23. Beside the positions, there are more parameters to be considered - for example the stamina and the velocity.

Based on this information a decision must be derived by every agent. If all commands including the parameterised ones are examined, there are about 300^{11} different commands an agent can decide upon. This problem is handled in the RoboCup Soccer Simulation League team *KickOffTUG* by a decision making that is based on a planning algorithm. In order to evaluate this planning system in comparison to a machine learning approach a decision making based on reinforcement learning has been implemented. These two different approaches are compared in this chapter.

5.1 Basic Analysis

In this section basic points of the two different systems are outlined and the further analysis is described.

5.1.1 Planning

The planning algorithm was implemented by Michael Reip and Stephan Gspandl. It was already described in Chapter 3 and in [Gsp07] and [Rei07]. The main advantage of the planning algorithm is that it is easy to understand for humans and thus the human responsible can reconstruct the actions of the agent. The main advantage of the planning system is that it does not need any training time, like learning algorithms do.

5.1.2 Learning

In comparison to planning, the learning algorithm has been implemented during the master studies. The first implementation which is described in Chapter 4 was quite successful. The idea behind it was to learn from games only with the reward function that returns a positive or negative reward if the ball is hold by the own team or opponent team or if a goal is scored.

In order to improve the learning system two additional rewards were introduced. The first reward is based on the idea that the players should try to get the ball. Thus if the distance between the player and the ball decreases a small reward is assigned. Experiments with rewards between 0.1 and 1 were performed. The experiments showed that the values near the min and max limit are suboptimal and a suitable value lies in the middle. Another improvement should address the problem that the learning system is too defensive. To gain better results the trainer calculates a reward, based on the x-coordinate of the ball. If the ball is on the center line the reward is 0. If it is on the own half a small negative reward and in the opponent half a positive reward is sent to the player.

5.1.3 Way of Analysis

Every system can be described with the PEAS-method described in [RN03]. PEAS stands for:

- **P**erformance
- **E**nvironment
- **A**ctuators
- **S**ensors

These components, especially the environment, the actuators and the sensors, have already been described in Chapter 2. But now the focus lies on the measurement of the performance. The setup for the performance tests is the

simulation of games. The main advantage of the RoboCup Soccer Simulation league is the possibility of simulating games and thus these simulations are used for the analysis. After every world championship the binary code of every team is published. Thus for simulating games the top teams can be used to play against the team *KickOffTUG*.

By now only the decision that the performance is measured by playing games is taken. The considered components of the systems are divided into four main sections.

1. Games (Section 5.2): As the performance measurement is based on games, these games are observed.
2. Situations (Section 5.3): In every game standard situations occur. Comparing these situations is another possibility to demonstrate the differences between the systems.
3. Player Types (Section 5.4): The players of a team can be divided into different types. In the analysis these player types are described and the differences in the implementation of the planning and the learning system are outlined.
4. Factors (Section 5.5): Every player is influenced by critical factors. These factors are pointed out and analysed.

A tool that helped during the analysis phase is the Loganalyser [Log04]. This is a tool for the visualisation of the RoboCup log files.

5.2 Analysis of Games

The main possibility how the different approaches can be analysed is playing games. In this work there are four different kinds of games that are evaluated. Games with the examined instance either with planning or learning. Games against each other and furthermore games against the reigning World Champion - the Brainstormers from Osnabrück.

5.2.1 Games with the Same Instance

The games with the same instance of the system are useful just to a certain extension. In these games the relation of the strikers to the defenders can be analysed. Beside that it can be evaluated if a team is more offensive - thus a game against the same system has a high rate of goals. Or if a team plays more defensive with a low ratio of goals.

The main difference between the planning system and the learning system in this kind of games was that the players of the learning system got stuck in the middle of the field concerning the length as well as the width. The result of the games learning against learning normally ended low - like 0:0, 1:0 or 1:1. In comparison to that the games with two instances of the planning system were wider in both dimensions of length and width. This is based on the different types of players that are used in the planning system, but not in the learning system.

To sum up the gained awareness out of these games: it is interesting to simulate these kind of games in order to demonstrate the difference of the attacking and defending part. But if a team is balanced no important knowledge for the analysis can be extracted.

5.2.2 Games Learning versus Planning

The games of the learning system against the planning system are the most interesting ones as these both systems should be evaluated against each other. The first thing that catches one's eye is that the learning system is much more ball focused. The reason is the reward function that assigns a reward connected to the ball. In comparison to this observation the planning system has plans that tells the agents to stand there if the ball is not in the area around. This is indispensable for the defense zone.

Another ambition of these games was to improve the learning model. Thus, for example, the reward function was adopted in order to take the x-coordinate of the ball into account. With this additional feature a more offensive behaviour should be enforced. This has been worked out but is still not feasible to make the game broader. Thus the learning system will always try to make its way through the middle.

The outcome of these games was normally a win by the planning system. The problem of the learning model was not the normal play, but in the end of the game when the learning agents run out of stamina. At that time the planning system had more resources to score a goal.

5.2.3 Games with Brainstormers

The Brainstormers from the University of Osnabrück are the reigning World Champion and thus a strong, but welcomed opponent for the test games. The system of the Brainstormers is based on a reinforcement learning algorithm as well [RG08]. Furthermore their strategy for covering the enemies is based on an assignment of every opponent player to a player of the team of the Brainstormers.

In the games between the Brainstormers and the planning system it could be observed that the Brainstormers normally attack with passes through the center. In these attacking situations the planning team has problems with defense as the defending formation is to slow to react. In comparison to this the games between the learning system and the Brainstormers had an unexpected effect on the formation of the World Champion. As the learning system is focused on the ball nearly all agents run to the ball. The Brainstormers followed because their enemies to cover where all close to the ball. Such an example can be seen in Figure 5.1. In this example the learning system is the yellow team playing from left to right and the Brainstormers are the red team.

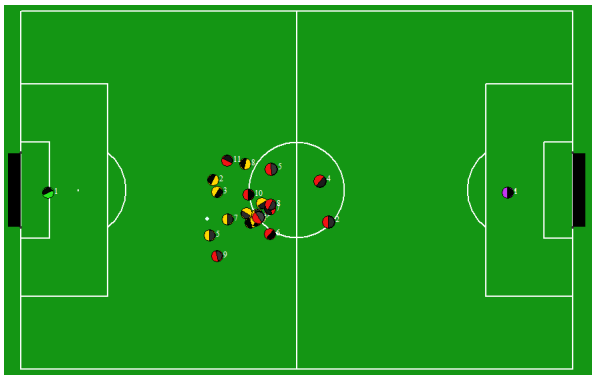

Figure 5.1: Screenshot from a game between the learning system and the Brainstormers

5.3 Standard Situations

In this section different situations that occur during a game are described. Typical situations are the kick-off, a throw in, the normal playing with the ball and the positioning without the ball.

5.3.1 Kick-Off

The kick-off is the begin of every game. If the opponent team gets the ball right after the kick-off, this team can go into an offensive game and has a higher chance to score a goal.

In the planning system every player has a plan for all standard situations. In these plans it can be identified, for example that always Player 10 and Player 11 perform the kick-off. This is feasible if all players are in the game.

But if one of these both kick-off-players is not able to kick the ball before the kick-off-time, the kick-off is lost and the other team gets the kick-off.

In the learning system every player is able to perform the kick-off. Thus it is not important that all players are on the field or working. The disadvantage of this method is that all players start to move before the kick-off. Most of these motions are unnecessary and decrease the stamina of the agents. This does not effect the agents in the current moment, but in the end of the game when the stamina will be low.

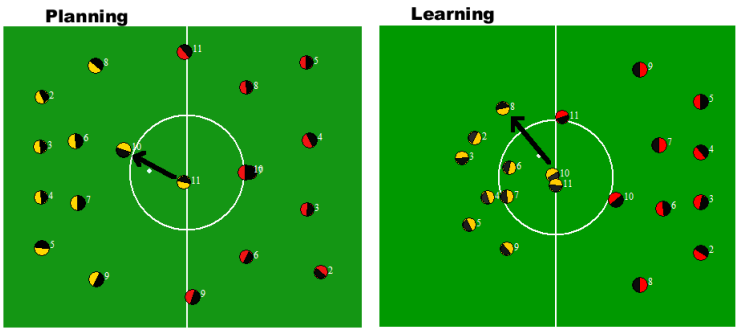

Figure 5.2: A sample kick-off for the planning and the learning system

In Figure 5.2 a sample kick-off of both systems is shown. In the left picture the planning system's agents are colored yellow and perform the kick-off. It can be seen that all the other players are still in the formation. In comparison to that the learning system's agents in the right picture in yellow, too, have left their formation and start an offensive game.

Both types of the kick-off are not optimal. While the learning system has too much flexibility the planning system is too stiff. The perfect way would be in the middle. At least the defenders should stay on their position, but it should be possible to perform the kick-off with another player than Player 11, if it is necessary.

5.3.2 Throw-In

The throw-in differs in the RoboCup 2D soccer simulation league in comparison to the real soccer. In the simulation league it is performed with a normal *kick* command, because there is no third dimension so robots cannot use their hands.

In most cases a throw-in is not decisive for the outcome of a game. The problem with the throw-in is that the robot has to run around the ball in order to kick it. Otherwise it can happen that the agent moves the ball over the outline again and then the other team gets the ball.

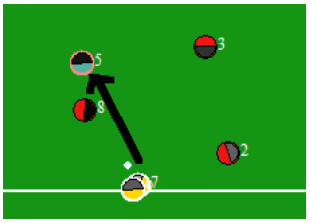

Figure 5.3: A sample throw-in for the learning system

The behaviour of the learning and the planning system is similar to the behaviour in any other standard situation. In the planning system the player that performs the throw-in is defined in the plan while in the learning system any player can execute the throw-in. In Figure 5.3 it is the learning system that has to throw in the ball. It can be seen that two of the yellow learning players are at the ball in order to perform the throw in. This is a result of the missing assignment from the players to the standard situation. Obviously it is not optimal that both players are running to the ball, but it does not have a big effect on the game. The more important issue is that the throw-in results in a successful pass to Player 5.

5.3.3 Free-Kick

A free-kick can lead to a goal if it is close to the opponent penalty area. In Figure 5.4 an example for a free kick is shown. This free-kick is performed by the planning system and the learning system is defending. First the analysis of the attack: Player 8 performs the free-kick. He has the choice either to pass to another player of the own team or to shoot directly a goal. The first choice is not applicable because just one other player from the same team (Player 7) is around, who is not in a position to be safely passed to. The direct way to the goal is not completely free, but the chance for scoring a goal is better with a direct shot.

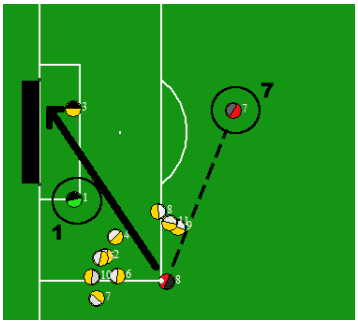

Figure 5.4: A sample free-kick performed by the planning system

The defense of the learning team is fine, but it could be better. A lot of players are defending and positioning themselves around the free-kick point. Unfortunately the line from the ball to the goal is free. An improvement would be if the goalie (Player 1) moves more into the middle of the goal because the closer goal post is already covered by the players. Another problem is that the Player 7 from the opponent team is uncovered.

5.3.4 Goal Kick

The goal kick is performed after the opponent team has tried to score a goal, but missed it and instead has shot into the out. This standard situation is normally performed by the goalie. The other players have the task to position themselves in a favourable strategic way.

If the play mode is *goal-kick* the standard plans tell the players - except the goalie - to position themselves in a strategic position. This strategic position is calculated with a potential field. If the goalie finds a pass partner the pass to this player is performed. The problem is, if this pass partner is not far enough away from the opponent players the goalie performs the action *clearBall*. In most cases with this action the goalie shoots the ball into the out and the opponent team gets a corner. It is obvious that this is a solution that can be good if the opponent team plays in an offensive way. But against a defensively prepared team this is rarely an optimal choice.

The goal kick in the learning system is not always performed by the goalie. Every agent that is close to the ball can decide to perform an action to kick the ball. This is an advantage if the opponent team is offensive and the goalie stays in the goal. But unfortunately the games have shown that the outcome of the goal kicks is similar to the outcome of the planning system. Thus the ball is lost too often after the goal kick.

5.3.5 Corner

With a corner kick the possibility to score a goal is quite high for the attacking team. Thus the defending team has to defend well in order to block the attack. In Figure 5.5 the learning system is playing in yellow dresses against the planning system having red dresses. The situation results from a one-player attack. Thus only Player 11 is in the opponent half. This player is responsible to perform the corner kick. All the other players of the red planning team run to their strategic position.

The problem in this situation lies in the calculation of the strategic position. Instead of moving to the direction of the opponent goal - like the black arrow in the figure shows, the agents are moving back into the direction of the own

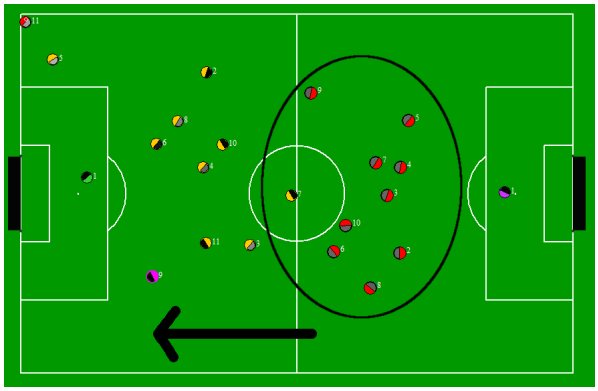

Figure 5.5: Corner performed by the planning system in a game against the learning system

goal. This decision is based on the potential field. This potential field is based on the positions of the players and the ball and has two different possibilities to calculate the strategic position. One for the defensive behaviour and one for the offensive behaviour, depending on the current situation on the field. Although the potential field works fine in most cases, in this one it fails.

As the current situation in Figure 5.5 is observed to be offensive, the implementation of the potential field results in all opponent players being repulsive as well as the own players. This should ensure that the players are free and well distributed over the field. But the players started already in the own half and thus they distributed themselves in this area. The attraction of the ball was not strong enough in this case. The learning system which is in the defending position is placed in an adequate way. Only the goalie should move backwards.

5.4 Player Types

The players in the planning system are divided into goalie, defender, midfield and striker. It makes sense to split the players into these types in order to ensure different behaviours on the different positions. In the learning system the player types are just split into the goalie and normal players.

5.4.1 Goalie

The goalie is different from all other players. First of all only one goalie can be started for a team. Beside that a goalie can perform other commands

compared to normal players - like *catch*. Thus it makes sense for all decision making systems to split the goalie from the rest of the team.

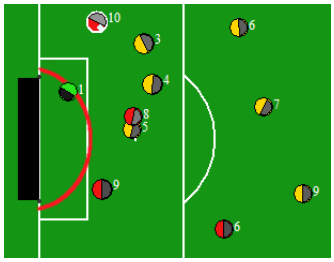

Figure 5.6: Display of the predominant moving path of the goalie

The goalie in the planning system moves along an ellipse like Figure 5.6 shows. This is not optimal because in some situations the strategic position is not on this ellipse. Furthermore if an attacker overplays the defenders and strikes alone towards the goal, the goalie should leave the ellipse and should run to the attacker. This is the only possibility to defend the goal because the angle between the goal and the attacker shrinks.

In comparison to this the goalie of the learning system has a pool of actions he can choose upon. At the moment this pool consists of *catchBall*, *findBall*, *intercept* and *turnAround*. This is not a lot, but it is feasible. These commands cover all actions a goalie should perform. During an example game the actions the goalie performed where counted. In Table 5.1 it can be seen that the action *intercept* was performed most frequently. The way this high-level action is executed depends on the distance to the ball and on the player type. If the ball is close and the player is connected as a goalie, it is checked if a catch command is suitable. If so it is executed otherwise if the ball is further away, the player runs toward the calculated intersection point of the ball and itself.

Action	Times performed
intercept	1888
catchBall	1755
turnAround	1710
findBall	643

Table 5.1: High-level actions performed by the goalie during a game

The frequency the actions *turnAround* and *findBall* are chosen depends on the offensiveness of the game. This is based on the fact that the goalie performs these actions if the ball is far away. In this context the word "far" means at the center line.

5.4.2 Defender

The player type *defender* only exists in the planning system. The defenders are placed in a 4-formation. The idea of this formation is copied from the real soccer tactics. In a lot of situations this works fine to defend the goal. But during the studies some problems occurred. For example that sometimes the central players are placed according to the ball and not in connection with the opponent players. Thus it can appear that the opponent players are uncovered in the defender zone. Another problem is that the defenders can be overplayed. Figure 5.7 shows such an overplay, where the defenders formation is too slow to react and furthermore the defenders are not on an optimal place. *Player 5* should cover opponent *Player 9*. But the position of *Player 5* is calculated in dependency only to the ball without any respect to the opponent players.

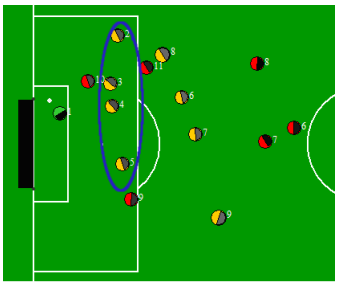

Figure 5.7: Overplay of the defenders of the planning system

An experiment to gain a more defensive behaviour in the learning system did not work well. In this experiment the reward function assigned a reward according to the ball. The effect was that all own players positioned themselves in the own penalty area. This is neither a good behaviour for defenders during the game nor for other players.

5.4.3 Midfield

The midfield is the type of the players that have to run a lot. This is based on the fact that they have to support the strikers if the game is offensive, but also support the defenders, when the game is in a defensive phase. In the learning system the midfield has the same homogenous player type as any other field player. In the planning system heterogeneous player types are assigned by the coach.

Another thing to point out is that the planning system distinguishes the midfield players into the central midfield players and the wing players. This is based on the idea that the wing players support an attack by center passes

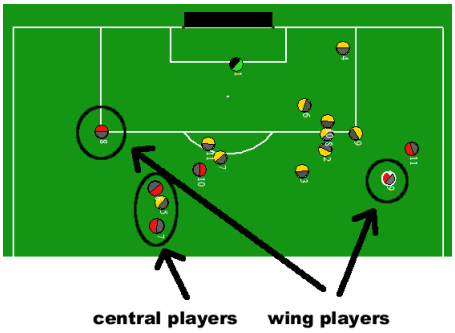

Figure 5.8: Attack of the midfield

from the outer side of the field. In contrast to this the central midfield players support an attack of the wing players in the center of the field. An example attack is outlined in Figure 5.8. There can be seen that the wing players are on the outside and the central players support the strikers. The wing player with the number 9 holds the ball until the Player 11 runs toward the goal, then a running pass is played in order to overplay the defenders.

5.4.4 Striker

The main goal of a striker is to score a goal. In the planning system the plans describe the possibility for players to score a goal. Compared to that in the learning system every player, except the goalie, has all possible actions in the action pool he can choose among. Thus in the learning system every player is able to score a goal. But this is just in theory - in order to shoot a goal the player has to be in the area around the goal.

In the planning system the action *scoreGoal* is in all plans, but the position is another one. The plans and thus as well the preference of a striker is a completely different one in comparison to a defender. For a striker the main aim is to score a goal and this is reflected in its plan. Among the players of the learning system the player which is close to the goal shoots. Although the player type is not assigned in the learning system the behaviour is adapted.

5.5 Analysis of Influencing Factors

There are different factors that influence a game or occur during a game. Such an influencing factor would be the formation of the team and an example for a factor that occurs during a game is the stamina of the agents. In this section some of these factors are examined in respect to both systems - the planning and the learning one.

5.5.1 Formation

Although the formation of the players is the same in the planning system and in the learning system, it is an important point to discuss. The standard formation of the team *KickOffTUG* is 4-4-2. Thus in the planning system the team plays with four defenders, four midfield players and two strikers. In the learning system the formation is just for the setup and for some standard situations like a kick-off.

In former days the *KickOffTUG* team had three lines of three players and one striker. This did not work because cross passes overplayed these lines. A more detailed description of the already introduced and analysed formations can be found in [Zeh07]. This work outlines that the current formation is the most applicable one for the team.

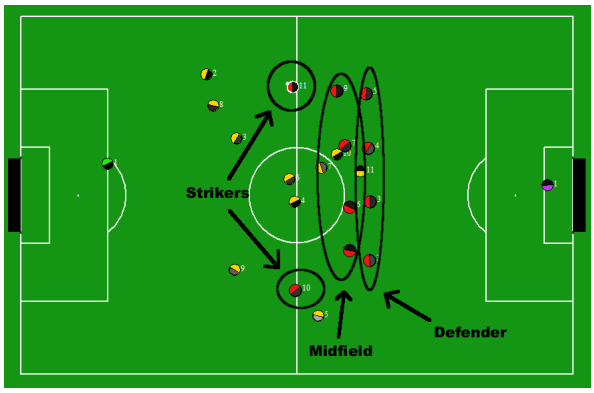

Figure 5.9: Analysis of the formation during a game

During a game the formation of the planning system can be observed. This can be seen in Figure 5.9. In this game the planning system is in red playing from right to left and the learning system plays in yellow. Although both systems are playing with the formation 4-4-2, it can only be recognised in the planning system - like the figure shows. The agents of the learning system only perform actions that have a limited connection to the formation and thus the formation of the learning system cannot be identified. This makes the team more flexible if the opponent team assigns every own player to an opponent player.

5.5.2 Stamina

One of the main differences between the learning system and the planning system is the stamina model. While the planning system has a stamina implemented the learning system has the stamina just as a parameter in

the considered state of the environment. Thus the stamina of the planning system is intentionally restricted with a value of 2500. On the contrary the learning agents do not reserve stamina for actions. How the different staminas look like during a game can be seen in Figure 5.10 for the planning system and in Figure 5.11 for the learning one.

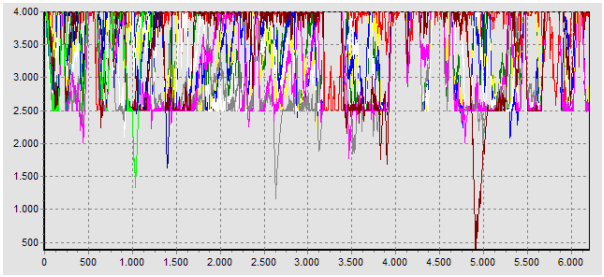

Figure 5.10: Diagram of the staminas of the planning system during a game

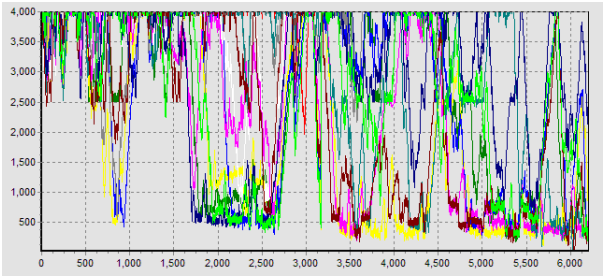

Figure 5.11: Diagram of the staminas of the learning system during a game

The only way the stamina is taken into account in the learning system is as a parameter in the state of the environment. Like every parameter this has a target function with weights that are adjusted after every decision. The disadvantage of the learning system concerning the stamina is that if the stamina is low the agents can not win duels. But on the other hand the agents appear more present on the field.

5.5.3 Basic Actions

The basic actions are the commands that are sent to the server. These commands are: *catch, dash, kick, say, turn, turn-neck* and *change-view*. A description of these commands can be found in Section 2.3.2.

The command *say* is used for the strategy and to exchange information between the players. As this is isolated from the decision making algorithm and in both the learning and the planning system the performance of the *say*

command is nearly the same. A player normally executes the say command in every cycle.

The *catch* can only be performed by the goalie. During a session of games it was observed which commands where performed by the goalie. The only important commands are *catch, dash* and *kick*. The counter of these basic actions can be identified in Table 5.2. These values are just an average normalized to a game with 6000 cycles. The *dash* command is the one that is performed most often. This is based on the fact that the goalie has to position itself often. This explains the low frequency of the other two actions *catch* and *kick*. The *catch* command as well as the *kick* is only performed if the ball is close to the goalie.

	Learning Goalie	Planning Goalie
dash	1263	2387
catch	49	23
kick	31	19

Table 5.2: Counter of the basic actions that were performed by the goalies

For the other players only the commands *kick* and *dash* are compared. This is based on the idea that only these commands are representative because with these commands the agents can kick the ball and move around. To analyse the difference between the planning and the learning system all main player types, defenders, midfield and strikers are considered. The results are listed in Table 5.3.

	Defender		Midfield		Strikers	
	Learning	Planning	Learning	Planning	Learning	Planning
kick	38	25	120	124	180	215
dash	2386	1805	2351	2299	2687	2246

Table 5.3: Counter of the basic actions that were performed by the different player types

From this table it can be identified that the defenders seldom kick the ball. As the main goal of a defender is to reduce the space for the opponent players they do not often have the ball. Furthermore the defenders try to get rid of the ball and pass it to the midfield in order to facilitate an attack. The frequency of the performance of the *kick* command for the midfield is higher. These players try to pass if it is possible, while the strikers dribble and shoot to the goal. This is just the basic intention of these players, certainly they all can perform other actions if the situation requires a different behaviour.

On the field there is just one ball and 22 players. This justifies the different

frequency between the commands *dash* and *kick*. As the *kick* command is only possible if the player has the ball, the *dash* command needs to be performed to get the ball in most situations.

The *dash* is performed more often by the learning system than by the planning system. The reason therefore lies in the stamina model of the planning agents. For the learning agents it is no problem to consume all their available stamina as described in Section 5.5.2.

5.5.4 Y-Position of the Players

The coordination system of a soccer field defines the *x-axis* between the two goals. The other one orthogonal to the *x-axis* is called the *y-axis*. Thus the analysis of the width of a play can be done by considering the y-position. The origin of both axes lies in the middle of the field - at the kick-off point.

The y-position of the players of the planning system over a whole game is drawn in Figure 5.12 with the cycles on the horizontal axis and the width of the field on the vertical axis. In this diagram it can be identified that nearly all players are moving from one part of the field to the other at the same moment. Only the player with the red line stays close to the center of the field. This line belongs to the goalie who does not need to move away from his goal area.

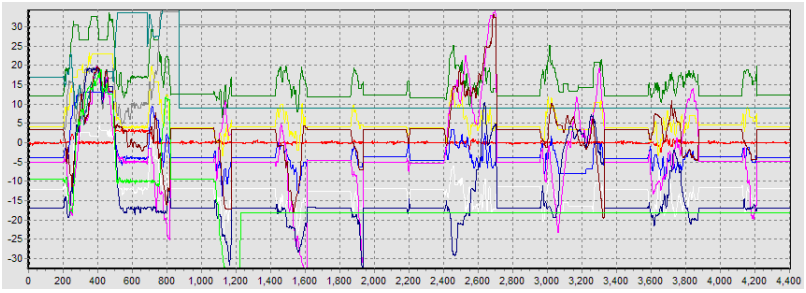

Figure 5.12: Diagram of the y-position of the planning system during a game

Figure 5.13 charts the y-position of players of the learning system. In the learning system this position does not have any regularity. This makes it harder for the opponent team to assign a part of the field to certain players for the covering. Similar to the planning system the red line which belongs to the goalie is close to the center line of the diagram. Another interesting point is that in the beginning of the game all players are on the upper part of the field. The reason for this was a throw-in which was blocked for several times. Furthermore all lines are closer to the axis which means that the play is more centralized.

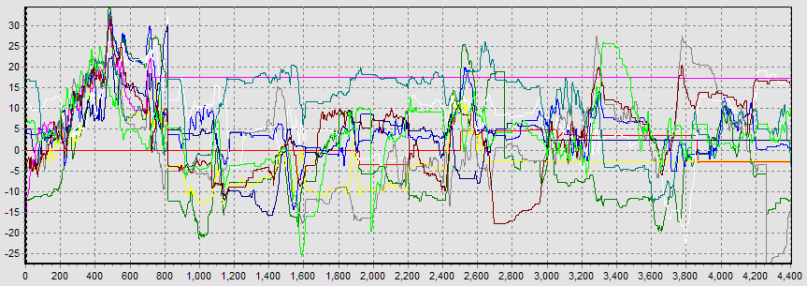

Figure 5.13: Diagram of the y-position of the learning system during a game

If a team is playing with a narrow strategy its play is based on short passes which have to be very accurate in order to reach the pass partner. If the game is more distributed every player has more space and thus basic actions like dribbling have a higher possibility to succeed. If the team chooses a narrow game - thus the players are close to the y-axis an opponent team often tries to score a goal over the wing.

Chapter 6

Summary

In this work two different approaches for the decision making process are analysed. The selected environment is *KickOffTUG* - the RoboCup 2D Simulation League team from Graz, University of Technology.

In this team the current decision making algorithm is implemented through a planning algorithm, which is described and evaluated. The learning system, which is compared to the existing planning system, was implemented during the studies for this work. This machine learning approach is based on the reinforcement Q-learning algorithm, with a target function which is specific for each single action. The target function takes a state of the environment as an input and produces a value indicating how appropriate the action is for this state. A certain action selector chooses the best action according to the player type, the play mode and the current state of the environment.

The analysis in Chapter 5 is based on simulated games between different opponents. In these games the regular flow of the game can be observed. This observation shows that the planning system sticks to its formation while the learning system is more focused to the ball. Another aspect to analyse the different decision making approaches is to evaluate the behaviour in the standard situations that occur during a game. These standard situations include: kick-off, throw-in, free-kick, goal-kick and corner. This analysis points out that the learning system is more flexible in these situation. In comparison to that the planning system is more organised and static. Another point that can be outlined in the analysis of the games is that the planning system is based on different player types. These player types are implemented in the planning system, but disregarded in the learning approach. Beside that there are factors that have an impact of the outcome of a game, for example the stamina. The stamina model of the planning approach is the reason why the planning system scores more goals in the end of a game compared to the learning system. In contrast to this is the

learning system more active in the beginning of the game.

The formation of planning system is the same as in learning system 4-4-2. But during a game this formation is only kept by the planning system. This has the advantage to cover players and space. In comparison to that the learning system has the advantage to react more flexible. The analysis in 5 shows, both approaches, planning and machine learning, are appropriate for the decision making process of a multi-agent system.

6.1 Future Work

The future work should focus on the evaluation of the high level actions. With an increasing performance of these actions the whole team would improve its competitiveness. For the most commonly used high level actions it would be suitable to implement a second approach with machine learning. The already implemented Q-learning algorithm can be used for this purpose.

Considering decision making, the planning system is more suitable for testing the high level actions. In this period there is no need to perform trainings for the decision making of the learning system. Including the improved high level functions a new training phase can start. A further improvement of the learning system could by reached when different various player types are implemented with the planning system as a paradigm.

Bibliography

[Bis02] Bishop, Christoph M.: *Pattern Recognition and Machine Learning*. Wiley, 2002

[BK02] de Boer, Remco ; Kok, Jelle: *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*. The Netherlands, University of Amsterdam, Diplomarbeit, February 2002

[BN95] Benson, Scott ; Nilsson, Nils J.: Reacting, Planning, and Learning in an Autonomous Agent. In: Furukawa, K. (Hrsg.) ; Michie, Donald (Hrsg.) ; Muggleton, S. (Hrsg.): *Machine Intelligence 14 - Applied Machine Intelligence*, Oxford University Press, 1995, S. 29–64

[BT90] Bailey, David L. ; Thompson, Donna: Developing neural-network applications. In: *AI Expert* 5 (1990), Nr. 9, S. 34–41

[CFH+02] Cheny, Mao ; Foroughi, Ehsan ; Heintz, Fredrik ; Huangy, ZhanXiang ; Kapetanakis, Spiros ; Kostiadis, Kostas ; Kummeneje, Johan ; Noda, Itsuki ; Obst, Oliver ; Riley, Pat ; Steffens, Timo ; Wangy, Yi ; Yin, Xiang: *RoboCup Soccer Server Manual*, 2002

[Cru07] *CruiseControl Home*. Online at http://cruisecontrol .sourceforge.net/. 2007. – last visited 17 October 2007

[DHVW02] Drücker, Christian ; Hüber, Sebastian ; Visser, Ubbo ; Weland, Hans-Georg: "As time goes by" - Using time series based decision tree induction to analyse the behaviour of opponent players. In: *RoboCup 2001: Robot Soccer World Cup V*, Springer-Verlag, 2002, S. 325–330

[EET01] Egly, Uwe ; Eiter, Thomas ; Tompits, Hans. *Skriptum zur Lehrveranstaltung Wissensbasierte Systeme*. Technische Universität Wien. 2001

[Elk93] Elkan, Charles: The paradoxical success of fuzzy logic. In: *In Proceedings of the Eleventh National Conference of Artificial Intelligence* (1993), S. 689–703

[EWK+08] Endert, Holger ; Wetzker, Robert ; Karbe, Thomas ; Hessler, Axel ; Brossmann, Felix: The Dainamite 2007 Team Description. In: *Proceedings of RoboCup 2007: Robot Soccer World Cup XI*, Springer-Verlag, 2008. – To appear

[Fer01] Ferber, Jacques: *Multiagentensysteme - Eine Einführung in die Verteilte Künstliche Intelligenz*. Addison-Wesley, 2001

[Gsp07] Gspandl, Stephan: *KickOffTUG: Eine KI-Lehr- und Forschungsplattform*. Graz, Austria, Graz University of Technology, Diplomarbeit, October 2007

[GSS07] Grasemann, Uli ; Stronger, Daniel ; Stone, Peter: A Neural Network-Based Approach to Robot Motion Control. In: *RoboCup 2002, Lecture Notes in Artificial Intelligence*, 2007

[HA50] Hilbert, David ; Ackermann, W.: *Principles of Mathematical Logic*. Chelsea, 1950

[Hau07] Haun, Matthias: *Handbuch Robotik*. Springer, 2007

[Joo07] *Joomla*. Online at http://www.joomla.de/. 2007. – last visited 17 October 2007

[kic07] *KickOffTUG*. Online at http://kickofftug.tugraz.at. 2007. – last visited 4 January 2008

[Kle99] de Klepper, Nathan. *Genetic Programming with High-Level Functions in the RoboCup Domain*. 1999

[KLM96] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement Learning: A Survey. In: *Journal of Artificial Intelligence Research* 4 (1996), S. 237–285

[Koh88] Kohonen, Teuvo: *Self Organization and Associative Memory*. Second Edition. Springer, 1988

[KSL08] Kalyanakrishnan, Shivaram ; Stone, Peter ; Liu, Yaxin: Model-based Reinforcement Learning in a Complex Domain. In: *Proceedings of RoboCup 2007: Robot Soccer World Cup XI*, Springer-Verlag, 2008. – To appear

[LC01] Lämmel, Uwe ; Cleve, Jürgen: *Künstliche Intelligenz: Lehr- und Übungsbuch*. Fachbuchverlag Leipzig, 2001

[LHF+97] Luke, Sean ; Hohn, Charles ; Farris, Jonathan ; Jackson, Gary ; Hendler, James: Co-evolving soccer softbot team co-

ordination with genetic programming. In: *Proceedings of the RoboCup-97 Workshop at the 15th International Joint Conference on Artificial Intelligence*, 1997, S. 398–411

[Log04] Andraz Bezek: *A RoboCup visualization and analysis tool*. Online at http://dis.ijs.si/andraz/logalyzer/. 2004. – last visited 4 January 2008

[Lug02] Luger, George F.: *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Forth Edition. Addison-Wesley, 2002

[Luk98a] Luke, Sean: Evolving soccerbots: A retrospective. In: *Proceedings of 12th Annual Conference of the Japanese Society of Artificial Intelligence*, 1998

[Luk98b] Luke, Sean: Genetic programming produced competitive soccer softbot teams for robocup97. In: *Proceedings of the Third Annual Genetic Programming Conference*, 1998, S. 204–222

[Maa07] Maass, Wolfgang. *Computational Intelligence*. Graz, University of Technology. March 2007

[MAS+07] Meyer, Jens ; Adolph, Robert ; Stephan, Daniel ; Daniel, Andreas ; Seekamp, Matthias ; Weinert, Volker ; Visser, Ubbo: Decision-Making and Tactical Behavior With Potential Fields. In: *RoboCup 2002, Lecture Notes in Artificial Intelligence*, 2007

[McC04] McCarthy, John: *What is Artificial Intelligence?* Online at http://www-formal.stanford.edu/jmc/whatisai/whatisai.html. 2004. – last visited 15 October 2007

[Min74] Minsky, Marvin: *A Framework for Representing Knowledge*. Memo 306. MIT-AI Laboratory, 1974

[Mit97] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill, 1997

[Moo95] Moore, Robert C.: *Logic and Representation*. Second Edition. CSLI, 1995

[Neh03] Nehmzow, Ulrich: *Mobile Robotics: A Practical Introduction*. Second Edition. Springer, 2003

[NF71] Nilsson, Nils J. ; Fikes, Richard E.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In: *Artifical Intelligence* 2 (1971), Nr. 3-4, S. 189–208

[Nil71] Nilsson, Nils: *Principles of Artificial Intelligence*. McGraw-Hill Book Co., 1971

[Nil86] Nilsson, Nils: *Problem-Solving Methods in Artificial Intelligence*. Morgan Kaufmann Publishers, 1986

[Nil94] Nilsson, Nils J.: Teleo-Reactive Programs for Agent Control. In: *Journal of Artificial Intelligence Research* 1 (1994), S. 139–158

[Qui03] Quinlan, John R.: *C4.5*. Kaufmann, 2003

[RA08] Rojas, Dario ; Atkinson, John: Generating Dynamic Formation Strategies based on Human Experience and Game Conditions. In: *Proceedings of RoboCup 2007: Robot Soccer World Cup XI*, Springer-Verlag, 2008. – To appear

[Rei07] Reip, Michael: *KickOffTUG: Multiagentensystem der RoboCup Simulation League*. Graz, Austria, Graz University of Technology, Diplomarbeit, October 2007

[Ret86] Retti, Johannnes: *Artificial Intelligence: Eine Einführung*. B. G. Teubner Stuttgart, 1986

[RG08] Riedmiller, Martin ; Gabel, Thomas: Brainstormers 2D Team Description 2007. In: *Proceedings of RoboCup 2007: Robot Soccer World Cup XI*, Springer-Verlag, 2008. – To appear

[RN03] Russel, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. Second Edition. Prentice Hall International, 2003

[Rob65] Robinson, John A.: A Machine-Oriented Logic Based on the Resolution Principle. In: *Journal of the ACM* 12 (1965), Nr. 1, S. 23–41

[Rob07a] The RoboCup Federation: *RoboCup Official Site*. Online at http://www.robocup.org. 2007. – last visited 5 October 2007

[Rob07b] The RoboCup Federation: Online at http://www. rescuesystem.org. 2007. – last visited 19 September 2007

[Rob07c] The RoboCup Federation: *RoboCup@Home Official Site*. Online at http://www.ai.rug.nl/robocupathome/. 2007. – last visited 6 September 2007

[Ros88] Rosenblatt, Frank: The perceptron: a probabilistic model for information storage and organization in the brain. In: *Neurocomputing: foundations of research* (1988), S. 89–114

[San89] Sanford, David H.: *If P, then Q: Conditionals and the Foundations of Reasoning*. Routledge, 1989

[Sav85] Savory, Stuart E.: *Künstliche Intelligenz und Expertensysteme: Ein Forschungsbericht der Nixdorf AG*. Oldenbourg, 1985

[SB98] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction*. The MIT Press, 1998

[Sch01] Schmaranz, Klaus: *Softwareentwicklung in C*. Springer, 2001

[Sch07] van Schooten, Boris: *YProlog Home*. Online at http://www.vf.utwente.nl/ schooten/yprolog. 2007. – last visited 10 January 2008

[Tes95] Tesauro, Gerald: Temporal difference learning and TD-Gammon. In: *Communications of the ACM* 38 (1995), S. 58–68

[Tra07] Edgewall Software: *The Trac Project*. Online at http://trac.edgewall.org/. 2007. – last visited 17 October 2007

[Wal05] Walde, Janette F.: *Design Künstlicher Neuronaler Netze: Ein Leitfaden zur effizienten Handhabung mehrschichtiger Perzeptrone*. Deutscher Universitäts-Verlag, 2005

[Woo02] Woolridge, Michael: *An Introduction to Multiagent Systems*. Wiley, 2002

[Zeh07] Zehentner, Christoph. *Teamstrategie KickOffTUG. Bakkalaureatsarbeit*. Graz, University of Technology. 2007