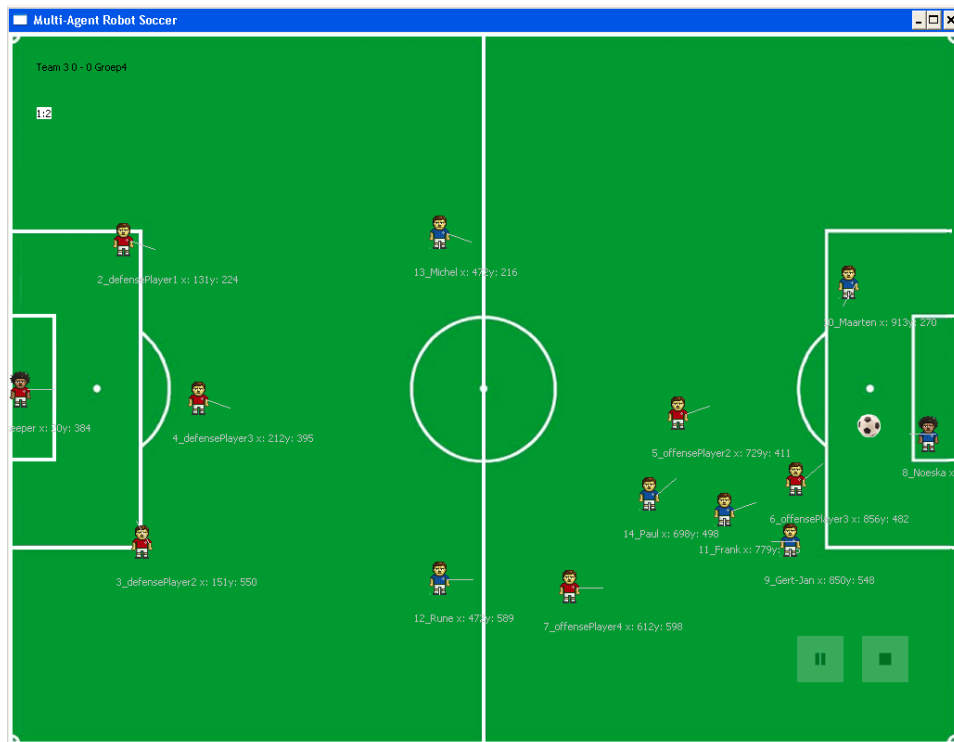


The Design and Implementation of a Multi-Agent Soccer Simulator as a Tool for AI Research and Education



Iwein J.J. Borm
September 1, 2006



Delft University of Technology
Faculty of Electrical Engineering, Mathematics, and Computer Science
Mediamatics: Man-Machine Interaction

The Design and Implementation of a Multi-Agent Soccer Simulator as a Tool for AI Research and Education

Master's Thesis in Media & Knowledge Engineering

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Iwein J.J. Borm

September 1, 2006

Man-Machine Interaction Group

Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Members of the Supervising Committee

drs. dr. L.J.M. Rothkrantz (chair)
dr. ir. C.A.P.G. van der Mast
ir. H.J.A.M. Geers

Copyright © 2006

Iwein J. J. Borm
1174398

Abstract

The Design and Implementation of a Multi-Agent Soccer Simulator as a Tool for AI Research and Education

Copyright © 2006 by Iwein J.J. Borm (1174398)

Man-Machine Interaction Group

Faculty of EEMCS

Delft University of Technology

Members of the Supervising Committee

drs. dr. L.J.M. Rothkrantz (chair)

dr. ir. C.A.P.G. van der Mast

ir. H.J.A.M. Geers

This thesis describes a multi-agent soccer simulator based on a highly simplified soccer model, and its application in an introductory AI course. The soccer model allows agents to start on a high level and removes many uncertainties that occur in physical soccer robots. Because of these simplifications, the system is suitable for researching high-level strategic behavior, such as cooperation and team work. The simulator features a three-layer model representing the physical environment, robot and behavior. This layered approach makes the system extensible. Reference teams of increasing complexity are available, and help to detect flaws in strategies, as well as pose a serious challenge for users to beat. Coursework was developed, that guides students through the process of creating a team capable of defeating the reference team, in order to teach students about basic AI concepts such as multi-agent systems, ad-hoc networks, rule-based reasoning and cooperative agents. The coursework has been used in a first-year undergraduate AI course at Delft University of Technology, and was found to be motivating and very educative.

Preface

This master's thesis describes the research and development I have done to graduate at the Man-Machine-Interaction group at Delft University of Technology. The goal of this thesis project is to develop a soccer simulator that can be used for AI research and as an educational tool, and to develop and use an educational assignment based on this simulator.

My first encounter with agents and multi-agent systems was early spring 2004. Whether it was the seemingly endless possibilities, the interesting assignments or the way agents can make complex problems look so easy, somehow agents and multi-agent systems have been at the center of my studies ever since. It was not a big surprise that my Master's project was also going to involve multi-agent systems.

In my search for a topic, one of my biggest requirements was that if I would have finished the work, I should be able to sit down with a random person and explain to him exactly what I did without losing his attention or raising countless questions. In case I should ever lose my enthusiasm, or if I am not physically present to explain the system, hopefully this thesis will be the next best thing.

Acknowledgements

The work you have in front of you is the fruit of my Master's project at Delft University of Technology. First and foremost, I would like to thank Maja Pantic for giving me the opportunity to be a part of the team, to develop and hone my programming and educational skills, and to motivate me to push for the limit.

Furthermore, I would like to kindly thank Reinier Zwitserloot and Robert Jan Grootjans for helping me with Fleeble and Eclipse. I would like to thank my supervisor Leon Rothkrantz for helping me focus on what is really important, keeping me on the track, and for giving me a lot of freedom in working on the thesis project. I would like to thank my father for his help with writing this thesis. I would like to apologize to all the people I have neglected during this project. Last but not least I would like to thank all students that evaluated the soccer assignment in 2005-2006.

Contents

Abstract	iii
Preface	v
Acknowledgements	vii
1 Introduction	1
1.1 Problem setting	1
1.1.1 Challenge	1
1.1.2 Game	3
1.1.3 Problem	3
1.2 Introductory AI Course	4
1.2.1 MKT-2	4
1.2.2 Research at TU Delft	5
1.3 Soccer Simulator	5
1.3.1 Model	5
1.3.2 Simulator	7
1.3.3 Education	8
1.4 Goals	9
1.5 Thesis Overview	10
I Preliminaries	13
2 Literature	15
2.1 Agents	15
2.1.1 Performance Measure	15
2.1.2 Simple Reflex Agent	16
2.1.3 Learning Agent	16
2.2 Multi-Agent Systems	17
2.3 Multi Agent Soccer	18
2.3.1 Aibo Soccer	18
2.3.2 Humanoid Robots	19
2.3.3 Soccer Simulator	20

2.3.4	Other Simulators	22
2.3.5	Multi Agent Soccer in Education	23
2.4	Overview	24
3	Tools	25
3.1	Overview	25
3.2	Programming Language	26
3.2.1	Educational Aspect	26
3.2.2	Conclusion	26
3.3	Agent Framework	27
3.3.1	Background Information	27
3.3.2	Role of the Agent Framework	27
3.3.3	Comparison	27
3.3.4	Java 1.5	28
3.3.5	History	29
3.3.6	GUI	30
3.4	Fleeble	30
3.4.1	Programming an Agent	31
3.4.1.1	Compiling the Agent	31
3.4.1.2	Loading the Agent	31
3.4.2	Agent States	31
3.4.3	Channels	32
3.4.3.1	Subscribing	33
3.4.3.2	Publishing	34
3.4.3.3	Dynamic Publishing and Subscribing	34
3.4.4	Thread Scheduling-induced Randomness	35
3.4.5	Namespaces	36
3.4.6	Special Channels	37
3.4.6.1	(INIT)	37
3.4.6.2	(CHILD_LOADED)	37
3.4.6.3	System	38
3.4.7	Message Queuing	38
3.4.8	Child Agents	40
3.4.8.1	MessageSource	40
3.4.9	Properties	42
3.4.10	Namespace Mirroring	42
3.4.11	Overview	43
3.4.12	Summary	43
3.5	Eclipse	45
3.5.1	CVS	46
3.5.2	Ant	46
3.5.3	SWT	46

II	Game	47
4	Model	49
4.1	Problem Description	49
4.2	Requirements	50
4.2.1	Actors	50
4.2.1.1	Students	51
4.2.1.2	Hobbyists	52
4.2.1.3	Researchers	52
4.2.2	Package	53
4.2.3	Summary	53
4.3	Laws of the Game	54
4.3.1	Goal	54
4.3.2	Players	54
4.3.3	The Field of Play	55
4.3.4	Ball in and out of Play	56
4.3.5	Foul Play and Misconduct	56
4.3.6	Offside	56
4.3.7	Duration of the Match	57
4.3.8	Environment	57
4.3.9	Overview	57
4.4	System Model	59
4.4.1	Player Layer	59
4.4.2	Robot Layer	60
4.4.3	Framework Layer	61
4.5	Player Capabilities Model	61
4.5.1	Movement	62
4.5.1.1	Basic Player Movement	62
4.5.1.2	Ball Movement	62
4.5.1.3	Collisions	62
4.5.1.4	Movement Speed	63
4.5.1.5	Overview	64
4.5.2	Visual Model	64
4.5.3	Aural Model	65
4.5.4	Changing Behavior	65
4.6	Justification	67
4.6.1	Speed Penalties	67
4.6.1.1	Shouting	68
4.6.1.2	Collisions	68
4.6.2	Movement	69
4.6.3	Communication Restrictions	69
4.6.4	Central authority	71
4.7	Customizing the Model	71
4.7.1	Changing Framework Parameters	72

4.7.2	Changing Visual Appearance	72
4.7.3	Advanced Customization	72
5	Design	73
5.1	Approach	73
5.2	Use Case Design	74
5.3	System Design	74
5.3.1	Global Design	75
5.3.2	Framework Channels	76
5.3.3	Robot Channels	76
5.3.4	Player Channels	76
5.4	Classes	83
5.4.1	Overview	83
5.4.2	Tournament Status	83
5.4.3	Initializing the Framework	83
5.4.4	Running the Game	85
5.4.5	Vision	87
5.4.6	Shouting and Listening	87
5.4.7	Team Creation	89
5.5	Movement	89
5.5.1	Coordinates	91
5.5.2	Movement Model	92
5.5.3	Collision Model	97
5.5.4	Ball Model	98
5.5.4.1	Ball Movement	98
5.5.4.2	Ball Collisions	98
5.6	User Interface Design	101
5.6.1	Settings	101
5.6.1.1	Main Screen	101
5.6.1.2	Team Creation Interface	102
5.6.1.3	Tournament Creation Interface	105
5.6.1.4	Error Handling	105
5.6.2	Game	107
6	Implementation	113
6.1	Approach	113
6.1.1	Tools	114
6.2	First Phase: Initializing the Framework	114
6.3	Second Phase: Sensors, Effectors and Movement	114
6.3.1	Position Tracking	115
6.3.2	Aural Model	115
6.3.3	Visual Model	115
6.3.4	Movement Model	115
6.4	Third Phase: Graphical User Interface	115

6.5	Final Phase: Added Features	116
6.5.1	Collision Detection	116
6.5.2	Ball Movement	116
6.5.3	Change Behavior	116
6.5.4	Other Features	116
6.6	Running the Simulator	117
6.7	Testing	122
6.8	Future Work	122
 III Research		125
7	Agent Strategies	127
7.1	Introduction	127
7.2	Player Skills	127
7.2.1	Low-level Player Skills	128
7.2.1.1	Turn the Robot towards a Point	128
7.2.1.2	Scout the Area	128
7.2.2	Intermediate-level Player Skills	129
7.2.2.1	Turn the Robot towards an Object	129
7.2.2.2	Track the Ball	129
7.2.2.3	Move to a Position	129
7.2.3	High-level Player Skills	130
7.2.3.1	Intercept a Ball	130
7.2.3.2	Passing the Ball	131
7.2.3.3	Give a Leading Pass	132
7.2.3.4	Give a Through Pass	132
7.2.3.5	Clearing the Ball	133
7.2.3.6	Move towards Opponent Goal	133
7.2.3.7	Move Free from Teammate	134
7.2.3.8	Avoid Opponent	134
7.3	Inference	135
7.3.1	Determine if an Object's Position is known	135
7.3.2	Determine if an Object is Visible	136
7.3.3	Determine if the Ball is Visible	136
7.3.4	Determine if the Ball is Kickable	136
7.3.5	Determine the Closest Opponent	137
7.3.6	Determine whether an Opponent is up Ahead	137
7.3.7	Determine whether there is a Teammate Standing Free	137
7.3.8	Determine whether a Teammate has the Ball	138
7.3.9	Determine whether the Ball is Free	138
7.3.10	Determine if the Ball Position is known	139
7.3.11	Determine if there is Chaos around the Ball	139
7.3.12	Determine if Player had Collision	139

7.3.13	Determine Player Speed	140
7.4	Strategy	140
7.4.1	Social Laws	141
7.4.1.1	Social Laws versus Communication	141
7.4.2	Communication	142
7.4.3	Formation	143
7.4.3.1	Positioning	143
7.4.3.2	Role Allocation	143
7.4.3.3	Changing the Strategy	144
7.4.4	Adaptive Behavior	145
7.4.4.1	Adaptive Team Behavior	145
7.4.4.2	Individual Adaptive Behavior	146
7.4.5	Rule-based Action Selection	149
7.4.6	A Multi-Agent Approach to Strategy	149
7.5	Summary	150
8	Development of a Team	153
8.1	Introduction	153
8.2	Design	154
8.2.1	Roles in the Team	154
8.2.2	Formation	154
8.2.3	Player Skills	154
8.2.4	Inference	155
8.2.5	Rule-based Action Selection	156
8.3	Implementation	157
8.3.1	Template	157
8.3.2	UpdateKnownInfo	159
8.3.3	AnalyzeSituation	160
8.3.4	DoStrategy	160
8.4	Advanced Team	160
8.5	Summary	161
IV	Education	163
9	MKT-2 Project	165
9.1	Educational Goals	165
9.1.1	Approach	166
9.2	MKT-2 Assignments	166
9.2.1	Assignment A: Roshambo	167
9.2.2	Assignment B: Rule-based Reasoning	169
9.2.3	Assignment C: Hollywood	170
9.2.4	Assignment D: Multi-Agent Systems	171
9.2.4.1	2003-2004: Peer-to-peer Networks	171

9.2.4.2	2004-2005: SMS Assignment	171
9.2.4.3	Discussion	173
9.3	Setting	174
10	Soccer Assignment	177
10.1	Educational Goals	177
10.2	Assignment	178
10.2.1	Scenarios	179
10.2.1.1	Player Level	179
10.2.1.2	Team Level	180
10.2.2	Design	180
10.2.3	Implementation	181
10.2.3.1	Implementation Questions	181
10.2.4	Extra Questions	182
10.2.5	Competition	182
10.2.5.1	Competition Results	182
10.3	Implementations	183
10.3.1	Team Setup	183
10.3.2	Reasoning	185
10.3.2.1	Knowledge Base	185
10.3.2.2	Situation Analysis	186
10.3.2.3	Rule Base	186
10.3.2.4	Actions	187
10.4	Evaluation	188
10.4.1	Classroom Observations	188
10.4.1.1	Educational Goals	188
10.4.1.2	Improvement	191
10.4.2	Survey	193
10.4.2.1	Survey Results	194
10.4.2.2	Educational Goals	195
10.4.2.3	Improvements	196
10.5	Conclusion	196
10.5.1	Educational Goals	196
10.5.2	Improvement	197
10.6	Discussion	198
V	Results	199
11	Conclusions	201
11.1	Literature	201
11.2	Game	201
11.2.1	Model	201
11.2.2	Design	202

11.2.3	Implementation	202
11.3	Research	202
11.3.1	Agent Strategies	202
11.3.2	Developing a Team	203
11.4	Education	203
11.4.1	MKT-2 Project	203
11.4.2	Soccer Assignment	203
12	Discussion & Future Work	205
12.1	Discussion	205
12.2	Future Work	205
12.2.1	Model	205
12.2.2	Soccer Simulator	206
12.2.3	Agent Strategies	206
12.2.4	Soccer Assignment	207
A	Publication for ELCONF'06	209
B	Tournament Parameters	217
C	Detailed description of MKT-2 Project	221
C.1	History	221
C.2	Background	222
C.2.1	Students	222
C.2.1.1	Java Programming Skills	222
C.2.1.2	Group work	222
C.2.2	Evaluation	223
C.2.2.1	Studentrate	223
C.2.2.2	Group Formation	224
C.2.2.3	Gantt Charts	226
C.2.2.4	Data Flow Diagram	227
C.2.2.5	Oral Exam	227
D	Soccer Assignment: Survey	229
	Bibliography	233

List of Figures

2.1	A Simple Reflex Agent	16
2.2	A Learning Agent	17
2.3	Aibo Soccer	19
2.4	Humanoid Soccer Robots at the 2005 Competition	20
2.5	2D Soccer Simulator	22
3.1	GUI of Fleeble Agent Framework	30
3.2	Loading an Agent in Fleeble	31
3.3	Fleeble Namespaces	37
3.4	Child Agents in Fleeble	41
3.5	Virtualhost for Locking Agents in a Certain Namespace	42
3.6	Class Diagram of Fleeble Agent Framework	44
3.7	Eclipse Software Development Kit	45
4.1	System Model: the Layered Approach	59
5.1	Use Case Diagram for User of Simulator	74
5.2	System Overview	75
5.3	Overview of the Framework Channels	77
5.4	Overview of the Robot Channels	79
5.5	Overview of the Player Channels	81
5.6	Field Coordinate System	93
5.7	Team Coordinate System	94
5.8	Relative Coordinate System	95
5.9	The Main Screen	102
5.10	The Team Creation Screen	103
5.11	The Tournament Creation Screen	106
5.12	Error Message from Incorrect Input	107
5.13	Example of the Tooltip Help	107
5.14	The Soccer Simulator Interface	108
5.15	The Visual and Aural Sensor Areas of a robot	111
6.1	Entering the Field	118
6.2	Exiting the Field	118

6.3	Pause / Stop Buttons	119
6.4	Game Time, Score and Team Name	119
6.5	Robot Visualization	120
6.6	The See Channel	121
7.1	Example use of Communication to Outplay Opponents	142
8.1	The SimpleTeam Formation	155
9.1	Assignment A: Rock, Paper, Scissors	168
9.2	Assignment B: Rule-based Reasoning	169
9.3	Assignment C: Hollywood	170
9.4	Assignment D 2003-2004: Peer-to-peer networks	172
9.5	Assignment D 2004-2005: SMS Assignment	173
10.1	The Setups that were used by the Different Teams	184
10.2	A Typical PlayerAgent	185
10.3	An Example Scenario	190
C.1	An Example Gantt Chart	226

List of Tables

1.1	Comparison of Chess and Soccer	2
1.2	The Three Layers of the Soccer Simulator	6
1.3	Robot Sensors and Effectors	7
1.4	User Groups and Corresponding Goals	7
3.1	Overview of available Java-based agent frameworks	29
3.2	Overview of Possible Agent States and their Icons.	32
3.3	Publishing and Subscribing to Channels	32
3.4	Dynamic Publishing and Subscribing	35
3.5	Fleeble's System Channels	38
4.1	Domain Characteristics of the Simulator	50
4.2	Human vs Simulator Players	55
4.3	Human vs Simulator Field of Play	56
4.4	Human vs Simulator Laws	58
4.5	Description of the Player Layer	60
4.6	Description of the Robot Layer	60
4.7	Description of the Framework Layer	61
4.8	Overview of the Movement Model	64
4.9	Overview of the Visual Model	65
4.10	Overview of the Aural Model	66
4.11	Overview of the Change Behavior Mechanism	67
5.1	Description of the framework channels	78
5.2	Description of the robot channels	80
5.3	Description of the player channels	82
5.4	The Different States of a Tournament	84
5.5	An Overview of VisibleObject, RobotObject and BallObject	87
5.6	The MessageTuple Object	88
5.7	The PlayerInfo Object	90
5.8	The MatchSetup Object	90
5.9	Overview of MoveTuple	92
5.10	Three Different Coordinate Systems	93
5.11	Using the Orientation Class for Retrieving different Coordinates	94

5.12	Parameters dealing with Ball Collision	100
5.14	Interface Elements of Team Creation / Modification	103
5.13	Interface Elements of the Main Screen	109
5.15	Interface Elements of Tournament Creation	110
7.1	Adaptive Team Behavior	146
10.1	Results from the 2006 MKT-2 soccer competition.	183
10.2	Survey Results	193
B.1	An overview of all Tournament Parameters	217
C.1	Studentrate	224
C.2	Studentrate (continued)	225

Listings

3.1	Programming an Agent	31
3.2	Publishing and Subscribing	33
3.3	Handling the incoming messages	33
3.4	Advanced handling of incoming messages	35
3.5	Send 'look!' every 100ms	39
5.1	Running the Game	86
5.2	The redraw() method	86
5.3	The Visual Model	88
5.4	Team XML file	91
5.5	The Movement Model	93
5.6	The Collision Model	97
5.7	The Ball Model	98
5.8	The Ball Collision Model	99
7.1	Turn the robot towards a point	128
7.2	Scout the Area	129
7.3	Move to a Position	130
7.4	Intercepting the Ball	131
7.5	Pass the Ball	132
7.6	Leading Pass	132
7.7	Through Pass	132
7.8	Move towards Opponent Goal	133
7.9	Move Free from Teammate	134
7.10	Avoid the Opponent	134
7.11	Determine if an Object's Position is Known	135
7.12	Determine if an Object is Visible	136
7.13	Determine if the Ball is Visible	136
7.14	Determine if the Ball is Kickable	136
7.15	Determine the Closest Opponent	137
7.16	Determine if an Object up Ahead	137
7.17	Determine if Teammate is Free	138
7.18	Determine whether a Teammate has the Ball	138
7.19	Determine whether the Ball is Free	138
7.20	Determine whether there is Chaos around the Ball	139
7.21	Determine whether there was a Collision	139

7.22	The proposed learning mechanism	148
7.23	A Simple Rule-based Action Selection Mechanism	149
8.1	SimpleTeam's Rule-base Action Selection Mechanism	156
8.2	Code template for SimplePlayer	157
8.3	Update the known info	159

Chapter 1

Introduction

Some people believe football is a matter of life and death. I'm very disappointed with that attitude. I can assure you it is much, much more important than that. Bill Shankly

This study explores the possibilities of a multi-agent soccer simulator based on a highly simplified soccer model as a tool for AI research and higher education. The simulator provides users with insight in various artificial intelligence techniques such as multi-agent systems, ad-hoc networks, rule-based reasoning, cooperative agents and team work, and can function as a testbed for research in these domains.

In the first part of this chapter, the current situation in the domain of robot soccer is explored. The second part presents the context in which this research was conducted. The next part provides an overview of the model, soccer simulator, and an application of these in an educational environment. The next part presents the goals for this project. Finally an overview of the structure of this report is given.

1.1 Problem setting

1.1.1 Challenge

Multi Agent Systems (MAS) deal with multiple agents that are collectively capable of reaching goals that are difficult or impossible to achieve by individual agents. One increasingly popular example of MAS is robot or embodied agents. The ultimate challenge of this domain is stated as follows [1]:

”By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rules of the FIFA[2], against the winner of the most recent World Cup.”

The 2050 goal is the successor of the chess challenge, that was accomplished in 1997 [3]. It can make AI interesting and popular once again. Soccer appeals to many individuals, and the challenge will appeal to a large number of researchers. The domain characteristics of soccer vary largely from the domain characteristics of chess. Table 1.1 [4] illustrates this.

Table 1.1: Comparison of Chess and Soccer

	Chess	Soccer
Environment	Static	Dynamic
State Change	Turn taking	Real time
Info. accessibility	Complete	Incomplete
Sensor Readings	Symbolic	Non-symbolic
Control	Central	Distributed

To achieve the 2050 goal, many problems have to be solved. Guidelines have been suggested [5] as to what must be done to achieve this goal. In accordance with this, the Robot World Cup (RoboCup) Federation [4] has introduced several robotic soccer leagues to host competition on a number of different abstraction levels of the ultimate goal. The RoboCup Federation strives to raise the bar in their leagues annually. The rules of the game slowly converge to that of actual human soccer. Examples of such leagues are the simulation league, humanoid robot league, Aibo robot league or the small-sized robot league.

State of the art robots are presently no match for the capabilities of human soccer players. A lot of progress has been made, and is being made towards achieving this ambitious goal. To have a successful team however, another key ingredient is required: Artificial Intelligence (AI). The simulation league of RoboCup is presently a testbed for exploring this, and the area receives a lot of attention. The goal of the simulator in this simulation league is to provide a realistic simulation of soccer, in accordance with most FIFA laws of the soccer game.

The problems that have to be solved for accomplishing the challenge would be incredibly useful in many other domains. Fully autonomous humanoid robots that can cooperate to accomplish complex tasks would meet some of the expectations set out in recent popular A.I. movies.

Artificial Intelligence: the art of making computers that behave like the ones in movies. Bill Bulko

1.1.2 Game

Human soccer is a game played by teams of 11 players. To win a game, a team has to score more goals than their opponents. A number of laws [2] exist, to ensure fair play, and punish foul play such as tackling or pushing the opponent. Free kicks or penalties may be awarded, and players can be removed from the field after being booked for serious foul play. When a ball leaves the field, the team that did not touch the ball last receives a throw-in. To prevent an unfair advantage to offensive players waiting at the opponent's side of the field for the ball, an offside rule is implemented. A player is offside when he is nearer to his opponents goal line than both the ball and the second last opponent at the moment when a co-player passes him the ball [2] [6]. Corner kicks and goal kicks are awarded when the ball crosses the goal line as a result of action by the defensive and offensive team, respectively. Players may not use their arms or hands to control the ball. Every team has one goalkeeper that is allowed to use his arms, in order to keep the ball out of the goal.

1.1.3 Problem

The high level aspects, such as dealing with cooperation and team work are not dealt with extensively in present literature. These high level aspects are crucial for a team's success. Imagine having a robot able to dribble, pass and catch the ball perfectly. This robot would be useless in the field, as it will not know where to pass the ball to. It does not know where its team mates are, or what strategy they are using. Having such a hypothetical robot does not solve the entire problem, as it leaves the issues of strategy and cooperation unsolved.

The main reason why most research is being done on the low level details of playing soccer, is that having a reliable, fast, low level implementation is a necessary requirement for any high level strategies. It is however not a sufficient requirement. There is no use in reasoning about complex strategies to trick the opponent when simple commands are not executed in accordance with the expectation. For physical robots, issues like unrealistically low shot accuracy, robots that can not distinguish between audience and team mates, having too little processing power for thorough analysis of visual data, are all real problems that require solutions prior to any sophisticated team play algorithms.

In both the simulation league, and in all physical robot leagues, a low level implementation of many primitive functions is required before having a fully functional team. Furthermore, there is a large dependance on the quality and stability of the low-level implementation on the performance

of high-level strategies. As a result, the problem domain that was already inherently very uncertain is extended with even more uncertainties.

The threshold for starting in the domain is rather high. Researchers have to commit themselves for a longer period of time, programming many low level details before being able to work on the higher level implementation - which is then plagued by the uncertainties introduced by this low level implementation.

Achieving the 2050 goal requires (1) technological improvement to produce sophisticated humanoid robots, but also (2) new AI methods to deal with the domain characteristics of soccer. Current practises attempt to tackle the latter through simulators that realistically simulate a human soccer environment.

Using this approach, researching high level strategies requires a transparent, stable low-level implementation for basic input / output and basic player skills. Developing such an implementation requires a long-term commitment from researchers. The rules of the game are predefined, hence decisions about introducing uncertainties can not be modified to suit (hypothetical) technological progress¹. High-level efforts will have to deal with uncertainty inherent to a soccer environment, uncertainty introduced by the predefined world model, uncertainty from the imperfect implementation of the low-level details, and with uncertainty introduced by the freedom of choice of the player.

1.2 Introductory AI Course

This research was initiated for the purpose of improving a first-year undergraduate introductory AI course at Delft University of Technology. This section introduces the course and related research.

1.2.1 MKT-2

MKT-2 is the second project in the first year of computer science undergraduate students at Delft University of Technology. The educational goals of the project are [7]:

1. To introduce the basic concepts of knowledge engineering and the relevant AI techniques, including search algorithms, knowledge representation techniques, rule-based reasoning algorithms, and agent technology.

¹For example, if a breakthrough in real-time object recognition would occur, the world model of the simulator would not match that of the physical state of the art.

2. To explain and instruct on issues related to AI programming in general and intelligent (multi-) agent applications in particular.

1.2.2 Research at TU Delft

At Delft University of Technology, several people have contributed to the research on creating, improving and evaluating the MKT-2 project. A simple agent framework, Fleeble, was developed specifically for the project by Reinier Zwitterloot and Robbert Jan Grootjans [8] [9] [10]. Several challenging assignments were developed [11][12] to achieve the aforementioned educational goals. Research has been conducted to develop an assignment and corresponding software to teach students about (mobile) ad-hoc networks and multi-agent systems [7]. Several conceptual and software problems have prevented this assignment from reaching its goals. This problem is addressed in this graduation project, as an alternative assignment based on the soccer simulator with similar educational goals was developed and used.

1.3 Soccer Simulator

The system described in this thesis was *not* designed from the perspective of being a realistic soccer simulation. Instead it was designed as an educational model, designed to suit the needs of a broad group of users, including students and researchers.

1.3.1 Model

The system uses an extensible three layer model to model the world, as described in Table 1.2. The world model described in this thesis is highly simplified, compared to that of existing soccer simulators and real human soccer.

Rules governing fair play, dealing with amongst others free kicks, yellow and red cards, penalties, throw-ins, corners, goal kicks and offside, are all necessary elements of human soccer. Including these rules in a simulator however, necessitates their inclusion in the design and implementation of any team. Another dimension of soccer is coaching and allocating players with certain strengths and weaknesses to certain positions. Although attention to these properties and events must be paid in order to compete with human players, they distract attention from developers towards handling these special cases. For this reason, all aforementioned rules are not included in the world model of the system. Furthermore, all robots are physically identical.

Table 1.2: The Three Layers of the Soccer Simulator

Entity	Metaphor	Description
Player	Brains	Defines the behavior of the robot. Has access to all sensory inputs and can use high level commands to address all the robot's effectors.
Robot	Human body	Physically present on the field, will attempt to do as the brain tells it, but can be limited by physical laws. In the simulator this layer is relatively transparent, but the robot layer can be transformed to introduce low-level uncertainties to simulate realistic input.
Framework	Physical laws	The rules of the game are controlled by the framework. This includes parameters such as ball speed, player speed, robot vision angle, but also for example the way collisions are dealt with. The framework layer is also responsible for the graphical representation of the world.

Robots have a number of basic sensors and effectors, as can be seen in Table 1.3. To support adaptive team behavior, an option is included that effectively *substitutes* the behavior of a certain player for that of another player. These are the only ways in which robots can interact with the system.

The world model used in the simulator can be modified and extended with relative ease at each of the separate layers. Extending the model with uncertainty about what can be seen or what was heard and by whom is as simple as adding a filter to the robot layer. The complete, perfect information that is available from the framework layer can be filtered² to simulate the imperfect sensory capabilities of robots.

²Such a filter can be deterministic, for example a filter that will pass on the content of a message, but not the sender. It can also be nondeterministic, to simulate noise or random behavior in the environment.

Table 1.3: Robot Sensors and Effectors

Name	Description
See	Robots have a narrow field of vision that provides perfect information updated at predefined time intervals
Move	Robot movement is composed of a possible turn, a possible special move such as a kick, and a direction. Movement is processed at predefined time intervals.
Shout	Robots can communicate with each other through message passing. Any data can be sent at any time, and all robots within a predefined distance will receive the message without any errors.
Listen	Robots will receive all messages sent by other robots within a predefined distance.

1.3.2 Simulator

The simulator is designed for three groups of users. As the requirements differ for each of these groups, the system is designed to be flexible and extensible. Three different goals can be formulated for each of these groups. The goals are not mutually exclusive, and considerable overlap will exist. The three different groups and their corresponding primary goals are illustrated in Table 1.4.

Table 1.4: User Groups and Corresponding Goals

User	Goal
Student	Achieve the educational goals mentioned in Section 1.3.3.
Hobbyist	Lower the threshold for starting in the domain of AI, and as such open the domain to a broader audience.
Researcher	Serve as a testbed for research on cooperation and team work.

The simulator is also designed to be easy to learn and use, and features an intuitive customizable interface. Reference teams are included to give users a kick start in developing their team and in providing insight in the problems that individual robots encounter.

1.3.3 Education

The simulator based on the simplified soccer model is very suitable for educational purposes. Depending on the level of the students, the time available for the project and the specific educational goals, custom assignments based on the simulator are readily made. The available reference teams pose a serious challenge to students, and make the performance of their implementations easily verifiable.

The development of the simulator was initiated by the lack of a good assignment for teaching students about multi-agent systems and ad-hoc networks in the MKT-2 project. Earlier assignments were not successful, and a lot of time was spent reflecting on what elements were crucial to the success of an assignment, and what elements were devastating. These were subsequently taken into account for the development of a new assignment.

This new assignment has the following educational goals³:

- Learn students to work in groups.
- Learn students to develop an implementation from a *vague, high-level* problem description, including:
 - analyzing the problem
 - designing a solution
 - identifying and distributing tasks between group members
 - implementing the individual tasks; learning about the difficulties of integrating several parts of a system, and learning to cope with unexpected results from their implementation.
- Learn students about:
 - distributed artificial intelligence,
 - multi-agent systems,
 - mobile ad-hoc networks,
 - decentralized control,

³The group work / problem solving goals are similar to the goals of most projects. The *vague, high-level* problem description, AI, and programming skill level goals are specific to this project

- cooperative agents,
- rule-based reasoning.
- Increase the programming skill level of the students.

The assignment that was given is the following: *Develop a soccer team that is capable of consistently defeating the reference team.* The assignment was used in 2005-2006 in the MKT-2 Project at Delft University of Technology. As a competitive element to increase the motivation of the students, a tournament between all student teams was held at the end of the project.

1.4 Goals

The main goal for this project is to design and implement a multi-agent soccer system based on the highly simplified soccer model that allows users to start immediately on a high level, and use this in an introductory AI project. The previous sections described the functionality and the goals of the soccer simulator. Rather than developing a 'realistic' soccer simulator, the simulator that is described in this thesis uses a highly simplified soccer model that allows users to start immediately on a high level, and is very suitable for on researching cooperation and team work. A summary of the goals is listed below:

1. Game

- Literature survey on the state of the art of robot and multi-agent soccer
- Design the simulator
 - Research the requirements of the model
 - Design the soccer model
 - Research available agent frameworks and tools
 - Design a model of the simulator, that considers the properties of the specific tools that are used, for implementation of the system
 - Design the user interface
- Implement the simulator.

2. Research

- Investigate the capabilities of a team in the simulator
 - Determine what the individual robots can reason about
 - Investigate player skills, reasoning methods, and adaptive team behavior, and how these can be implemented in the simulator

- Implement a reference team based on rule-based reasoning.

3. Education

- Develop an educational assignment based on the soccer simulator, that achieves the goals described in Section 1.3.3.
- Use the assignment in an introductory AI project
- Evaluate the assignment
 - Verify that the assignment accomplishes its educational goals
 - Evaluate the different approaches that were used by the teams of students to develop their soccer teams
- Research potential alternative educational applications of the soccer simulator.

Talent wins games, but teamwork and intelligence win championships. Michael Jordan (Famous basketball player)

1.5 Thesis Overview

Chapter 2 provides some background information about (multi-) agent systems, robot soccer, and its application in education.

In Chapter 3 the development tools that were used are described and justified. A detailed explanation of the agent framework that was used is included, as an understanding of its properties and limitations is important for the understanding of certain properties of this particular implementation.

Chapter 4 describes the soccer model that is incorporated in the simulator. It justifies the decisions and trade-offs that had to be made, and describes how the model can be extended or modified.

In Chapter 5 the design of the system is discussed. The global system design is introduced, followed by a detailed explanation of all relevant classes that were used. A thorough description of the movement model is given, and the user interface design is discussed.

Chapter 6 describes the implementation of the soccer simulator. It begins by describing the approach that was used for implementing the system. Then it provides an overview of the most important characteristics of the three main elements of the simulator. It concludes with a presentation of the final result.

Chapter 7 introduces several aspects to agent strategies. It describes numerous low-, medium-, and high-level player skills, and a number of inference steps that map the basic percepts to high-level information about the environment. Agent and team strategy are discussed, and adaptive behavior is introduced. Furthermore, it explains the rule-based action selection mechanism.

Chapter 8 discusses the design and implementation of the SimpleTeam for the simulator. The SimpleTeam was used in the introductory AI course. It describes the player skills, action selection mechanism and the inference steps.

In Chapter 9, some background information about the MKT-2 project is provided. The predecessors of the soccer assignment are analyzed and discussed. The chapter concludes with a description of the setting of the MKT-2 project in 2005-2006.

Chapter 10 introduces the soccer assignment that was used in the MKT-2 project in 2005-2006. The educational goals and the outcome are discussed. Classroom and survey evaluation results are presented and analyzed. The chapter concludes with a number of improvements for future projects.

Chapter 11 describes the conclusions of the project. The results are evaluated and compared to the goals that were formulated in Section 1.4.

Chapter 12 concludes this thesis by a discussion and suggestions for future work.

Appendix A contains a publication about the educational aspect of this graduation project, that was accepted for presentation at the E-learning Conference 2006.

Appendix B contains an overview of the tournament parameters.

Appendix C provides an in-depth description of the educational approach that was taken in the MKT-2 project.

Appendix D contains the survey that was handed out to all students participating in the MKT-2 project in 2005-2006.

This report, by its very length, defends itself against the risk of being read⁴. Winston Churchill

⁴completely

Part I

Preliminaries

Chapter 2

Literature

Literature is the question minus the answer. Roland Barthes

This chapter provides an overview of the state of the art in the domain of soccer. The first section introduces agents and multi-agent systems, followed by an introduction to Multi Agent Soccer. Within multi agent soccer, Aibo robots, humanoid robots, soccer simulators and multi agent soccer in education are described. The chapter concludes with a brief overview of the state of the art. This chapter serves only to introduce the topics and to provide a general overview. A discussion of specific theories or techniques is left for later chapters.

2.1 Agents

An Intelligent Agent (IA) is a computer program that uses AI-techniques to autonomously accomplish specific goals within a predefined problem domain¹ [13]. An agent is a computer program that can be seen as an **entity** that **observes** its environment through **sensors**, and **reacts** on this using its **effectors**.

2.1.1 Performance Measure

A rational agent will perceive its environment, reason about this, and choose the action that it expects to yield the best result. Before such an agent can be built, a performance measure that evaluates the success of an agent is required. A common trade-off in determining the performance of an agent is effectiveness versus efficiency. When the goal is to win a soccer game, and the agent wins in the last minute of the extended time by an own goal of the opponent, should this yield the same performance rating as the agent that defeats his opponent by 12-0?

¹There are more definitions of an Intelligent Agent

Determining good performance measures is a nontrivial task. For example when determining the performance measure of a vacuum cleaning agent, using the amount of dust that is collected would have the following undesirable effect: An agent that collects one bag of dust, puts this on the floor again and vacuums it up will get a very high performance rating. The agent is not effective at performing the task (cleaning the house), however.

2.1.2 Simple Reflex Agent

Figure 2.1 shows a simple reflex agent. First this agent perceives its environment. Then, it reasons about the appropriate action using (if-then) rules. The action that is chosen yields the highest expected performance rating. Finally, the action is performed using the agent's effectors.

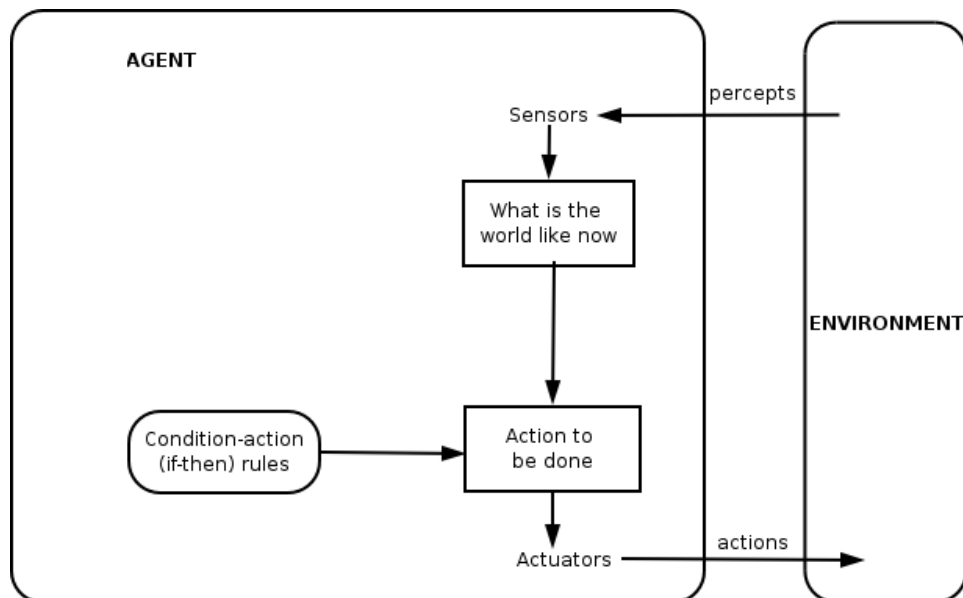


Figure 2.1: A Simple Reflex Agent

2.1.3 Learning Agent

A more advanced agent is the learning agent. This agent has the same sensors and effectors, but reasons in a different way. The learning agent is illustrated in Figure 2.2 [13]. The critic provides feedback to the learning element on how the agent is doing, based on the fixed performance standard. In a soccer game, this performance standard could be that the agent's team should have a higher score than the opponent. The learning element determines how the agent should perform in the future, based on this feedback, and determines how the performance element - that decides how to

react on certain inputs - should be adapted to perform better. The problem generator will suggest new actions to the performance element. Without the latter, the performance element would only act according to do what is best, given what it knows. Through exploring the unknown, new 'best moves' could be discovered.

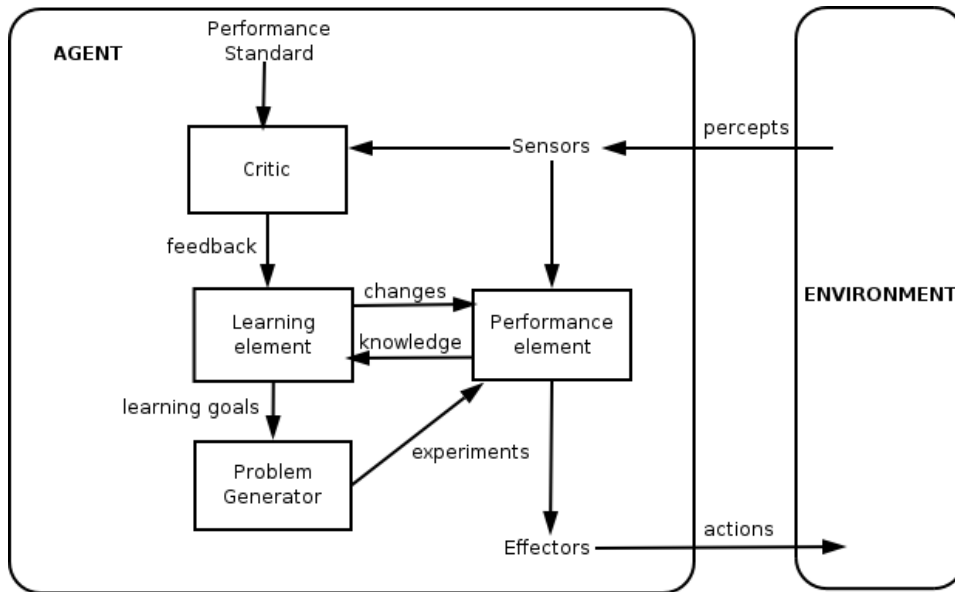


Figure 2.2: A Learning Agent

2.2 Multi-Agent Systems

Multi Agent Systems are systems composed of multiple agents that are collectively capable of reaching goals that are difficult to achieve by single agents. Agents are in principle autonomous, but for many serious activities a certain amount of human supervision, coordination or control is required. Human agents can also be a part of a multi-agent system. Conventional approaches to difficult problems would create highly customized, complex implementations. Multi-agent systems make it easy to structure the data flow between important parts of a system. This makes it easier to change and maintain, and prevent many complexity related bugs from occurring.

The characteristics of MAS's are that (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous [14].

2.3 Multi Agent Soccer

As early as in 1994, soccer was introduced as one of the great challenges for Artificial Intelligence [15]. A lot of research and experimentation has been going on in the increasingly popular domain since then. In the year 2000, the 2050 goal [1] was formulated as the new challenge for AI for the coming 50 years, in response to the completion of the chess challenge in 1997.

A number of important issues that would have to be addressed to reach this goal were identified by [15], and were stated as follows: (1) *Real-time decision making*; (2) *Planning*; (3) *Plan recognition*; (4) *Modeling*; (5) *Learning*; (6) *Multi-agent theory*; (7) *Robot architectures*.

These issues have been researched and a vast body of literature on the specific topics have been produced. The current practises are still nowhere near the 2050 goal. Competing with humans and in accordance with the FIFA rules demands highly sophisticated humanoid robots that have physical abilities comparable to that of humans. Developing such robots can not be done overnight. A set of intermediate targets with regards to the robots, the rules of the game and the environment in which the game takes place have been set out in [5]. Through gradually adding complexity and making the competitions ever more humanoid, the goal is to be met. This section provides an overview of the state of the art in several application domains of robot soccer.

The RoboCup [4] consists of 5 leagues; (1) small-sized robots; (2) middle-sized robots [16]; (3) four-legged robots (Aibo's); (4) humanoid robots; and (5) simulation. The four-legged league, the humanoid league and the simulation league will be discussed. After this, an overview of multi-agent soccer in (computer science) education is given.

2.3.1 Aibo Soccer

The Aibo [17] is a small autonomous robotic pet dog, that was introduced in 1999 and is being used for the four-legged league of the RoboCup [4]. (See Figure 2.3) In this league, 24 teams of four Aibo robots compete on a 6m by 4m playing field. The robots are allowed to communicate with each other acoustically or through their wireless network cards. As the Aibo robots are complete *off the shelf*, and come with a wireless network card, camera, microphone, speaker, LED display, and a wide array of sensors and moveable parts, the RoboCup regulations forbid any hardware modifications to the standard Aibo. This forces participating teams to focus solely on controlling the robot, and achieving cooperation and team play, rather than focussing on improving or modifying the hardware.

The Aibo's behavior can be reprogrammed using the Aibo Software Development Environment [18]. This SDE consists of (1) the Open-R SDK - a C++ based environment that can be used for programming the Aibo. These tools can control the movements of the robot's joints, gather information from sensors and the camera, or communicate using the wireless LAN; (2) R-Code SDK - allows the execution of R-Code, a scripting language, on the Aibo; and (3) Aibo Remote Framework, that allows PC-based applications to monitor and control the Aibo.

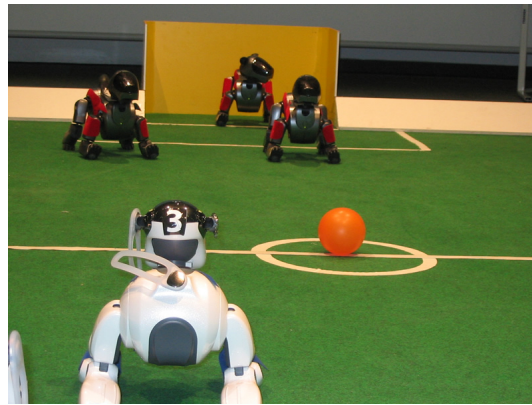


Figure 2.3: Aibo Soccer

2.3.2 Humanoid Robots

In the year 2002, the RoboCup Federation added a new league for humanoid robots to its annual competitions. Humanoid robots are robots with human-like bodies and human-like senses. The robots are between 30 and 130cm of height [19] and function autonomously. Despite a lot of effort, basic skills such as walking and kicking are still far from perfect and even the best robots will occasionally fail at this.

Every few years, the rules are tightened, for instance through decreasing the maximum foot size, to make the game gradually more difficult, and more like human soccer [20]. The first soccer match between humanoid robots was held at the 2005 competition, where a 2 vs 2 match was held. (See Figure 2.4)

Many companies and researchers have worked on designing and developing humanoid robots for this competition [21] [22] [23] [20]. The resulting robots are still very experimental, and there is no de facto standard like the Aibo is for the four legged robot league. As this league is still very young, it is likely to receive a lot of attention in the (near) future.

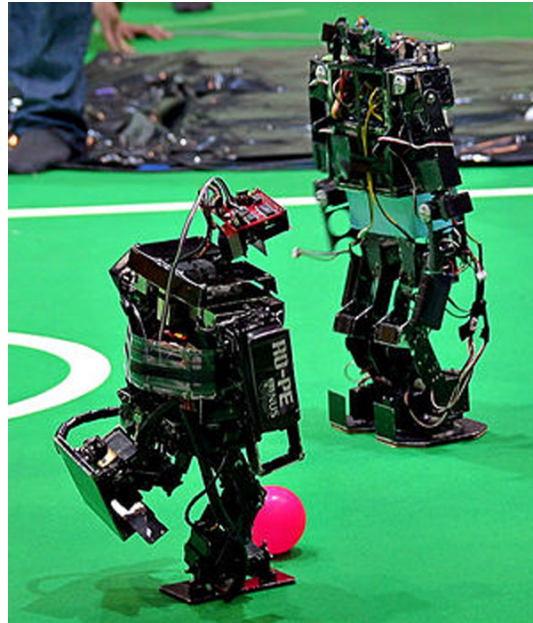


Figure 2.4: Humanoid Soccer Robots at the 2005 Competition

2.3.3 Soccer Simulator

The RoboCup Federation has another important league; the soccer simulator [24]. This simulator was introduced in 1995 [25] and has seen a lot of development since. In this league, no actual robots are used, but teams of 11 autonomous software agents compete on a simulator in a match lasting 10 minutes.

Unlike the four legged and humanoid leagues, this league is not faced with the hardware constraints of current technology, and a vast body of literature has been written on various techniques used for creating cooperative teams, such as machine learning, multi-agent collaboration, and opponent modeling.

The Soccer Simulator is a platform for researching cooperation and team work. It is not restricted by the current state of technology, however there is a large threshold for starting working with the simulator for several reasons.

There is several variations within the simulation league; A 2D competition, a 3D competition, a 3D-development competition and a coaching competition.

Following are some of the rules of the 2006 RoboCup 2D Soccer Simulation competition [26]:

- Scripts will be executed by a different user in your user group. Your scripts and your team have to be at least group read- and executable.
- The scripts should use absolute paths or change to the respective directories.
- Double check that your kill scripts kill all of your programs (goalie, players, coach) even if your programs terminate automatically.
- Teams can only use Linux operating system.
- The scripts will also be called for penalty kick out. Make sure your team can handle this.

Furthermore, a large *fair play* clause is added to the rules:

Violation of the fair play commitment play includes for example:

- using another teams binaries in your team;
- if a team is jamming the simulator by sending more than 3 or 4 commands per client per cycle;
- if a team communicates by other means than via the server using the say command, for example by using direct inter-process communication;
- if a team attempts to disturb other teams communication by recording and sending strings of former communication or by attempting to fake communication of the opponent team.

Of course the fair play clause is sensible, but it also points out that the framework itself is not capable of controlling such fair play. The rules pointed out above lead to a similar conclusion. Double checking whether proper access rights are given, that you have a working kill script, that your team will work when it is called for a kick-out are things that a properly designed system should check for you, before even engaging in any competition. Finally, the demand for the usage of the Linux operating system is very restrictive towards supporters or different operating systems.

The communication in the 2D league has to be done using TCP/IP, and the only supported platforms are SuSe Linux, Redhat Linux, Solaris, Mac OS. Windows is not mentioned in the manual. (See Figure 2.4 for a screenshot of the 2D simulator).

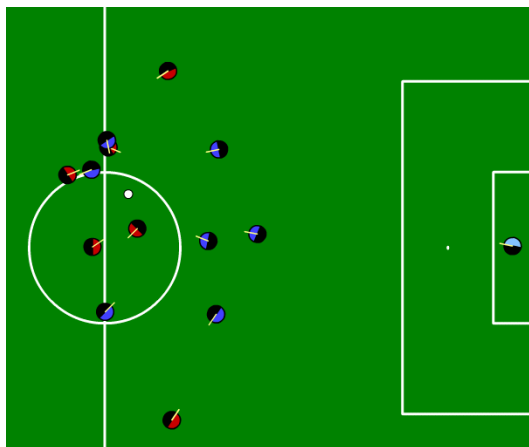


Figure 2.5: 2D Soccer Simulator

All communication with the server has to be done over TCP/IP. The only supported platforms are SuSe Linux, Redhat Linux, Solaris, and Mac OS. Windows is not mentioned in the manual [27], and as such Windows users are not able to work with the soccer server. The simulator uses g++, and as such teams have to be programmed using C++.

Libraries exist for developing software agents in the simulator [28].

The 3D simulator is built on a generic simulator for physical multi-agent simulations, Spark [29], and is designed a lot better. It can be used on Windows and is a lot more user-friendly than its 2D colleague.

2.3.4 Other Simulators

Apart from the two simulators of the RoboCup Simulation league, more simulators have been developed. An important question to ask with simulators is *what are they simulating?* The obvious goal for the RoboCup simulation league is to simulate a realistic humanoid soccer environment. There is however still a large gap between the current state of technology and such a realistic soccer environment. Due to the expensive hardware involved in experimental actual robotic soccer, and the facilities required to host experiments, it is not very practical to field-test everything. A number of simulators have been developed that use models of the robots, and feed them with the same data as the actual robots would get. This allows for cheaper and faster testing, and can develop new technology that anticipates upcoming hardware advances in the actual robots.

The following are a number of such simulators that have been developed recently.

- Ubersim: A high-fidelity C++ simulator for the small-sized RoboCup league [30].
- SimRobot: A generic physical robot simulator, that has been used to model the German Aibo team [31].
- Player/Stage: Player is a network-transparent robot control interface, Stage is a lightweight simulator [32].
- WeBots: Webots is a mobile robotics simulator that has been used for simulation (simple) soccer robots [33].
- UCHILSIM: A realistic simulator designed to simulate the four legged league of RoboCup. [34]

2.3.5 Multi Agent Soccer in Education

The domain of multi-agent soccer is especially suitable for applications in higher education. The game is well known and very popular, so the problem is realistic and challenging. It has been used in undergraduate and graduate computer science since as 1997.

Within these projects / courses on robot soccer, there is a clear distinction between two variants:

1. focusses on both the hardware, the hardware/software interface and the software. Actual robots are built or programmed, and usually a competitive element between student groups conclude the coursework [35] [36] [37] [38] [39];...
2. focusses solely on the software, using the Soccer Server [25] as its main reference [40] [41] [42] [43] [44].

As programming a full intelligent team in Soccer Server will take tens of thousands of lines of code, it is undesirable to start from scratch. To solve this problem, RoboSoc [45] was developed. RoboSoc can be seen as a layer over the Soccer Server that makes it more user friendly, and allows easier control over low-level details, allowing students to focus on high level implementation details faster.

2.4 Overview

This chapter introduced Agents and Multi-Agent Systems. The Four legged, Humanoid and Simulator leagues of the RoboCup competition were described, and an overview of the state of the art was given. A number of other simulators that have been used for soccer were described, and it was shown how multi-agent soccer is used in education.

Considerable progress is being made in all areas. Most soccer simulators apart from the Soccer Server are specifically designed for simulating actual robots playing soccer, whereas the Soccer Server is designed to simulate human soccer. Many courses and projects in higher education that deal with soccer choose to include the hardware aspects as well, resulting in actual robots that display some intelligence, rather than highly intelligent virtual robots. Some courses are based on the RoboCup simulation league and use Soccer Server in conjunction with RoboSoc, a system that makes it easier to implement teams in Soccer Server.

Chapter 3

Tools

Intelligence is the faculty of making artificial objects, especially tools to make tools. Henri Bergson.

This chapter describes the tools that were used in this project. This chapter begins by providing an overview of what is required, following is a description of Fleeble - the agent framework that was used, in terms of the requirements. Finally, a brief overview of Eclipse and some of its plugins is given.

3.1 Overview

Just like the development of any other software application, careful attention has to be paid to the selection of the tools. The development will have to be done using a certain programming language, using a certain Software Development Kit (SDK), and using a certain agent framework to match. Prior to selecting any of these, some basic questions need to be answered:

- Who will use the system and what is their background knowledge?
- Which characteristics are important? Speed? Performance? Ease of use? Platform independence? Distributed or centralized? Stand-alone or cooperative multi-player games?
- What programming language is best for development of the particular system?
- What programming language is best suited for the intended users to program their teams in?

3.2 Programming Language

Prior to selecting an agent framework, it is important to decide on the programming language to be used. All languages have their strengths and weaknesses. Considering the application – a 2D simulation of a soccer match, where players are simulated and controlled by agents; speed and real-time performance of the simulator are not as much of an issue as in for example complex traffic simulators. The low-level control that languages like C or C++ offer might make them more efficient, their low-level capabilities also distract programmers from the higher level issues. This is especially the case when design experience is lacking - as is obviously the case with students. The ease of use at a high level of abstraction is more important than speed and performance. When dealing with multi-agent systems, thread management becomes an important issue for the software developers.

3.2.1 Educational Aspect

For the requirements, it is important to keep in mind that the assignment produced in the educational part of this thesis will rely heavily on the simulator. As such, choosing an agent framework that has a high learning curve because of poor design, too many options, or use of a language or syntax that the students are not familiar with would be problematic. The assignment would then require a lot of extra tutorials and support. The focus of the assignment would change towards learning the new environment rather than learning AI techniques. Using Java and a Java-based agent framework seems like a natural choice, as Java has become the default programming language for undergraduate computer science students at Delft University of Technology.

3.2.2 Conclusion

Speed and real-time performance are important features of the system, but the speed of regular desktop computers is more than high enough to support such small-scale soccer simulations in any programming language. Also for potential larger scale simulations distributed over a network, present networking speeds are more than sufficient. The fact that a large subset of the potential users will have limited programming experience - and most of this experience in Java, combined with Java's object-oriented approach that is very suitable for modeling agents, make Java the right choice for this project.

3.3 Agent Framework

This section describes the criteria for the selection of an agent framework, justifies the decision to use Fleeble, and provides further background information about Fleeble.

3.3.1 Background Information

Although agent technology is a relatively new area in the domain of artificial intelligence, and the applications relying on agent technology put very specific demands on their frameworks, a large body of literature and software has already been produced to support people in working with agents. Agent frameworks, including software packages and agent-based applications are now readily available for new research to build on (See [46] for a long list).

3.3.2 Role of the Agent Framework

The role of the Agent Framework in the multi-agent soccer system is dual. First, it provides a generic interface and tools for the communication between the framework and the individual soccer agents. The Soccer Server [25] uses the TCP/IP protocol to host its communication, whereas an agent framework will often have its own communication model, that might be easier for novice users. Functionality of certain agent framework tools can also serve to restrict soccer agents to prevent cheating and to force focus on important areas. Secondly, the agent framework is a de facto *black box* that students doing the coursework will have to work with. They will have to understand exactly what goes on and what options are at their disposal.

To deal with the latter, [7] proposed some criteria for an agent framework that is to be used in the introductory AI course in question:

- It should be simple and user friendly, and it should include built-in Java-implemented agent "templates" that facilitate example-based learning and can be edited to include the desired AI algorithms according to the goals of a programming assignment.
- It should embody the concepts of concurrency (a kind of multi-threader set-up), multi-agency (by allowing simple communication from one agent to the other), mobility (transferring agents from one host to another) and persistency (saving settings between executions).

3.3.3 Comparison

Based on the aforementioned requirements, a Simple Agent Framework [8] was developed. After a major rebuild, this was renamed Fleeble [9][11], and

was developed by Grootjans & Zwitserloot. An interesting comparison of 26 Java-based agent frameworks was conducted [7], based on issues derived from the aforementioned criteria. These issues are listed below. The comparison is illustrated in Table 3.1. Based on these criteria, Fleeble was found to be the best option.

1. Is the tool available for free?
2. Does the developer provide support for the tool?
3. Is the tool available for free?
4. Are useful examples readily available?
5. Is the related documentation readable?
6. Is synchronous agent-to-agent communication (i.e., waiting for reply) supported?
7. Is asynchronous agent-to-agent communication (continuing immediately) supported?
8. What is the communication transmission form?
9. Can the framework control agents resources (e.g., disk or network capacity used)?
10. Can the framework ask an agent to shut down?
11. Can the framework terminate the execution of a malfunctioning agent?
12. Can the framework store agents states between executions?
13. Can the framework store objects (e.g., a database) between executions?
14. Does a self-explained GUI per agent exist?
15. Does the GUI support an overview of all running agents?
16. Does the framework support mobility?

3.3.4 Java 1.5

Fleeble is programmed in Java 1.5 [50]. With the introduction of Java 1.5 in late 2004, Java has become a much more powerful language. Java 1.5 introduced new features like generics and annotations. These features are used a lot throughout Fleeble, but are quite intuitive and easy to learn.

Students that have worked with Fleeble and Java 1.5 had little trouble adjusting to the new syntax and were enthusiastic about the possibilities offered by the new features.

Table 3.1: Overview of available Java-based agent frameworks [7]. • = "yes", × = "no", PP = Peer to Peer, M = Multicast, PS = Publish-Subscribe.

Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Agent Factory [46]	•	-	×	×	×	×	PP	×	×	×	×	×	×	×	•
IBM Aglets [46]	•	Free	×	×	•	•	PP	•	•	×	×	•	•	×	•
AMETAS [46]	•	-	×	×	×	•	M	×	•	×	×	×	•	×	•
Beegent [46]	×	-	•	×	×	•	PP	×	×	×	×	-	×	×	•
Cougaar [46]	•	Free	×	×	×	•	M	•	×	×	×	•	×	×	•
CIAgent [47]	•	\$	•	•	×	•	PP	×	•	×	×	-	•	×	×
DECAF [46]	•	-	×	×	×	•	PP	×	×	×	×	×	•	×	×
FIPA-OS [48]	•	Free	×	×	•	•	PP	×	×	×	×	•	×	×	×
Grasshopper [46]	•	-	•	×	•	•	M	•	-	-	•	×	•	•	•
Hive [46]	×	Free	×	×	×	×	None	×	×	×	×	×	•	•	•
JACK [46]	•	\$	×	×	×	•	PP	×	×	×	×	-	×	×	×
JADE [46]	•	Free	•	×	×	•	M	×	•	×	×	×	•	•	•
JAFMAS / Jive [46]	×	-	×	×	•	×	M	×	×	×	×	×	×	×	×
Kaariboga [46]	•	Free	•	•	×	•	PP	•	•	×	×	×	×	×	•
LIME [46]	•	Free	•	•	×	•	PS	×	×	×	×	×	×	×	•
MadKit [46]	•	Free	•	×	•	•	M	×	•	×	×	×	×	•	•
NOMADS [46]	×	Free	×	×	×	×	None	•	•	•	×	×	•	×	•
OpenCybele [46]	•	Free	•	•	•	•	PS	×	•	•	×	×	×	×	×
Pathwalker [49]	•	Free	×	×	×	•	PP	×	×	×	×	×	×	×	•
SeMoA [46]	•	Free	•	•	×	•	None	•	•	•	•	•	×	×	•
Tryllian [46]	•	\$	•	×	•	×	PP	×	-	-	•	•	×	×	•
Voyager [46]	•	\$	×	×	•	•	PS	•	×	×	•	×	×	×	•
ZEUS [46]	×	Free	•	×	•	×	PP	×	×	×	×	×	×	×	×
SAF [8] [12]	•	-	•	•	×	•	PS	•	×	•	×	•	•	•	×
Fleeble [11]	•	-	•	•	×	•	PS	•	×	•	•	•	•	•	•

3.3.5 History

The Fleeble Agent Framework is a simple Java-based agent framework, designed for teaching introductory AI [9] [11] [7]. Work on Fleeble's predecessor, Simple Agent Framework (SAF) [8] started in 2001-2002 as a first working prototype for an agent framework for the introductory AI course. It was used and evaluated that year, and redesigned the year thereafter. Along with the redesign came its new name; Fleeble. Minor and major bugs have occurred throughout the first couple of years, often in conjunction with the addition of new functionality. Since 2004-2005, Fleeble is considered stable and reliable, and all known bugs have been resolved. Fleeble is open-source

and thoroughly documented.

3.3.6 GUI

The user interface of Fleeble was designed to be simple. A typical session of Fleeble will look something like Figure 3.1. On the left are all the namespaces and their channels. On the right are all currently active agents. On the bottom is the output of a channel, in this case the *System.out* channel. On the right, the pop-up menu shows the current input channels for the Soccer Agent.

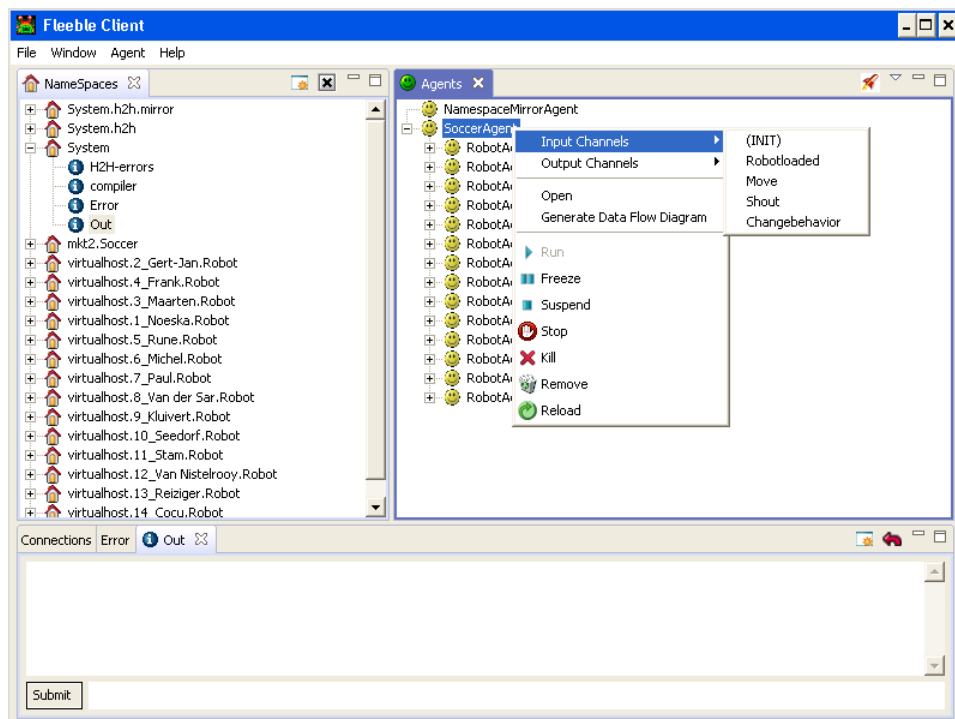


Figure 3.1: GUI of Fleeble Agent Framework

3.4 Fleeble

This section serves as an overview of Fleeble. A number of key principles, definitions and methods are explained. This serves to illustrate the simplicity of creating and working with agents in Fleeble, and to create an awareness of its possibilities. An understanding of the concepts dealt with in this section shall be assumed for the chapter regarding the implementation of the soccer simulator. A more detailed explanation of most of these features can be found in Fleeble’s extensive documentation [10].

3.4.1 Programming an Agent

Programming an agent is illustrated in Listing 3.1.

Listing 3.1: Programming an Agent

```
1 import fleeble.*;    //required.
2
3 public class MyAgent extends Agent
4 //agents must be PUBLIC.
5 {
6     //agents must extend 'Agent'
7     //nothing else is required!
8 }
```

3.4.1.1 Compiling the Agent

Compiling the agent can be done in the *conventional* way, for example using the default *javac* compiler, but also in a user-friendly way by Fleeble itself. When selecting the agent that is to be loaded, the Java file can be selected, and Fleeble will automatically compile the code.

3.4.1.2 Loading the Agent

Loading an Agent is just as simple as starting Fleeble, going to the *File* menu, clicking on the *Load Agent* button (See Figure 3.2) and selecting the .java or .class file containing the (compiled) agent.

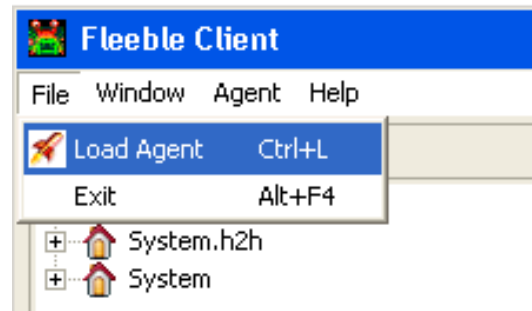









Figure 3.2: Loading an Agent in Fleeble

3.4.2 Agent States

An agent can be in one of the following states:

Table 3.2: Overview of Possible Agent States and their Icons.

Agent State	Icon
CREATED	
RUNNING	
FROZEN	
SUSPENDED	
STOPPED	
KILLED	
REMOVED	

The state of an agent determines what it can or can not do. The state that an agent is currently in is displayed using an icon. (See Table 3.2) A thorough explanation of states is given in [10]. The function of most states is straightforward and self-explanatory. The most important state is RUNNING. This is the default state, and while in this state, the agent will be able to send and receive messages when in this state.

3.4.3 Channels

Communication in Fleeble is done using channels. Channels can be thought of as chatrooms. Agents can *subscribe* to a channel and all data on the channel will be delivered to the agent. Channels can carry any type of data. Any Java object that is serializable¹ can be sent over a channel. Sending data to the channel can be done through *publishing*.

Table 3.3: Publishing and Subscribing to Channels

Subscribing	<code>@Subscribes("soccer.input", "soccer.status")</code>
Publishing	<code>@Publishes("soccer.output", "soccer.result")</code>

Table 3.3 shows the code that has to be added before the class declaration in order to subscribe or publish to a channel. The syntax uses Java 1.5's Annotations [52]. Subscribing and publishing can be done in a static and

¹See [51] for an explanation of serialization. A serializable object can be stored and transmitted over a network. For objects to be serializable they must either extend the *Serializable* interface or be a primitive type

in a dynamic manner. Dynamic publishing and subscribing is explained in section 3.4.3.3. Listing 3.2 illustrates static subscribing and publishing.

Listing 3.2: Publishing and Subscribing

```

1 @Subscribes({ "soccer.Input" , "soccer.Status" })
2 @Publishes({ "soccer.Output" , "soccer.Result" })
3
4 public class MyAgent extends Agent {

```

yields in an agent that is subscribed to *soccer.Input* and *soccer.Status*, and allowed to publish on *soccer.Output* and *soccer.Result*.

3.4.3.1 Subscribing

The agent will receive all data that passes over this channel. Data can enter the channel through other agents, or through humans manually entering data on the channel. For this particular example to work according to expectation, the agent is required to handle the incoming channels. To do this, two methods have to be declared, as illustrated in Listing 3.3.

Listing 3.3: Handling the incoming messages

```

1 public class MyAgent extends Agent {
2     public void handleInput(MyObject pObject) {
3         // any java code
4     }
5
6     public void handleStatus(StatusObject pStatus) {
7         // any java code
8     }
9 }

```

Using Java 1.5's powerful generics [53], Fleeble will automatically forward any data on the specific channels towards these methods.

The *MyObject* and *StatusObject* are included to illustrate yet another powerful feature of Fleeble.

Agents are not ignorant of each other, but deal only with their inputs and outputs. Each agent has a certain job to perform. To perform this job, it expects a certain type of input, and is expected to return a certain output. An easy metaphor would be a car factory. A large assembly line consisting

of several hundred employees together build a car, whereas each employee will perform only one or two tasks on this car. If there are n employees, then an arbitrary employee staged at point k in the assembly line of state k will expect a car that is completed until state $k-1$ as its input, and return a car that is complete until k as its output.

In a multi-agent environment a system developer will always know what type of data can be expected. It was previously mentioned that any Java *Object* can be sent over channels. Since the type of object that can be expected is known, Fleeble allows agents to further specify this in the handler methods. This makes programming agents a lot easier, as it avoids tedious casting of objects, but it also makes them more intuitive.

If a channel can be used for multiple types of input, another method expecting the other kind of input can be implemented. Fleeble will automatically look for the method that matches the data. If no match can be found, Fleeble will try to route the message to *handleOther(Message pMessage)*. If this method does not exist, an error will occur. This is illustrated in Listing 3.4.

3.4.3.2 Publishing

The agent will now also be able to publish on channels. Sending a message to a channel that the agent is allowed to publish on is done in the following way:

```
1 ... ambassador.send("soccer.Output", anyObject); ...
```

where *anyObject* is the object that the agent intends to send. Bear in mind that another agent waiting for input on the *soccer.Output* channel will expect this to be in a certain format. As such, the *anyObject* should conform to this format. Not acting in accordance with this expectation will channel the object to the *handleOther* method.

3.4.3.3 Dynamic Publishing and Subscribing

Apart from the *static* way of publishing and subscribing to channels, Fleeble also features dynamic (un-)publishing and (un-)subscribing. This is particularly useful when the environment changes based on for instance user input, or other events. The methods that can be used for this are shown in Table 3.4

Listing 3.4: Advanced handling of incoming messages

```

1  public void handleInput(MyObject pObject) {
2      // this is called when an Object of the type
3      // MyObject
4      // is sent over the channel
5  }
6
7  public void handleInput(MyOtherObject pOtherObject)
8  {
9      // this is called when an other object, of type My
10     // OtherObject is sent over the channel
11 }
12
13 public void handleOther(Message pMessage) {
14     // this is called when no other matching method
15     // can be found
16 }
17
18 public void handle(Message pMessage) {
19     // the default implementation of this method makes
20     // sure messages
21     // arrive automatically at designated methods.
22     // Overwriting it
23     // ensures all incoming messages are routed
24     // through this method.
25 }

```

Table 3.4: Dynamic Publishing and Subscribing

Function	Method
Publish	ambassador.publish("soccer.Somechannel");
Unpublish	ambassador.unpublish("soccer.Somechannel");
Subscribe	ambassador.subscribe("soccer.Somechannel");
Unsubscribe	ambassador.unsubscribe("soccer.Somechannel");

3.4.4 Thread Scheduling-induced Randomness

In Fleeble, every agent has its own thread of execution. Data coming over the channels that an agent is subscribed to is handled by this single thread. Messages that arrive while the agent is busy processing a previous message

will be put on a queue, and processed on a first-in-first-out basis. Java has an internal thread scheduling algorithm that will handle all threads in a predefined manner.

As Fleeble is also built using threads, the order in which the channel handlers are processed by Java's internal thread scheduling can not be anticipated, but the order of processing of messages is guaranteed on a per queue basis. As such, a sequence of messages sent by one agent will be guaranteed to remain in sequence provided that one and the same channel is used for each message in the sequence. When two agents send a message at the same time to the same channel, which of the agents - and thus messages - will be processed first depends on Java's thread scheduling. Furthermore, depending on Java's thread scheduling, the first message might be handled by the corresponding channel thread, placed on another agent's incoming message queue - and perhaps even processed by the receiving agent, before the second message is even retrieved. On a long-term average, all threads will receive equal processing power, and no individual agent has an unfair advantage. On a short term however, thread scheduling can play a significant randomizing role in time-dependant environments.

The understanding of this mechanism is very important, as it determines nearly all of the randomness in the system. Especially when dealing with larger multi-agent systems in Fleeble, this property needs to be considered in the system design.

3.4.5 Namespaces

Working with multi-agent systems can yield a large number of channels. Working with multiple systems at the same time; for instance a soccer simulator and a facial action recognition system might cause problems, as both are likely to use an *input* channel. Rather than prefixing channel names using awkward abbreviations, unnecessarily complicating your code, Fleeble introduced namespaces. In the above example, all channels appear in the *soccer* namespace. Namespaces are quite comparable to Java packages [54]. Figure 3.3 shows an example with 4 namespaces. The channels are all prefixed with the namespace for subscribing and publishing, but handling the channels does not require a namespace. Publishing to the *Move* channel would require

```
1 @Publishes ( { "mkt2.Soccer.Move" } )
```

whereas handling it is done regardless of the namespace;

```
1 public void handleMove(MoveObject pObject) {
```

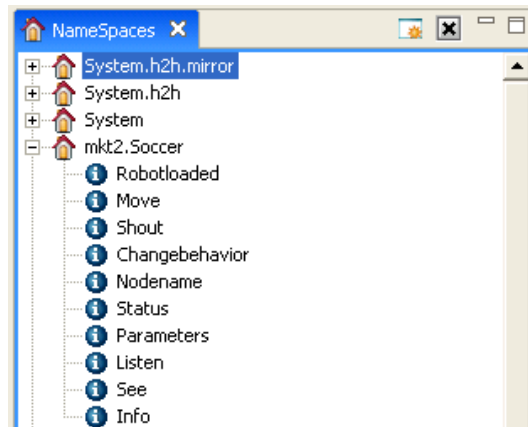


Figure 3.3: Fleebble Namespaces

3.4.6 Special Channels

Other than the user-defined channels, several *special* channels exist; (INIT), (CHILD_LOADED), System.out, System.error.

3.4.6.1 (INIT)

First of all, any agent can subscribe to "(INIT)".

```

1 @Subscribes({ "(INIT)" })
2
3 public class MyAgent extends Agent {
4     public void init() {
5         // subscribing to (INIT) causes this method
6         // to be called when this agent is loaded
7     }

```

3.4.6.2 (CHILD_LOADED)

Second of all, when an agent loads another agent, this agent is now its child agent. To know whether the operation was successful, the parent can subscribe to "(CHILD_LOADED)".

```

1 @Subscribes({ "(CHILD_LOADED)" })
2
3 public class MyAgent extends Agent {
4     protected @Override void childLoaded(ChildLoaded
5         info, MessageSource source)
6         // the ChildLoaded object contains information
7         about the cause

```

```

6      // of the child being loaded, the MessageSource
          can be used for
7      // for instance to change the status of the child
          agent.
8  }
```

3.4.6.3 System

Using normal Java editors, or running Java from the command line, (debug) information is usually printed to the default outputstream using `System.out.println("my text here");`. As Fleeble is intended to be used as a standalone tool, it features several *System* channels, see Table 3.5.

Table 3.5: Fleeble's System Channels

Channel Name	Description
System.Out	Everything that is printed to <code>System.out.println</code> is redirected to this channel.
System.Error	Exceptions that occur in your code are normally printed using <code>System.err.println</code> . Anything printed to <code>System.err.println</code> is redirected to this channel.
System.compiler	Exceptions related to the compilation of agents appear in this channel (Remember that Fleeble can compile agents automatically).

3.4.7 Message Queuing

Agents in Fleeble each take up one Java thread. All incoming messages are automatically placed in a queue. Once an agent is done processing one message, the next message on the queue will be processed. This is an important mechanism that has to be understood by developers prior to programming complicated agents, as it is a frequent cause of mistake. Thread scheduling-induced randomness is another property of this approach, and was explained in Section 3.4.4.

The fragment of code in Listing 3.5 would send *"look!"* every 100 milliseconds. Assuming that the particular multi-agent system would result in a reply on for instance *"soccer.See"* with whatever it is that the agent can see, this might seem like a natural way of ensuring such information is provided every 100 ms. However, because the agent contains of only one

Listing 3.5: Send 'look!' every 100ms

```

1 public void init () {
2     Object lWaiter = new Object ();
3     while ( true ) {
4         synchronized (lWaiter) {
5             try {
6                 ambassador.send ("soccer.Look", "look!");
7                 lWaiter.wait (100L);
8             } catch (InterruptedException e) {
9                 e.printStackTrace ();
10            }
11        }
12    }
13 }

```

thread, the message queue will fill up with replies on *"soccer.See"*, causing a *"StackOverflowException"* eventually.

Two solutions to get the desired behavior can be thought of:

```

1 public void init () {
2     ambassador.fork ();
3     ....
4 }

```

Using *ambassador.fork* will cause a second thread to start processing the queue.

```

1 public void init () {
2     ambassador.send ("soccer.Look", "look!");
3 }
4
5 public void handleSee (Info pInfo) {
6     //process the See info first
7     ....
8     ....
9     ambassador.send ("soccer.Look", "look!");
10 }

```

This second approach starts with a single *look* request, and upon receiving a *see*, will send the next.

The (potential) flaw in the latter method is when the agent that reads the *look* requests and returns *see* information is not loaded yet, or has crashed.

As the initial *look* is lost, there shall be no *see* information incoming. Many solutions to this can be thought of, and it is often the preferred way of working with messages, as it prevents overflowing of message queues.

3.4.8 Child Agents

Child agents can be loaded by an agent in a very straightforward manner. The use of child agents can vary, but it is commonly used for the following two tasks:

1. Load a number of other agents. This is the *convenience* usage of child agents. Only a single *LoaderAgent* will have to be run in order to start the entire Multi-agent system, while ensuring that all required agents are loaded in the desired order (if this is a relevant issue).
2. In dynamic multi-agent systems. This is explained in detail in later chapters about the Simple Soccer Simulator. As an example, the agent that is to control a robot is loaded by the *RobotAgent*.

As can be seen in Figure 3.4, the *SoccerAgent* resides on the highest layer, and has 14 child agents; the *RobotAgents*. Each *RobotAgent* has loaded his own specific player agent. The reason these agents *have* to be loaded as child agents is discussed in the next section.

3.4.8.1 MessageSource

In section 3.4.6.2 it was described that subscribing to the (CHILD_LOADED) channel gave a *ChildLoaded* and a *MessageSource* Object. The *MessageSource* object can be extremely useful for several purposes:

1. Replying. Normally in Fleeble, communication is done using channels and all subscribers to a channel will see all communication. A problem occurs however when dealing with agents that manage database requests. A large number of agents can request specific data, and the resulting data from every request would be delivered to all subscribers of the channel. Apart from privacy issues, and the fact that all subscribers would need to implement some kind of filter to use only the requested data, such amounts of communication would flood the system. It is possible to reply to any particular *MessageSource* over a particular channel. This results in the data only being visible for the particular recipient, and thus saving a lot of extra work.
2. Changing status. *Ambassador* has several (restricted) methods that can be used to change the status of an agent, if the agent has enough rights to do so. A parent always has the right to change the status of his child agent, and it uses the *MessageSource* to do this.

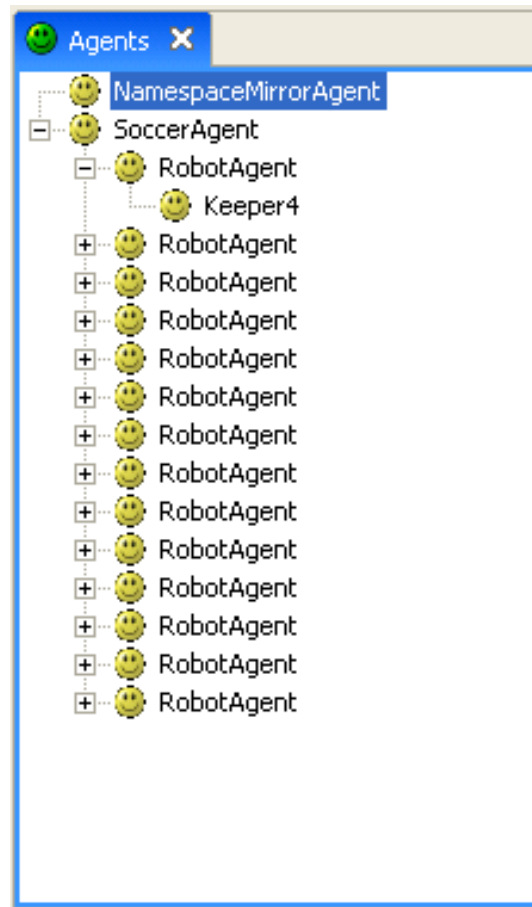


Figure 3.4: Child Agents in Fleebble

3. Virtualhost. Another method of the Ambassador is to set a virtualhost on a child agent. What this does is *locking* the agent to a certain namespace. Figure 3.5 shows this. A child agent was loaded, and given *virtualhost.1_keeper* as its virtualhost. All channels that the child agent tries to publish or subscribe to will be automatically prefixed with *virtualhost.1_keeper*. Agents can not communicate using channels that are *above* their root, causing them to be locked in their virtualhost namespace. Other agents that do not have a virtualhost are naturally free to communicate wherever they like, and they *can* communicate with the agent. The agent displayed in Figure 3.5 has published / subscribed to *Robot.Move*, *Robot.Shout*, . . . This mechanism is particularly useful for simulating multiple computers, as it is a transparent layer that effectively restricts agents from contacting other parts of the system, which is a very desirable effect sometimes. Also,

it allows any *arbiter*, or *system / framework* agents to control the communication between the different subsystems using the dynamic publishing that was introduced in Section 3.4.3.3.

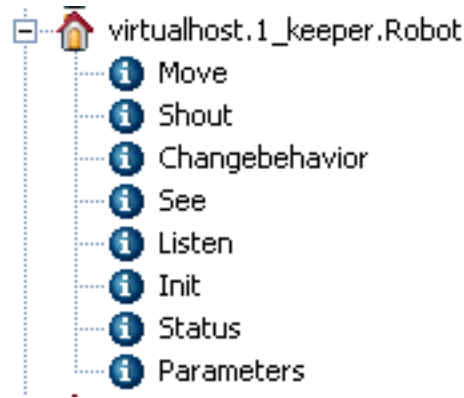


Figure 3.5: Virtualhost for Locking Agents in a Certain Namespace

3.4.9 Properties

Apart from communication through channels, Fleeble also features *properties*. Properties are basically nothing more than a set of $(name, Object)$ tuples, that are globally available. Properties are stored to disk when Fleeble shuts down, and are restored when it starts up. They can be used to store data, such as user preferences, but also as an alternate means of communication. Unlike channels, there is no way to subscribe on properties, an agent can only *get* or *set* a property.

Agents that are locked in a namespace through the virtualhost mechanism (See Section 3) do not have access to the global properties. All properties that are stored and used by these agents are prefixed by that virtualhost. The properties can be accessed by agents outside the virtualhost, but the agent affected by the virtualhost can not access the properties outside his virtualhost (i.e. one or more *levels* up).

Using properties in conjunction with virtualhost is not a way out of the lock. Properties of agents that have a virtualhost are prefixed by that virtualhost, and can not access properties outside this virtualhost (i.e. one or more *levels* up).

3.4.10 Namespace Mirroring

The features that have so far been discussed will only work for a single (local) instance of Fleeble. Fleeble also supports a transparent networking layer

that allows multiple Fleeble hosts on various computers and communicate over the same channels using a feature called Namespace Mirroring.

This feature can be used to 'mirror' all channels inside a namespace. Anything that appears on one side (computer) will also occur on the other side. Several hosts can be linked together to create a larger network of Fleeble hosts that operate over the same namespace.

An example of how to use the namespace mirroring feature is shown in the following piece of code:

```
1 ambassador.send(System.h2h.mirror.command, open
2 fleeble.tudelft.nl:1234 mkt2.Soccer);
```

This command would open a connection with the server *fleeble.tudelft.nl* at port 1234, and connect to the Fleeble host listening to that port. The namespace *mkt2.Soccer* would be mirrored between both Fleeble hosts, and agents residing on either host would not need to be aware of this. The feature effectively allows for a transparent network layer.

3.4.11 Overview

To provide a better overview of the specific features and the internal structure of Fleeble, an overview of the fundamental classes is useful. Figure 3.6 [7] shows the class diagram of these.

3.4.12 Summary

This section has shown a number of key features of the Fleeble agent framework. Although a lot of features are left unmentioned (in particular with regards to networking), it does provide a good overview of what Fleeble is capable of. The user interface is very intuitive, and since Fleeble will automatically compile and provide user-friendly feedback regarding bugs in your agent's code, Fleeble can be used effectively in conjunction with simple editors such as Notepad or ConTEXT ² [55].

In short, Fleeble:

- has an intuitive interface,
- features *Publish / Subscribe* communication using *channels*,

²ConTEXT is frequently used for introductory Java courses at Delft University of Technology

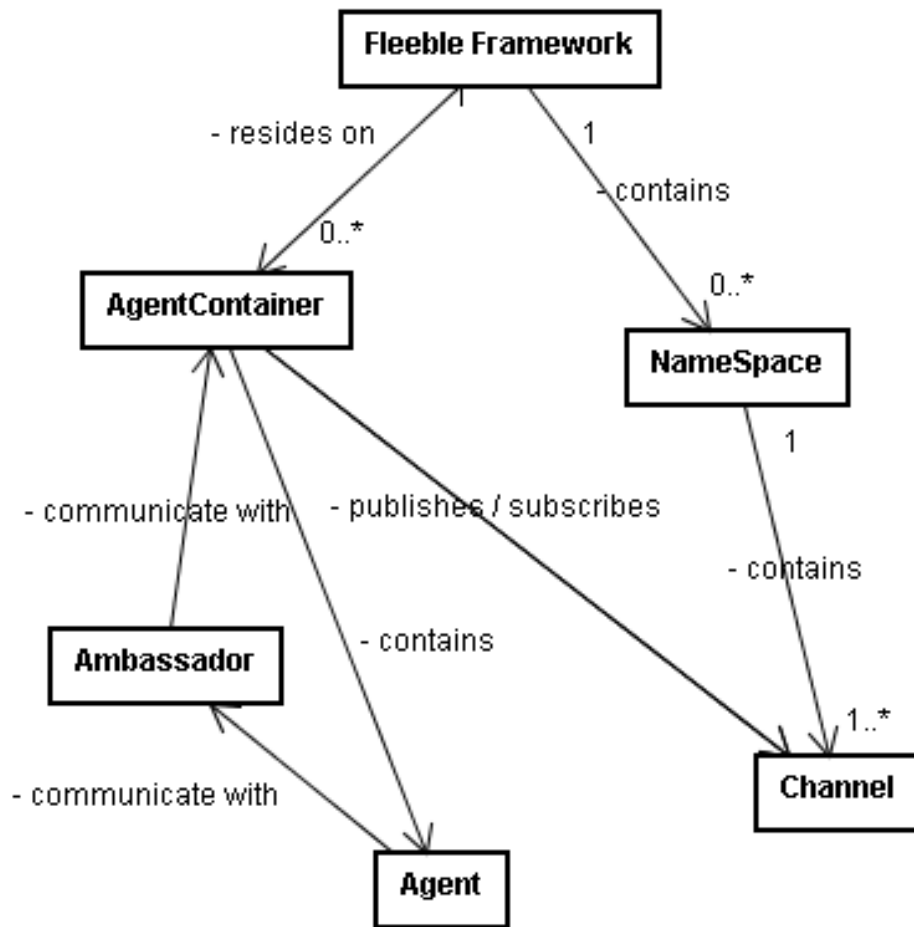


Figure 3.6: Class Diagram of Fleeble Agent Framework

- agents have a single thread for handling their messages, but multiple threads can be added using Fleeble's *fork* method,
- uses *namespaces* to separate different (multi-) agent systems,
- allows dynamic publishing or subscribing and the dynamic loading of agents. Parent agents have control over their child agents,
- can lock agents in a namespace using *virtualhost*,
- store data or user preferences in *properties*.

3.5 Eclipse

Eclipse [56] is an open source platform-independent Java-based development platform. Because of Eclipse's plugin architecture, support for many programming languages, including but not limited to: C/C++, Fortran, Java, PHP, Perl, Ruby and Python. The Eclipse user interface is illustrated in Figure 3.7

Some important features of eclipse are:

- easy development of Java code. Can automatically create methods, auto-compile the code, gives good feedback regarding bugs,
- debugging of Java code,
- version control. See section 3.5.1,
- Ant, a Java-based build management tool. See section 3.5.2,
- SWT, a Java graphical toolkit that was developed by Eclipse. See section 3.5.3.

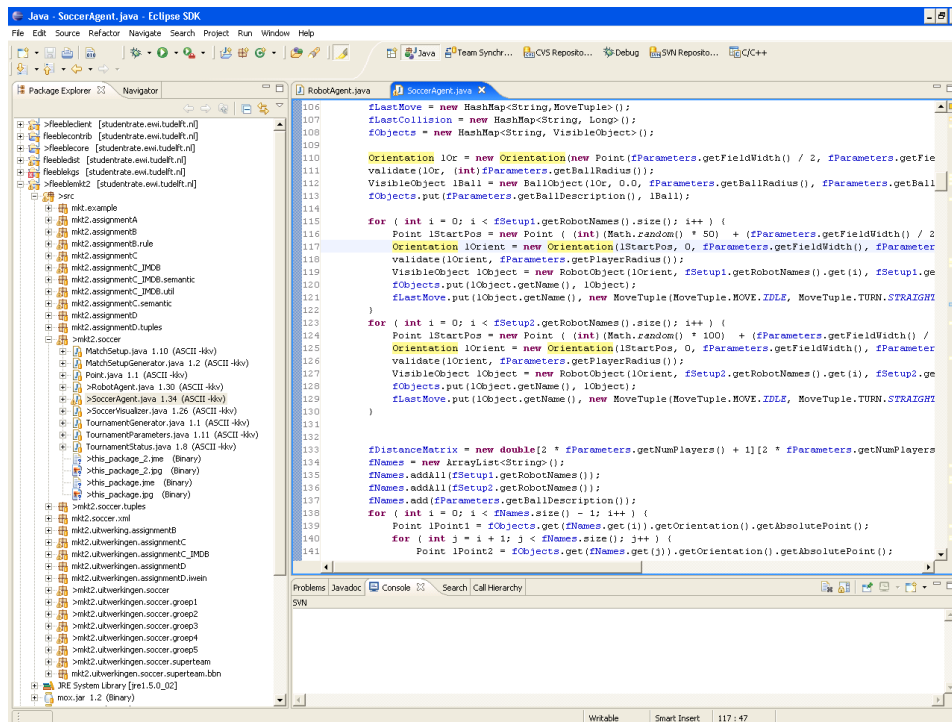


Figure 3.7: Eclipse Software Development Kit

3.5.1 CVS

Eclipse comes with a Concurrent Versions System (CVS) [57] plugin. This consists of a central repository where all source code is located, and members of a project group are able to commit their work to this server, or update the new code from the server to their computer. It keeps back-ups of all previous versions of a file, including a description of what was changed. CVS supports versioning in the design cycle, so one can experiment with competing implementations using the same codebase while maintaining good control. CVS supports multi-authoring and can merge modifications made by different authors in parallel, giving warnings if conflicts arise from modifications by any two authors.

3.5.2 Ant

Ant [58] is a Java-based build management tool. It functions quite similar to the Make [59] tool, which is well known in the Unix and/or the C/C++ world. Both tools are used for creating executables and installers (build management) from source files. Make uses a *makefile* that tells the program how the application files are to be generated from the source files and how to resolve dependencies. Ant uses an XML-based syntax that is parsed and run by Java classes, making it platform-independent.

3.5.3 SWT

SWT [60] is short for *Standard Widgets Toolkit*. It is a third-generation³ graphical user interface toolkit. SWT is a lot faster than Swing, because it uses native code⁴. It uses native widgets, which result in programs that "look-and-feel" a lot like the existing programs for the Operating System in question. It is also a lot simpler to use SWT than to use Swing [62].

³The first Java graphical toolkit was AWT[61], which features a large amount of native code and was generally considered as a poor toolkit. The second toolkit was Swing[62], which was overall a lot better, but due to its high-level implementation was relatively slow

⁴platform specific code. Is generally faster, but against the Java principle of platform-independence.

Part II

Game

Chapter 4

Model

Simplicity is the ultimate sophistication, Leonardo da Vinci

The problem that was described in Section 1.1.3 serves as a guideline in the requirements of the soccer simulator. This chapter begins by elaborating on the problem and the main requirements for the simulator. Then the laws of the game are explained, and compared to those of the official soccer game. The next section describes the system model. It describes the three layers that form the system, the links that exist between the layers and their function. The player capability model is described next. Then, a justification for several important decisions is provided. Alternative solutions will be described and compared. Finally, the extendability and applicability of the model is discussed.

4.1 Problem Description

The domain of soccer is inherently very uncertain. Table 1.1 illustrates the difference between the characteristics of the "solved" chess challenge, and the characteristics of soccer.

To adequately deal with these domain characteristics and establish a team that is capable of defeating the human world champions, research on low, medium and high level is necessary.

The incomplete information accessibility inherent to the soccer environment introduces many uncertainties. The current state of the art in parsing and interpreting sensor readings, and in executing low-level commands using a robot's effectors, introduces further uncertainties to the problem domain.

It is however likely that the technology will advance, and some uncertainties that presently add to the complexity of physical implementations

are likely to be removed in the foreseeable future. If it is assumed that the robotics hardware technology improves sufficiently in the foreseeable future, the next hurdle will be the implementation of cooperation and team work. Without these concepts properly addressed there will be no winning team.

4.2 Requirements

The goal of the soccer simulator is to address the problems laid out above, for the purpose of becoming a tool for AI research and education. Users should start on a high level of abstraction, rather than focus on low-level input, output and basic player skills. The uncertainties that are introduced by sensor readings¹ and incomplete control over the robot's effectors² should be removed. The focus should be on dealing with the partly unpredictable behavior of the (opponent) players, cooperation and team work. Table 4.1 illustrates the domain characteristics that result from the removal of these uncertainties.

Table 4.1: Domain Characteristics of the Simulator

	Soccer
Environment	Dynamic
State Change	Real time
Info. accessibility	Incomplete
Sensor Readings	<i>Symbolic</i>
Control	Distributed

4.2.1 Actors

Now that the general outline of the simulator has been given, it is important to identify the actors that will use the system. The specific requirements to the system depend on the different actors. These actors are:

1. Students computer science or related areas, with basic knowledge of programming Java and agent technology,
2. People with a personal interest in AI, that have at least basic programming skills, and
3. People with affinity to success and interest in MAS, AI, and education.

¹Visual sensor readings from cameras, auditory through microphones, and other sensors may exist

²Effectors include all movable parts, speakers, and often a (noisy) wireless network

To design a system that will be useful to these three groups of people, it is crucial that a flexible system is built that can be modified to suit personal preferences. As these groups are likely to use the system for different reasons and with different background knowledge, it is important that the requirements for all groups are incorporated in the system.

4.2.1.1 Students

For (computer science) students, it is important that the system is visually appealing, that it is easy to identify with the problem, that the coursework leaves room for creativity and experimenting, but restricts the students in such a way that they will not spend too much time on (low-level) details. Most importantly, it should be motivating and fun. It should be challenging, and very educative. It should also be such that different levels of students, such as undergraduate and graduate students, should all be able to learn a lot from working with the simulator³, and be motivated to proceed in the domain of AI. The interface between the framework and the user's code should be transparent. The focus of the coursework has to be on accomplishing the educational goals, such as learning about multi-agent systems and team work, rather than struggling with a new programming language and environment, or on resolving hardware errors.

It is important that working with the simulator does not require a lot of studying to make the basic environment work. Students should not be required to learn a new programming language, nor a very difficult interface. The interface should be intuitive, and preferably in the programming language they are most familiar with, Java. Example soccer agents should be provided to give students a kick start. Important aspects that contribute to the success and appreciation of courses by students depends on the educational approach used in the assignment. A constructivist approach is often favored over conventional objectivist approaches⁴. Courses about robot soccer will usually only take a small part in the curriculum, and as such creating a robot team from scratch is prohibitively difficult. It is next to impossible if the teams are supposed to display any intelligent cooperative behavior. The coursework for the Simple Soccer Simulator should give students a large amount of freedom with regards to the design and implementation, but it should also give clear objectives and an easy way to verify compliance with these objectives. *The purpose of the simulator is to enable students to create a cooperative team in a short amount of time by handling most low-level details, and by keeping the rules of the game as simple as possible.*

³The corresponding coursework should be adjusted to the level of the students; Graduate students could research for instance the performance of new AI techniques, whereas undergraduates could implement a cooperative team.

⁴See Section 9.1.1

Section 1.4 stated that the goal of the simulator would be to focus especially on the high-level details, such as cooperation and team work, rather than spending all available student time on low-level issues, as is the case in most alternative robot soccer projects. Projects working with real robots will obviously have certain advantages, as students learn about hardware, the hardware-software interface and deal with actual physical problems. Such projects also face disadvantages; working with actual robots restrains developers to the current state of robotics technology.

Alternatively, working with the existing Soccer Server and distributing tasks amongst different student groups might yield at least one team that works, but the students will not have learned to deal with a real problem at an academic level, and too little time will be spent on cooperation and team work. They will not have obtained a complete overview of building a team and the associated problems and decisions that come with the process of building a full team.

4.2.1.2 Hobbyists

For people with a personal interest in AI, all above criteria will hold, however several new criteria have to be included. Rather than relying on teaching assistants and help from professors in clarifying the assignment and the software, it is important that both are thoroughly documented and there is little ambiguity in the manual. All essential functionality should work as expected and it should be made easy to verify whether a strategy actually works. The package should come with example agents and teams to provide a kick start, and as a serious challenge to compete with.

4.2.1.3 Researchers

The most important difference for individuals with a professional interest in robot soccer is that these persons often have a lot of expertise, and may have working code from different platforms, that they intend to re-use. Rather than working with a game that has its parameters optimized for enforcing team work, experts might want to set the parameters to simulate the current state of technology, and use it as a simulation environment. As such, the system should be very flexible. With regards to whether or not messages can be sent and what information will be presented to the agent, it should be easy to add some code, change these features - and the rules of the game, according to the expert's demands. Whereas hobbyists and students will probably only be interested in working with the default system - and perhaps slightly modified parameters, professionals will want to be able to adjust more fundamental areas of the system, altering the way the system

works completely. This calls for thorough documentation of all elements of the game and the simulator.

4.2.2 Package

The soccer simulator will play a large role in accomplishing aforementioned goals, but other elements also play a key role in this. A decision on whether or not to start in a complex domain like multi-agent soccer depends not only on the learning curve of the simulator, but also on the user experience of the entire package. This package consists of a complete educational environment with characteristics matching the needs of a training environment. Of particular importance are:

- The soccer and functional model. How are the rules defined, what are the technical possibilities of the players and the simulator?
- Usability aspects of the soccer simulator. How easy is installation and configuration? Creation and management of teams? Simulating a basic game?
- The quality of the documentation. Documentation should be given on how to use the simulator, example code snippets, and instructions on how to change properties of the simulator.
- The availability and quality of a reference implementation. If these are available, users can immediately start with a working team. This is a lot more motivating than having to program a team from scratch prior to ever using the simulator. Such reference implementations can also be used as a competition in simulated games, or for *playing* the game for entertainment or analysis purposes.
- A suitable range of assignment(s) for the users (students) at various levels of complexity. Integrating the simulator in projects in higher education, or providing it as a stand-alone tool as a challenge to hobbyists can serve as an aid towards understanding of important AI concepts related to team work and cooperation.

4.2.3 Summary

Many decisions that have to be made for designing the soccer model. The key principles of *keeping it simple* [63], and *avoiding possible distractions from the main objective* play a large part in these decisions. Students in particular - but also hobbyists - may lack design experience. Providing them with a wide array of options, such as coaching, different player skills, penalties and corners, will change the focus from the general game towards dealing with these specific scenarios. Not giving them the option to be distracted to

prevent this is the preferred approach. There are several functional, and several non-functional requirements for the simulator. The latter include thorough documentation, user-friendliness, and the availability of example teams. The high-level requirements for the soccer simulator are summarized as follows:

1. Remove uncertainties regarding inputs / outputs
2. Simplify the rules of the game
3. Start on a high level
4. Make the simulator extensible, and allow users to add certain uncertainties to the model.

He who chooses the beginning of a road chooses the place it leads to. It is the means that determine the end. Harry Emerson Fosdick

4.3 Laws of the Game

In this section, the game is described and the laws of the game are compared to the human laws, as described in [2] [64].

4.3.1 Goal

The goal of soccer is to get the ball in the other team's goal, thereby scoring a goal. The team that has scored the most goals is considered the winner. If the score is equal at the end of the match, the game is a draw.

4.3.2 Players

Human soccer is played by teams of 11 players, of which one is the goalkeeper. Apart from the goalkeeper, players are not allowed to intentionally touch the ball with their hands or arms. All other body parts can be used to move the ball around. Naturally, all players have their individual strengths and weaknesses. A coach for each team is present outside the playing field, and can communicate strategic information to his players. During a game, players can be substituted by the coach. In most official matches, a maximum of three substitutions are allowed per game. A referee is present to ensure that play is in accordance with the rules of the game.

The soccer model features teams of 7 robots⁵. A player consists of a *robot* and its *behavior*. All robots are exactly identical, whereas the behavior is

⁵This parameters can easily be adjusted in the tournament parameters however

defined by the user agents. Robots are circular⁶ and 2-dimensional, and as such do not feature legs or arms. Although a robot can act as a goal keeper, there are no special privileges for this robot. Because all robots are identical, substitutions would not have any effect. The behavior of robots can be substituted, but not by a coach. There is no coach or any other outside interference allowed. Decisions regarding substituting the behavior have to be made by member(s) of the team. There is no referee, the rules of the game are controlled through the system.

Each robot can have its own *behavior* (player agent). As such, the success of a team depends solely on the performance of these player agents. Substituting the behavior on a robot effectively means that the player agent is killed, and the new behavior is installed on the robot. This process can only be initiated by the player agent originally residing on the robot agent, and can be executed at any time during the game. This feature allows for the (autonomous⁷) dynamic changing of the strategy.

The differences between human soccer and the model are summarized in Table 4.2. The capabilities of the individual players with regards to sensors and effectors are explained in Section 4.5.

Table 4.2: Human vs Simulator Players

	Official	Simulator
# players per team	11	7
individual player skills	yes	no
coach	yes	no
referee	yes	no
player size	varies	circular (1.85m radius)
body parts	body, arms, legs, head	body
substitutions	3 per game	behavior can be "substituted" at any time

4.3.3 The Field of Play

The official field width and height of human soccer are compared to those used in the simulator in Table 4.3. The goal is nearly twice as wide as the human goal, to compensate for the large size of the robots.

⁶The radius of the robot is 1,85m

⁷There is no outside interference, the decision to change the behavior is made by the individual agents

Table 4.3: Human vs Simulator Field of Play

	Official	Simulator
Field Length	90-120m	100m
Field Width	45-90m	75m
Goal width	7.32m	15.23m
Goal height	2.44m	N/A

4.3.4 Ball in and out of Play

Human soccer features two basic states of play; ball in play or ball out of play. The game begins with a kick-off. Until the end of the playing period, the ball is always in play, unless the ball leaves the playing field, or the referee stops the play. When either of these occurs and the ball is out of play, the ball can come back in play through either of the following: *kick-off*, *throw-in*, *goal kick*, *corner kick*, *indirect free kick*, *direct free kick*, *penalty kick*, *dropped-ball*.

In the soccer model, the ball comes into play when the game begins, at the center of the field. The ball is always in play after that. The ball can not leave the playing field, there is no referee to stop the play.

4.3.5 Foul Play and Misconduct

In human soccer, players can commit fouls, such as handling the ball, tripping or pushing an opponent. As a result of fouls, the referee can decide to put the ball out of play, and the player committing the foul can be punished by a yellow card (warning) or a red card (player has to immediately leave the field).

In the soccer model, there is neither a referee, nor yellow or red cards, nor the option to commit fouls. The ball is always in play.

4.3.6 Offside

One of the rules of human soccer is offside. This effectively restricts attacking players to remain forward of the ball and the second-last defending player. Violation of the offside law causes the ball to be placed out of play, and the defending team will receive a free kick.

The soccer model does not contain an offside law. Players are forced to be on their own half before kick-off, but are free to move wherever they desire afterwards.

4.3.7 Duration of the Match

In human soccer, a match lasts two 45 minute periods, separated by a 15 minute half-time interval. To make up for time lost due to injuries or substitutions, extra time can be added at the end of each period.

In the soccer model, a game lasts 10 minutes by default⁸. As there are no physical substitutions or injuries, there is no need for any extra time.

4.3.8 Environment

The environment of human soccer matches consists of an audience, a particular stadium, type of grass, a certain weather type (i.e. sun, clouds, rain),...

The environment of the soccer model is defined entirely by the framework, and there are no external factors that can influence the game.

4.3.9 Overview

An overview of the characteristics of the model compared to those of human soccer is provided in Table 4.4.

⁸This can be extended to suit personal preference. For the final competition in the educational part of this project, 30 minute matches were held

Table 4.4: Human vs Simulator Laws

	Official	Simulator
Field Length	90-120m	100m
Field Width	45-90m	75m
Goal width	7.32m	15.23m
Goal height	2.44m	N/A
# players per team	11	7
individual player skills	yes	no
coach	yes	no
referee	yes	no
player size	varies	circular (1.85m radius)
body parts	body, arms, legs, head	body
substitutions	3 per game	behavior can be "substituted" at any time
ball in play	unless outside playing field or stopped by referee	always
put ball back in play	kick-off, throw-in, goal kick, corner kick, indirect free kick, direct free kick, penalty kick, dropped ball	kick-off
fouls	handling ball, tripping or pushing opponent	no fouls
bookings	yellow and red cards	no bookings
offside	yes	no
duration	2 x 45 minutes, 15 minute half-time, possible extra time	10 minutes, configurable
environment	audience, weather, stadium, type of grass	no environment factors that influence the game

4.4 System Model

The game that will be played is described in the previous section. The system model that is used in the simulator is composed of three layers, each with corresponding tasks. Figure 4.1 illustrates these. An abstract explanation of each of the layers and the links between them is given in the following sections. The links between the layers are implemented using channels in Fleeble⁹. A detailed description of the actual inputs and outputs making up the player capabilities is given in Section 4.5. A key element of the model is that communication between different robots (and corresponding players) can only occur through the framework.

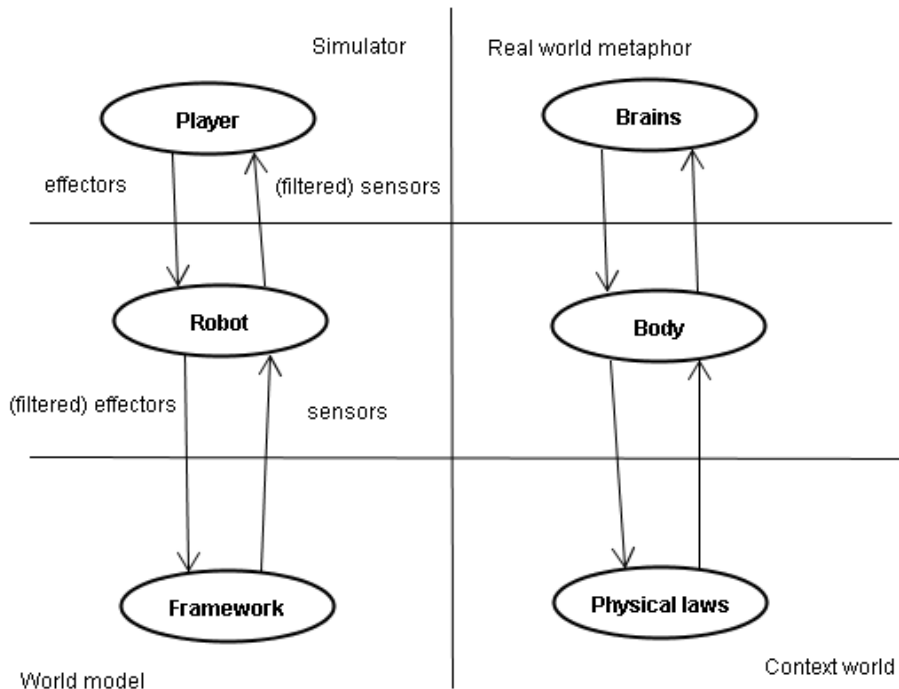


Figure 4.1: System Model: the Layered Approach

4.4.1 Player Layer

The player layer and the corresponding links are displayed in Table 4.5.

⁹This is still the case in a distributed approach where multiple computers are involved in the simulation, as namespace mirroring (See Section 3.4.10) ensures a transparent communication over channels between the various hosts.

Table 4.5: Description of the Player Layer

Layer	Link name	Description
Player	-	Defines the behavior of the robot. All reasoning is done in this layer.
Player	(filtered) sensors	The inputs from the robot layer. This is by default everything that the robot senses, but a filter can be added to simulate technical imperfections at the robot.
Player	effectors	The desired outputs for the robot.

4.4.2 Robot Layer

Table 4.6 describes the function of the robot layer and its corresponding links in the system model.

Table 4.6: Description of the Robot Layer

Layer	Link name	Description
Robot	-	The robot is physically present on the field. It will attempt to act in accordance with the commands given by the player layer.
Robot	sensors	Everything that a robot can observe from its environment. Within the sensor's range, perfect perception by the robot is guaranteed. Uncertainties may be added to simulate imperfect technology. These sensory inputs are then sent to the player layer.
Robot	(filtered) effectors	A player may desire certain actions to be taken, that are technically not possible. As a result of a defect, obstacle or some other obstruction, the requested action can not be executed and is filtered out. The commands that <i>can</i> be carried out are transmitted to the framework layer.

4.4.3 Framework Layer

In Table 4.7, the function of the framework layer and its links is illustrated.

Table 4.7: Description of the Framework Layer

Layer	Link name	Description
Framework	-	The rules of the game are controlled by the framework layer. Rules governing movement, vision, communication and the rules of the game are defined and secured by this layer. The current state of all robots is kept and updated by this layer. The rules of the game and the framework parameters are publicly known by all layers.
Framework	(filtered) effectors	The desired action to be taken by individual robots has to be tested against the physical possibilities that are defined in this layer. When an action is possible, the state of the system is updated.
Framework	sensors	The state of the system resulting from processing all individual robot's (filtered) effectors is returned to the robots through this link. Within the physical limitations defined by the framework layer, perfect information about the current state is given to the robot layer.

4.5 Player Capabilities Model

This section describes the player capabilities model. First, the movement model is described. Basic player movement, ball movement, collision handling and movement speed are explained. Then, the visual and aural models are explained. Finally, the change behavior mechanism is described.

4.5.1 Movement

Robot movement is caused by the player layer requesting a certain effector to be used by the robot layer. The robot layer will check whether it is possible to perform the desired action. The framework layer is then responsible for checking the validity of the movement¹⁰, updating the state of the game with the new position, and periodically returning this through the sensory link to the robot layer. The robot layer will then return the perception to the player layer. Basic player movement, ball movement, collision handling and movement speed are discussed in the following sections. An overview summarizing the movement model concludes this section.

4.5.1.1 Basic Player Movement

A robot is a circular solid object with a radius of 1.85 meters (See Table 4.4). A robot can move either forward, backward, left, or right. The orientation of a robot can be changed by turning either left or right. A robot is able to kick a ball if it is directly in front of him. A robot can not leave the field, as the framework will prevent it from doing so.

4.5.1.2 Ball Movement

The ball is also a circular object, but unlike robots, it is not 'controlled' by any higher layer. A ball acquires movement through collision with robots, or through being kicked by a robot (a special type of collision). The ball will move in a straight line, in the direction of the collision or kick. The velocity of the ball decreases with time, as a result of simulated friction with the field. The ball comes to a halt naturally. Ball movement is entirely predictable, except when the ball is kicked. A small random direction element is added when a ball is being kicked.

4.5.1.3 Collisions

Collision detection in the model is rather simple, due to the circular nature of all objects. The distance between two objects is determined, if this is below a certain threshold, a collision occurs.

As there are no bookings, fouls or injuries, there are no individual punishments to either of the two parties of a collision, but rather an equal penalty for both. The maximum speed of both robots is reduced for a fixed amount of time.

¹⁰It is, for instance, not allowed to walk off the playing field

Every time the state of the game is updated, the 'new' positions of all robots is calculated prior to collision detection. For all new collisions, robots are split up to the minimum distance required not to collide anymore. The rotations of the robots are unaffected. The distance that the robots have to split up is divided equally among the robots. That is, when a robot is moving at 20 pixels / update towards an other robot that he is standing next to, and is standing still, the following will happen: The first robot will first be positioned 20 pixels overlapping the second robot. This will be detected by the collision detection, and both robots will move 10 pixels away from each other¹¹. The result is that both robots will have moved 10 pixels across the field. As a natural consequence of this, a robot that is 'pushing' another robot, will be able to push this other robot out of the way, unless equal counter force is applied¹², or if the other robot uses a 'block'. A block is a special move that will severely reduce the effects of a collision on the blocking robot.

Collisions with the ball can have several effects. If the ball comes in at an angle within the predefined catchable angle, the robot *catches* the ball, and the ball will automatically move with the robot. Furthermore, the ball automatically rotates towards the direction the robot is facing, such that a robot that is carrying the ball for a while will always have it directly in front.

If the incoming angle of the ball is outside the catchable area, and the ball is moving at a high speed, it bounces off the robot. This can be prevented by using the special move 'block'. This effectively catches any ball close enough to the robot. If the ball is not moving at high speed, the robot will catch the ball and it will automatically rotate to the front.

4.5.1.4 Movement Speed

Robots have exactly the same baseline physical capabilities. As a result the basic movement speed is fixed. There is no acceleration or deceleration model implemented; robots reach their maximum speed instantly.

Robot speed can be affected by several penalties, however. These penalties can be related to the movement, recent collisions or recent shouting (See Section 4.5.3).

¹¹This is done by taking the line between the center points of both robots, and moving both 10 pixels further on the line

¹²Remember from Table 4.4 that robots do not have individual skills, and are equally strong

Forward movement is the fastest. Walking backwards, and sidestepping left or right are penalized for the duration of the movement in that direction. Not turning is the fastest, turning left or right are penalized for the duration of the turn. While kicking, the robot also receives a speed penalty. The exact parameters are defined in the parameters, and are globally known.

4.5.1.5 Overview

Table 4.8 provides a brief overview of the movement model described in the previous sections.

Table 4.8: Overview of the Movement Model

Property	Description
Movement direction	Forward, backward, left, or right.
Turn	Left or right.
Special moves	Kick or block.
Speed	Instantly maximum speed, possible penalties.
Speed penalties	Awarded for direction, turn, special, collision or shouting.
Ball speed	reduces over time, increased through interactions with robots.
Ball catching	Automatic when incoming angle is within catchable angle, or when the ball speed is low or a block is performed. The ball will bounce off when outside catchable angle at high velocity. Ball auto-rotates towards front of robot.

4.5.2 Visual Model

Vision is the primary sensor of the robot. Although not the only one, intelligent teams can be developed reasoning solely about the visual perception. Robots have a field of vision that is equal in size for each robot, and defined by a given view angle and view range. All objects¹³ within this field of vision will be recognized. For each of the objects that are recognized, the name of this object, the team it is playing for¹⁴, and the exact position and

¹³Objects refer to either the ball or other robots)

¹⁴If the object is a robot

heading are perceived without any errors. Robots will also receive their own position. A brief overview of the visual model is given in Table 4.9.

Table 4.9: Overview of the Visual Model

Property	Description
Field of vision	Everything within predefined view range and view angle.
Errors in perception?	None.
What is perceived?	For each object (ball or robot) in field of vision; name, team, exact position and heading (and velocity for the ball).
Is own position known?	Yes.

4.5.3 Aural Model

To reach high-level strategic behavior by robots, communication is essential. The aural model describes how communication between robots is possible.

Robots have a certain shout range. This is a fixed, predefined distance, and objects within this range in the circular area around the robot will receive any messages that the robot decides to send. Any type and amount of data can be sent at any time. The only restriction to communication is that a (severe) speed penalty lasting several seconds is inflicted on the sender of a message. Both friendly and opponent robots within the shout range will receive the message. Apart from the content, the name of the sender and the time at which the message was sent are also included. An overview of the aural model is given in Table 4.10.

4.5.4 Changing Behavior

Since the physical capabilities of robots are exactly identical, substituting them would be a rather useless activity. To allow for adaptive team behavior, an option to substitute the player on a robot is included. Substituting the player¹⁵ on a robot effectively means replacing the player that is currently loaded on the robot by another player. This can be done at any time during the game, and as often as desired. The knowledge base of the existing player can be stored using properties (See Section 3.4.9), and as such the newly substituted player will be able to access data collected by the old

¹⁵Remember from Section 4.3.2 that each robot has its own player agent that determines its behavior. Different robots may have different behaviors.

Table 4.10: Overview of the Aural Model

Property	Description
Receive messages	All objects within predefined shout range that are in the circular area around the sender. This includes friendly robots, but also opponents.
Errors / noise in reception?	No errors.
What can be sent?	Any type or amount of data.
What is received?	Content of message, name of sender, time at which message was sent.
When can messages be sent?	At any time during game.
Restrictions to communication	A fixed-time speed penalty.

player. This feature allows for the adaptive team behavior that is described in Part III.

Each robot can have its own *behavior* (player agent). As such, the success of a team depends solely on the performance of these player agents. Substituting the behavior on a robot effectively means that the player agent is killed, and the new behavior is installed on the robot. This process can only be initiated by the player agent originally residing on the robot agent, and can be executed at any time during the game. This feature allows for the (autonomous¹⁶) dynamic changing of the strategy.

As the game is dealing with software agents, the change behavior functionality is by no means necessary. Keeping a global state in the player agent (i.e. OFFENSIVE or DEFENSIVE) that determines what behavior the player currently exhibits would result in similar results. When the agent reasons that it is best to change to a different behavior, it will toggle this status. Such an approach is undesirable however, as it would cause very long, poorly structured agents. Also, the semantic value of the 'player' as representing the 'brains' is lost, since the new state is likely to reason about inputs and outputs in a completely different manner.

Changing the behavior can only be initiated by the player presently residing on the robot. As there is no central authority present, the decision to change behavior has to be made - after (possible) collaboration with fellow

¹⁶There is no outside interference, the decision to change the behavior is made by the individual agents

team mates - by the player. An overview of the change behavior functionality is provided in Table 4.11.

Table 4.11: Overview of the Change Behavior Mechanism

Property	Description
What does it do?	Change player (behavior) presently residing on the robot.
When can it be used?	At any time.
How often?	No restrictions.
Knowledge of existing player	Can be transferred to new player through properties.
Who can change behavior?	The player presently residing on the robot.
Restrictions / penalties?	None.

4.6 Justification

In this section, various decisions and trade-offs that were made in developing the model are discussed. Alternative solutions are described and compared.

4.6.1 Speed Penalties

A common problem in simulations is how to penalize certain types of behavior such that intelligent solutions - that succeed at accomplishing the goal - will prevail. Working with artificial, arbitrary point systems that award agents for accomplishing some goal, and penalizes them for committing fouls, or for, for instance, communicating.

For such an approach to work, a good performance measure is required. The only real performance measure in soccer is the amount of goals that were scored, compared to the amount of goals the opponent scored. Other measures could be suggested, such as ball possession, number of shots or the percentage of time the ball was on the opponent's half. The problem with defining such artificial performance measures is that they do not capture the actual performance of a team. A team with 90% ball possession can still lose with 99-0. Using the aforementioned performance measures, the latter team would have 'won'.

The problem with artificial performance measures and point distributions has caused the two predecessors of the soccer assignment described in Part IV to fail at accomplishing their goals.

Rather than using an arbitrary artificial point system, the only performance measure for the soccer teams is the final score of a match. Robots are penalized through speed penalties for conducting behavior that causes speed penalties in human players. Examples of such behavior are shouting, collisions, kicking, turning, and blocking.

4.6.1.1 Shouting

Sprinting and shouting at the same time is rather difficult for humans. Running at top speed and still shouting at the same pace as one would while standing is impossible. Since robots that are moving will always move at their top speed, and restricting the amount of messages or the maximum size of messages is not an option (See Section 4.6.3), robots receive a speed penalty. Starting at the time of their most recent shout, the robot will have a lowered maximum speed for a predefined amount of time. As the speed penalty is reasonably big, this forces the choice between communication and autonomous reasoning. The time that the penalty lasts represents the time required to shout any message. The framework is indifferent about the amount of information that was actually sent, stimulating users not to worry about the size of the messages, but rather about when to send them.

4.6.1.2 Collisions

Dealing with collisions in a model without a referee, fouls, injuries or bookings is difficult. However, the model also features identical physical capabilities for robots. Because of this, it is assumed that not a single party can be blamed for the collision, and both parties are punished by a speed penalty - to resemble the time or difficulty real players or robots would require to resettle themselves.

Alternative models could distinguish between victims and aggressors, and punish the aggressors. Using such a model would effectively lock the game, as a strategy with 7 robots in front of the goal would prevent the opponents from ever scoring. As a trap door mechanism to prevent aggressors from pushing opponents around, their speed - and as such their impact on the victim - is severely reduced, and victims have the option of using a block, a special move that will reduce the impact of collisions.

The approach taken to deal with collision makes collisions, like humans tackling each other, a possible way of halting or slowing down an attack. It is a valid approach to stop robots from advancing over the field too quickly.

4.6.2 Movement

The model features a low-level movement control. Movement could be pre-implemented by the robot layer, but this would contradict the model that was described in Figure 4.1. The model defines a number of sensors and effectors of the robot layer, and passes these on to the player layer, that functions as the brain. Implementing a high-level movement model that automatically moves a robot to a position, or to track a player or ball, should naturally be done in the player layer, as it includes *reasoning* about the environment. Such movement methods would deal with issues such as whether to avoid collisions (at all cost), move in a straight line or over a certain flank. These are all high-level decisions that determine the behavior of the player, and should be nested in the player layer.

4.6.3 Communication Restrictions

Unlike the domain of chess, a soccer environment is dynamic. The environment is also constantly changing and decentralized. Also, the information accessibility is incomplete. Allowing unrestricted unlimited communication would face three problems:

1. Unrestricted unlimited communication would be cheating on the problem domain. Rather than developing AI methods to deal with the incomplete, dynamic, decentralized, real-time properties of the environment, allowing unrestricted unlimited communication would allow for brute-force approaches using centralized supercomputers to determine optimal movement.
2. Robots often face limited processing power and network bandwidth. Although these are increasing rapidly, certain overhead is always present when processing and sending messages. Communication always has a certain price.
3. The soccer challenge serves to develop AI methods that deal with incomplete, dynamic, distributed, real-time problem environments. Due to the physical nature of certain environments, intense communication may not be feasible.

A team solely based on communication would be like the system described in the first problem. A team solely based on autonomous reasoning would not be able to establish the complex team play that can be acquired through a combination of both. For these reasons, users should develop a team that carefully considers this trade-off, and decides when and what it shall communicate. A measure frequently taken to prevent players from communicating all the time with large chunks of data is to put a limit on the amount of

data that can be sent, and to limit the maximum amount of messages that can be sent. This approach is *fundamentally flawed* for two reasons:

- Limiting the amount of data that can be sent says nothing about the amount of information content. Shannon [65] defined the amount of information of an event to be the expected value of its outcome's surprisal. To translate this in soccer terms; If there is a fixed number of messages that could be transferred, then the amount of information in receiving the message *look out behind you* equals the expected value of the surprisal¹⁷ of the outcome. As there is a finite amount of possible messages¹⁸, the resulting message size will be very small. The maximum message size in the Soccer Server simulator, 512 bytes [66], is more than enough to be able to send everything you could possibly want, as it allows for $2^{(8*512)}$ unique messages. Applying maximum compression to communication would as such result in a large benefit for teams doing so, as their efficient bandwidth usage results in practically unlimited communication possibilities. It would also mean a large disadvantage for teams not investing time in compressing their communications. As the focus of the game should *not* be on optimizing tiny details such as the compression of the communication, this is an undesirable disadvantage. It is however not realistic that humans would ever speak like this, and neither is the artificial bandwidth limitations. Sending a larger amount of data will only slow down the recipient that attempts to parse it. Perhaps - when sending movies over the channels - the whole network will slow down. Since the simulator will be 2D and not deal with any real camera input, and it will not depend on wireless networking limitations, this is unlikely to happen.
- Limiting the number of messages that can be sent could bear some relation to reality; for humans, it is not possible to shout complex instructions taking five seconds to shout and, half a second after starting this, shout instructions to someone else. Applying some transformation to account for the amount of data and the amount of time a human would take for shouting this is not feasible. Such a transformation would not be able to distinguish between the team that made their communication as efficient as possible, and the team that uses a quick inefficient communication model, while such a distinction would be required to adequately estimate the duration of the *communication*.

¹⁷Surprisal represents the surprise of seeing the particular outcome. If an event will happen once in a million times, the surprisal of this event is about 20 bits - the amount of bits required for representing this probability

¹⁸If not for every message, then at least for all individual parts of a message such as position information on the field, and there is only a small number of these individual parts, so the concatenation of these will not be big

Since this can not be done in a fair way, setting an arbitrary limit on the number of messages that can be sent is of course an option. It is however rather arbitrary and finding a real world metaphor to explain the restriction other than *human players can not shout an infinite number of messages either* is difficult. Technically, such a limitation would also be very arbitrary, as the (wireless) networking capacity is rapidly increasing. Restrictions to the amount and frequency of messages would at best resemble the present state of the art in wireless networking, whereas this is likely to advance in the foreseeable future.

The approach taken by the model stimulates users to consider the trade-off between communication and autonomous reasoning. It can easily be explained using human players as a metaphor¹⁹. The speed penalty puts the robot at a disadvantage. This disadvantage will be very big when the robot is carrying the ball and moving towards the opponent's goal, but relatively small if a robot is covered by several opponents and effectively out of play.

4.6.4 Central authority

Unlike most other simulators and approaches, the simulator does not feature any central authority. The main reason for not doing so is that adding a coach would change the focus to effectively controlling the coach rather than striving for teamwork and cooperation based on the limited information that is available.

The restriction of working without a central authority introduces an interesting problem to users. For example when changing behavior, if a team has 3 offensive players, and they would reason autonomously that when they lose by 15 points, one offensive player has to switch to a defensive player, the result would be that all offensive players would be defensive players. This undesired result can be countered through programming a certain order in which players will need to change - at the cost of being predictable, or by giving a certain player *team captain* responsibilities and letting him decide who should switch. Dealing with this is an interesting dimension, as most realistic artificial intelligence problems do not have a central authority either.

4.7 Customizing the Model

Flexibility was previously mentioned (See section 4.2.1) as an important requirement for dealing with the different types of users. The system is cus-

¹⁹Human players can not shout and run at top speed simultaneously

tomizable in both its behavior and in its appearance. This section describes the customizations that can be made to the model

4.7.1 Changing Framework Parameters

The parameters determining all physical laws that the framework enforces are defined in an XML file. A description of the meaning and functioning of the specific parameters is given in Appendix B.

4.7.2 Changing Visual Appearance

The visual appearance of the game is also easily modified. Custom icons for displaying the robots can easily be defined and integrated in teams. Users can define their own team name, and name their players as well.

4.7.3 Advanced Customization

The work described in this thesis is customized for the specific purpose of reaching the goals stated in Section 1.4. An intrinsic element of achieving these goals is the highly simplified world model that was described in this chapter. Professionals working with the system, this simplicity requirement might be counterproductive. Fortunately for these users, the layered approach as described in Table 4.1 is perfectly suitable for highly specific customizations.

If professional users would want to simulate realistic robots - and the corresponding uncertainties regarding inputs and outputs - this can easily be added to the system by applying a filter to the *Robot* layer that will add a certain type of noise or modification over the data. The *Player* layer's implementation will not be affected by this, as the link between the layers is not affected. Similarly, if the nature of the aural, visual or movement models would be edited in the *Framework* layer, neither the *Robot* implementation or the *Player* implementation would be affected. *This feature makes the framework robust and extensible.*

Furthermore, this feature makes the simulator suitable for testing different technological states and their impact on cooperation and team work. Given an implementation of the current state of technology, and an implementation where the uncertainty about position information would be reduced by, for example, 50%, the results could be compared and reasonable predictions towards the effect of the technological progress on the team's performance could be made.

Chapter 5

Design

Do not squeeze your inspiration and your imagination; do not become the slave of your model, Vincent van Gogh

This chapter presents the design of the soccer simulator. It begins by explaining the approach that was used for designing the system. The next section describes the design of the use cases. Subsequently, the global system design is discussed, followed by a detailed overview of the design of the main classes that will be used. Next, a detailed description of the movement model and corresponding algorithms is given using pseudo-code. Finally, the user interfaces are discussed.

5.1 Approach

An expert is a man who has made all the mistakes which can be made in a very narrow field, Niels Bohr

The actual design of the system consumed nearly as much time as its implementation. Although the decisions regarding all aforementioned aspects of this simulator were made in a relatively small period of time, the step that converted the conceptual simulator to a working simulator that works fast and according to specification was rather difficult.

Although the author had a little knowledge on Aibo soccer and the Soccer Server, these were excluded from the design process on purpose. The goal of the system is to solve the problem laid out in Section 1.4, not to clone or adapt an existing soccer simulator for use in some educational environment.

It is noteworthy that the resulting implementation is quite different from the previous simulator. A number of problems are however still solved in a similar manner as in the previous design.

5.2 Use Case Design

Three different groups of users were identified in Section 4.2.1; students, hobbyists and professionals. The primary requirements for these three types

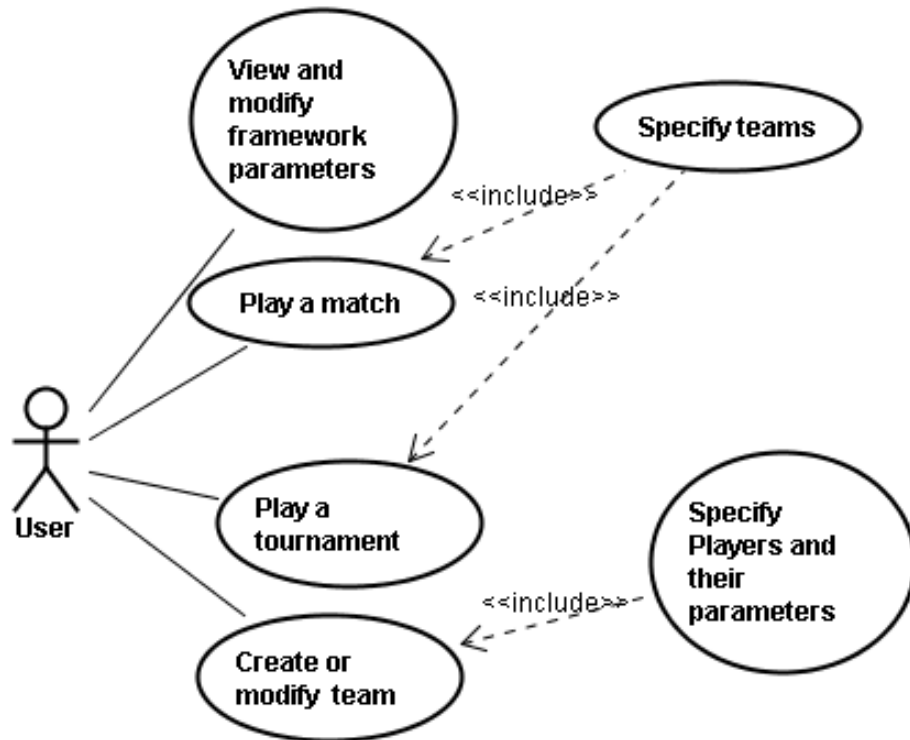


Figure 5.1: Use Case Diagram for User of Simulator

of users are taken into account in the design, however the way they will use the system is essentially the same. Figure 5.1 illustrates this. A user can play a game of soccer, start a tournament, create or modify a team, or edit the framework parameters. Prior to starting a game or tournament, the user has to specify the teams that should be included in the match(es). To create a team, the players and their parameters have to be specified. These parameters include name, icon, position on the field, and team name.

5.3 System Design

This section begins by describing the global design of the system, then all individual elements are discussed.

5.3.1 Global Design

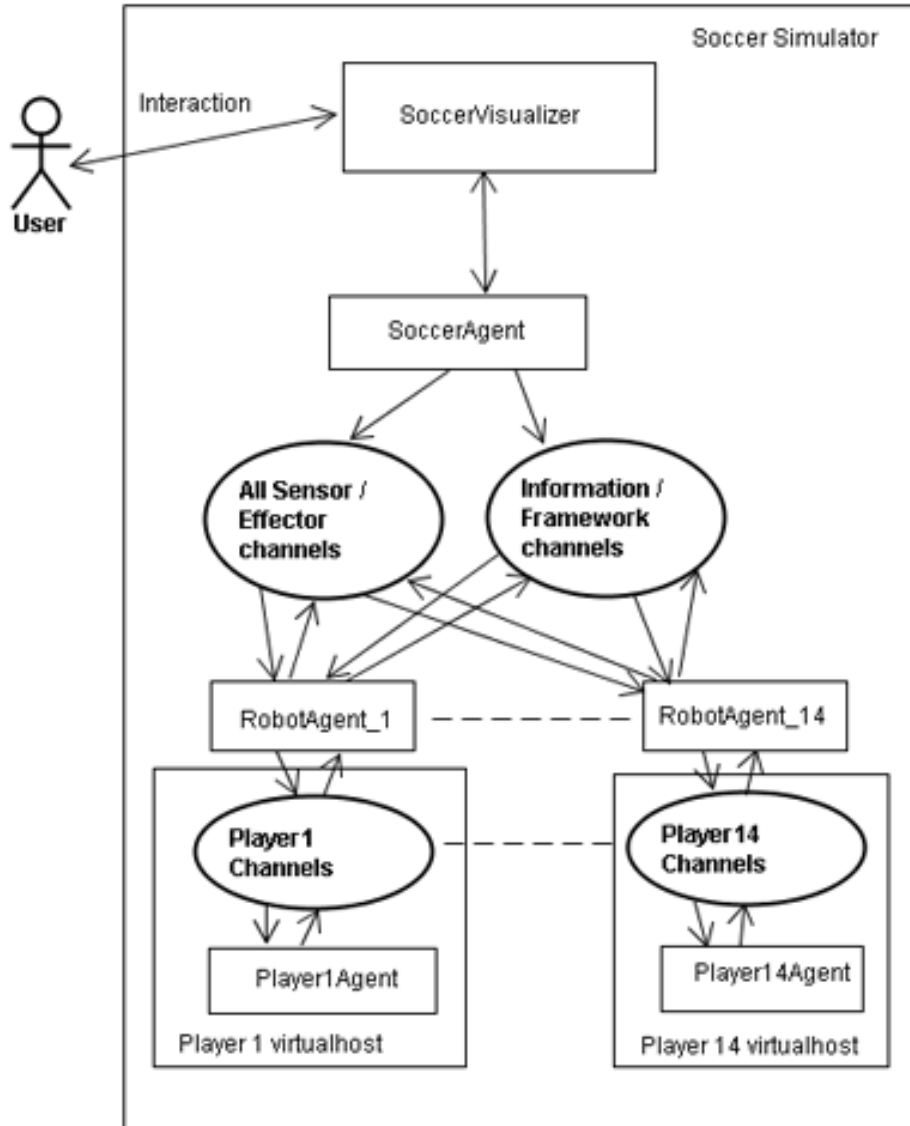


Figure 5.2: System Overview

The Simple Soccer Simulator has a single user interface, which is entirely controlled by the SoccerVisualizer. A central role is played by the SoccerAgent, that implements the entire framework layer (See Table 4.1). It enforces the game rules, and will act as an arbiter between the different RobotAgents. The SoccerAgent is also responsible for maintaining all position and status information for all the different robots. The SoccerVisualizer

will update the information from the SoccerAgent at regular intervals and display them. The RobotAgents represent the human body according to Table 4.1, and will pass on requests from their corresponding PlayerAgents to the SoccerAgent. As the game is played with 7 vs 7 players by default¹, there are 14 RobotAgents. Each RobotAgent is implemented as a ChildAgent of the SoccerAgent. Information about the game settings and status is communicated over a number of Information / Framework channels. Robots can control or read out their sensors and effectors through a number of other channels. To prevent communication between different robots from occurring outside the framework, all RobotAgents will load their corresponding PlayerAgent and give them a virtualhost (See Section 3) to lock them in their namespace. Communication within this virtualhost occurs through several channels that will be explained in more detail in the next sections.

5.3.2 Framework Channels

The framework channels facilitate communication between the SoccerAgent and the RobotAgent. This communication can be categorized in 3 groups:

- Debugging information
- Updating system-wide variables
- Initializing the system

An overview of the channels can be seen in Figure 5.3, a detailed explanation of the function of each is given in Table 5.1. Section 5.4.3 addresses the 1 and 2 in the figure.

5.3.3 Robot Channels

The robot channels function as input and output channels for the robot. They are positioned between the SoccerAgent and the RobotAgent. The RobotAgent will send its desired actions over the corresponding output channels. These are subsequently processed and validated by the SoccerAgent. The resulting sensory inputs are then returned to corresponding RobotAgents. An overview of the channels is given in Figure 5.4, and a description of each of the channels is given in Table 5.2

5.3.4 Player Channels

The player channels are very similar to the robot channels because the robot layer functions mainly as a transparent layer between the framework and the player. These channels are the only means of interacting with the

¹The amount of players is one of the game properties and can be easily changed. 7 is used as a default, but it can be changed to user preferences at any time.

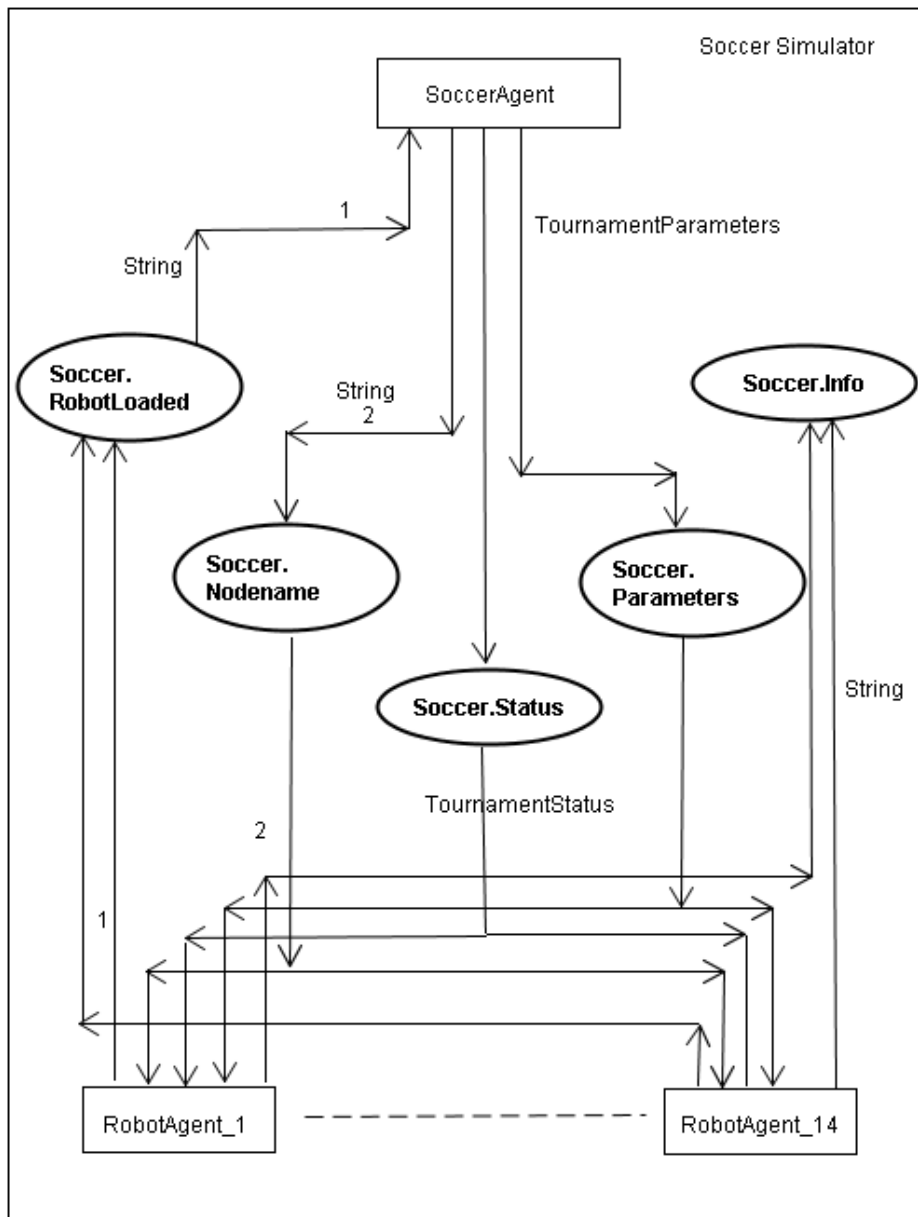


Figure 5.3: Overview of the Framework Channels

system for PlayerAgents, as all PlayerAgents are locked in their virtual-hosts (See section 3). Figure 5.5 provides an overview of the channels. The Information channels mentioned in this figure are *Robot.Init*, *Robot.Status* and *Robot.Parameters*. Because the virtualhost is prefixed to these channel names, the actual channel names as they appear in Fleeble will be extended,

Table 5.1: Description of the framework channels (I = Initializing, U = Updating, D = Debugging, S = SoccerAgent, R = RobotAgent)

Channel	Cat.	From -> To	Description
RobotLoaded	I	R -> S	See section 5.4.3.
Nodename	I	S -> R	See section 5.4.3.
Status	U	S -> R	Whenever the Status changes (i.e. change to <i>paused</i> mode, or if a team has scored), this is passed on to the RobotAgents. Section 5.4.2 describes this in more detail
Parameters	I	S -> R	When all robots have been loaded, the parameters describing the game rules are passed on to the RobotAgents. See Appendix B to learn more about these parameters.
Info	D	R	Used as a debugging channel for the RobotAgent, for printing high level error messages

in the format of: *virtualhost.#_your_robot_name.Robot.Channel_Name*. Table 5.3 gives a description of the functions of the channels. As mentioned in section 4.7.3, the robot layer could easily be extended to simulate actual robots with realistic uncertainties regarding inputs and outputs and reasonable physical limitations.

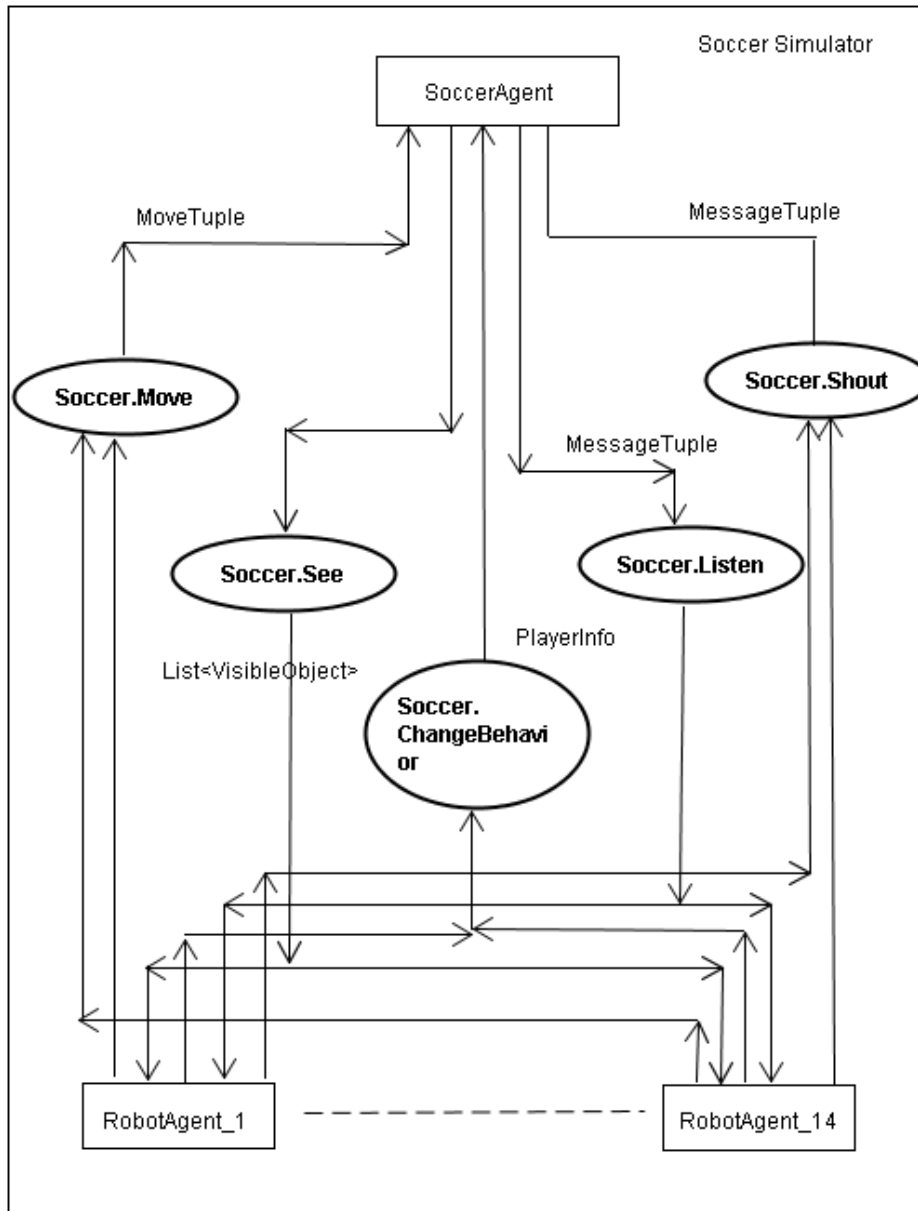


Figure 5.4: Overview of the Robot Channels

Table 5.2: Description of the robot channels (S = Soccer Agent, R = Robot Agent)

Channel	From -> To	Description
Move	R -> S	Robots can send their desired movement over this channel. Movement includes the direction of the movement, a possible turn, and possibly special moves such as kicking. See section 5.5 for a detailed description.
See	S -> R	Main input for the robot. It will receive perfect information of all objects within its field of vision on regular intervals. See section 5.4.5 for more details.
ChangeBehavior	R -> S	Used to change the PlayerAgent residing on this RobotAgent. Robot-specific properties can be changed, such as the robot's base position, icon, but the main function is to change the behavior (PlayerAgent).
Listen	S -> R	All data that was sent by other players within hearing range is received on this channel. The data can be in any format. Section 5.4.6 further elaborates on this.
Shout	R -> S	Data that this robot would like to <i>Shout</i> , share with nearby other players. Shouts will be received by all enemy and friendly players within a predefined shouting range. There is no limit on the amount of data that is transmitted, but a speed penalty is applied for shouting. Section 5.4.6 describes the details of this.

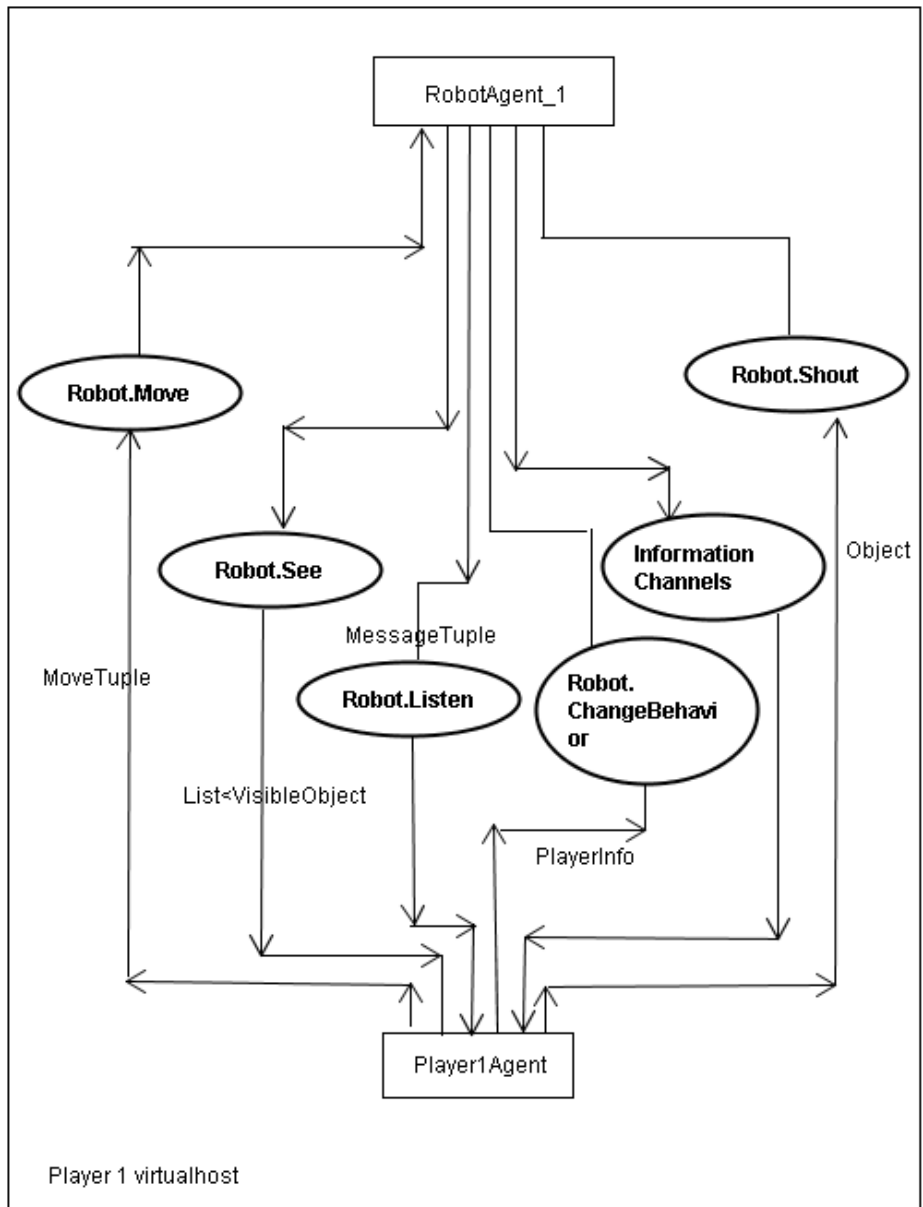


Figure 5.5: Overview of the Player Channels

Table 5.3: Description of the player channels (P = PlayerAgent, R = RobotAgent)

Channel	From -> To	Description
Move	P -> R	Same as corresponding channel in robot agent.
See	R -> P	Same as corresponding channel in robot agent.
Listen	R -> P	Same as corresponding channel in robot agent.
ChangeBehavior	P -> R	Same as corresponding channel in robot agent.
Shout	P -> R	Same as corresponding channel in robot agent.
Listen	R -> P	Same as corresponding channel in robot agent.
Init	R -> P	In the team creation process (Section 5.4.7), robots are given names, a player agent, a base position and an icon. These are distributed by the SoccerAgent upon initialization and passed on by the RobotAgent when the PlayerAgent has loaded. Knowledge of your robot's name can be useful in several situations.
Status	R -> P	The status of the match. This includes details about the status of the game, but also of the score. Section 5.4.2 expands on this.
Parameters	R -> P	Contains the game parameters that can be used directly in certain (optimization) algorithms inside the implementation. Appendix B will provide an overview of the actual parameters.

5.4 Classes

5.4.1 Overview

The system consist of 4 main elements, and each element consists of one Java class. Outside these primary classes there is are number of helper and wrapper classes. The 4 main elements and their corresponding classes are:

1. GUI - SoccerVisualizer
2. Framework - SoccerAgent
3. Robot - RobotAgent
4. Player - PlayerAgent²

The purpose of each of these classes has previously been discussed. The rules of the game are embedded in the *TournamentParameters* Object and distributed among all agents. A detailed overview of the specific parameters that make up the TournamentParameters object is given in Appendix B. This section will further specify the functionality of the 4 primary classes. It will mention and discuss all classes that are used.

5.4.2 Tournament Status

TournamentStatus is an Object that keeps track of the number of goals, the time at which these were scored and the *State* of the game. The permissable states are:

It is updated with each change of state³. Updating it results in the SoccerAgent sending a message to the RobotAgents, which in term further propagate the message to their corresponding PlayerAgents.

5.4.3 Initializing the Framework

Prior to simulating any games, the user has to specify the *teams*. This is done through the user interface. The user interface⁴ is discussed in more detail in section 5.6. For the remainder of this section it can be assumed that this is a layer that provides the preferences and settings required to start a game.

²Users will have to implement this, or use the readily available reference implementations

³note that this automatically implies that it is updated after a goal is made, as this will cause the game to switch from GAME to SETUP

⁴The most important part of the user interface is the SoccerVisualizer, although the TournamentGenerator and MatchSetupGenerator are responsible for displaying and handling the creating of tournaments and teams, respectively.

Table 5.4: The Different States of a Tournament

State	Description
INIT	Start state. The teams have to be defined or created. Nothing else is displayed on the screen.
SETUP	The <i>PlayerAgents</i> can not exert any control over their robots in this state. The <i>RobotAgents</i> will walk towards their <i>base positions</i> automatically. At the end of a <i>SETUP</i> state the simulator will change to <i>GAME</i> state. The simulator comes in <i>SETUP</i> state whenever the game begins, or when a goal was made.
GAME	The state where the <i>PlayerAgent</i> is in full control over the <i>RobotAgent</i> . During the <i>GAME</i> status, the tournament clock in the upper-left corner of the screen will be updated. This is not the case in any of the other states.
PAUSED	During this state the game freezes and the time stops.
END	This state occurs when the game has ended. The <i>PlayerAgents</i> have no more control over their robots and the <i>RobotAgents</i> walk off the field. The simulator will change to the <i>INIT</i> state when this is complete.

The preferences include the team names, all individual robot names and their base positions. Since the virtualhost name that is given to a *PlayerAgent* depends on their name, the preferences must be known in advance. Once they are known, the following mechanism will start:

1. SoccerAgent: Load 2 x num-players RobotAgents.
2. Each RobotAgent: upon being loaded, send a "loaded" message over channel Soccer.Robotloaded"⁵.
3. SoccerAgent: Store for each received "loaded" message the MessageSource object that can be used to uniquely identify the sender of this object. Link this MessageSource object to a robotname⁶. When all "loaded" messages have been received, send the TournamentParameters over the Soccer.Parameters channel to all robots.
4. SoccerAgent: Reply⁷ to each individual robot over channel *Soccer*

⁵This corresponds to the '1' in Figure 5.3

⁶The robot names are prefixed with a number, as described in Appendix B under player-startnumber

⁷Replying is explained in Section 3.4.8.1, and is characterized by the message only being received by one predefined agent rather than by all subscribed agents.

Nodename with their robot name⁸. After sending the nodename, the SoccerAgent will send a PlayerInfo (See section 5.4.7) object to each RobotAgent with the robot-specific information.

5. RobotAgent: Upon receiving their nodename, they will know what virtualhost they will need, and subscribe and publish to the corresponding channels. After receiving the PlayerInfo object, the RobotAgent will know what PlayerAgent to load, and load this in the CREATED agent state. The RobotAgent will receive a notification of the PlayerAgent being loaded in the (CHILD_LOADED) method, and apply the correct virtualhost. Then the state of the PlayerAgent will be changed to RUNNING.
6. RobotAgent: Send the PlayerInfo to the PlayerAgent over channel Robot.Init.
7. RobotAgent: Send the TournamentParameters and TournamentStatus to the PlayerAgent over Robot.Parameters and Robot.Status, respectively.

The system started in SETUP state, and the RobotAgents will start moving the robots towards their base positions. Once these positions have been reached, the TournamentStatus will change to GAME and the PlayerAgents will control the robots.

5.4.4 Running the Game

The system design is designed so that the robots will receive updates at the "Robot.See" channel. The information that is sent is originally provided by the SoccerAgent. The SoccerVisualizer is responsible for calling the method in the SoccerAgent that updates the current state, and processes visibility.

The SoccerVisualizer utilizes a private inner class *DisplayThread* that is responsible for updating the GUI and the position information at predefined time intervals. The way this works is described in Listing 5.1:

The tournament time is the time that is displayed on the field, and the time that is used to determine when the game is over. This mechanism effectively ensures that when the simulator is run on a slow computer, the tournament time will update in accordance with the actual amount of updates, rather than with the amount of physical time that has elapsed. This guarantees that for any match, all players will have had equal chances, regardless of the speed of the computer.

Listing 5.2 is a pseudo-code implementation of the redraw method that is called by the tournament thread.

⁸This corresponds to the '2' in Figure 5.3

Listing 5.1: Running the Game

```

1 do always {
2   if in GAME status {
3     redraw()
4     if ( time.now - time.before_redraw < 1 /
5         refreshrate )
6         wait until 1 / refreshrate time has passed
7         since
8         time.before_redraw
9
10    add 1 / refreshrate to the tournament time
11
12    // this thread also handles the MatchSetupGenerator
13    // for the team
14    // creation interface, and the TournamentGenerator
15    // when the game status is INIT, and all other
16    // possible
17    // Tournament States.
18  }
19 }

```

Listing 5.2: The redraw() method

```

1 redraw()
2   new_positions = socceragent.processMoves()
3   if in GAME status and time.now - time.
4     lastVisibilityUpdate
5     > visibility_update {
6     socceragent.processVisibility()
7   }
8   draw all objects
9 }

```

The new robot positions are calculated in the SoccerAgent using the processMoves function, and are used when displaying all objects on the field. When the simulator is in GAME status, and at least *visibility_update* milliseconds have elapsed since the last visibility update, the visibility is processed and all robots will receive a list of VisibleObjects.

5.4.5 Vision

The robot will automatically detect visual information from the field, and receive this on channel `Robot.See`. This information arrives as a list of `VisibleObjects`, a wrapper class representing `VisibleObjects`. Table 5.5 describes this class, and its two subclasses.

Table 5.5: An Overview of `VisibleObject`, `RobotObject` and `BallObject`

Class	Parameter	Description
<code>VisibleObject</code>	Orientation	This object's orientation, composed of rotation and position.
<code>VisibleObject</code>	Name	The name of this object.
<code>VisibleObject</code>	Radius	The radius of this object; either player-radius or ball-radius in current implementation. Possible extensions with different types of robots would make this field more interesting.
<code>RobotObject</code>	Teamnumber	A number, either 1 or 2, that represents whether this robot's team is playing on the left or on the right. Particularly useful for determining team coordinates using the Orientation class.
<code>RobotObject</code>	Teamname	The name of this robot's team.
<code>BallObject</code>	Velocity	The current speed of this ball. Robot's don't have a velocity as this is always constant and only adjusted by movement or shout-related penalties.

The `TournamentParameters` determine the viewing range and viewing distance, as explained in Appendix B. The pseudo-code in Listing 5.3 illustrates how the visual information is gathered and communicated to the `RobotAgent`:

5.4.6 Shouting and Listening

Communication between the robots is facilitated through the `Robot.Shout` and `Robot.Listen` channels. Any Java object can be transmitted by the `PlayerAgent`. To ensure that the recipient will know who sent the message, and at what time, the `RobotAgent` will wrap a `MessageTuple` object (See Table 5.6) around the transmitted object.

Listing 5.3: The Visual Model

```

1 do every 1/view-update seconds while in GAME status {
2
3   for every robot i
4     for every object j
5       if distance_matrix(i,j) <= view-range
6         add to possibilities (i,j)
7         if j is not the ball
8           add to possibilities (j,i)
9
10    for all possibilities
11      calculate relative orientation (i,j)
12      if relative angle < view-angle / 2
13        add to player i visibility list j orientation (
14          that now also
15          contains j relative to i orientation)
16
17    for every robot i
18      send list of visible objects
19  }

```

Table 5.6: The MessageTuple Object

Parameter	Description
Sender	Name of the sender of the message.
Timestamp	The time at which the message was sent. This is also used by the SoccerAgent to ensure that the shout penalty is given to the sending robot.
Content	A Java Object containing the contents of the message.

The following illustrates the flow of events when a Robot A decides to Shout a message, and Robots B and C are within the shout-range.

1. Robot A: send an arbitrary object O to channel Robot.Shout.
2. RobotAgent(A): receive object O on channel Robot.Shout, wrap a MessageTuple around it, add the current time and the robot's name, and send to SoccerAgent on channel Soccer.Shout.
3. SoccerAgent: receive MessageTuple through Soccer.Shout. Verifies the

rules. Find all robots that are within the shout-range distance from Robot A, and send the MessageTuple to these Robots, over channel Soccer.Listen.

4. RobotAgent(B and C): receive MessageTuple over channel Soccer.Listen, send this over Robot.Listen.
5. Robot B and C: receive MessageTuple on channel Robot.Listen.

5.4.7 Team Creation

A team is nothing more than a combination of num-players⁹ agents with a few extra properties. Teams are stored and loaded from simple XML files. They are read using Mox [67], an easy, straightforward XML parser for Java. Parsing the XML files automatically puts the data in MatchSetup objects, described in Table 5.8. A PlayerInfo object is created to be sent towards the RobotAgents and the PlayerAgents. The PlayerInfo object is described in Table 5.7.

Listing 5.4 is an example XML file for two robots.

The following describes the flow of events starting with users entering the paths to the XML documents of both teams, and ending with the state that section 5.4.3 started in.

1. Both XML files are read and parsed.
2. Two corresponding MatchSetup objects are created.
3. 2 x 7 RobotAgents are loaded.
4. The MatchSetup objects are sent to the SoccerVisualizer to set the team name, and to read the icons.
5. If all goes well without errors, the SoccerVisualizer changes the game state to SETUP.

5.5 Movement

Movement plays a center role in the robot control. To limit the amount of available movement options, and to standardize movement, a MoveTuple object is introduced. Movement is composed out of three elements; (1) direction, (2) turn, (3) special. MoveTuple is a wrapper, combining these three. The MoveTuple is the only object that is allowed (recognized) over the Soccer.Move and Robot.Move channels. As forward movement is generally a lot faster than sideways, or backward movement, *penalties* are included

⁹Default is 7, see appendix B

Table 5.7: The PlayerInfo Object

Parameter	Description
Robot Name	Each robot has his own name. It will be prefixed with a number to guarantee unique names, but the name is displayed on the screen.
Icon	On the user interface, each robot is displayed by a certain icon. The system icons have a radius matching the player-radius, but custom icons are allowed and easily included.
Agent	The desired player, behavior for this particular robot.
Base Position	This robot's starting position (composed of an x and a y value). After a goal has been made, or when the game has just begun, the RobotAgent takes over control and will move to this position. As it is unknown what side the robot will be on, using field coordinates would not be very helpful. Rather than that, team coordinates (See section 5.5.1 for an explanation about the coordinate systems) are used.

Table 5.8: The MatchSetup Object

Parameter	Description
Team name	The name of the team.
List<Robot Name>	(ordered) list of all robot names of this team.
List<Icon>	(ordered) list of all icons.
List<Agent>	(ordered) list of all the agents.
List<Base Position>	(ordered) list of all base positions.

for every type. This penalty is the factor that the robot's velocity will be multiplied with, and it applies only for the time that the move is updated. Table 5.9 provides an overview of the MoveTuple method and the penalties.

The penalties were not included in the TournamentParameters because they are strictly related to movement. Adding all elements to the parameters would give a better overview, but would also unnecessarily complicate the

Listing 5.4: Team XML file

```

1 <?xml version="1.0"?> <team>
2   <name>Holland</name>
3   <robot>
4     <name>Keeper</name>
5     <baseposition>
6       <x>384</x>
7       <y>30</y>
8     </baseposition>
9     <agent>mkt2.uitwerkingen.soccer.KeeperPlayer</
10      agent>
11     <icon>keeper</icon>
12   </robot>
13   <robot>
14     <name>Attacker</name>
15     <baseposition>
16       <x>212</x>
17       <y>140</y>
18     </baseposition>
19     <agent>mkt2.uitwerkingen.soccer.SimplePlayer</
20      agent>
21     <icon>player</icon>
22   </robot>
23 </team>

```

system by making everything dependent on the specific instance of this object, or on the global object if it is declared static.

5.5.1 Coordinates

The game features three different sets of coordinates, as described in Table 5.10. They are illustrated in Figures 5.6, 5.7 and 5.8, respectively. Although the framework uses only field coordinates, easy conversion towards the other coordinate systems is provided. For example, to determine whether an object is within a certain angle from yours¹⁰, the following is already sufficient:

```

1 boolean within_angle = | cos (relative_rotation) | <
   threshold

```

¹⁰This measure could be useful in order to dodge this object to prevent collisions

Table 5.9: Overview of MoveTuple

The <i>MoveTuple</i> object			
Type	Value	Penalty	Description
MOVE	FORWARD	1.0	Basic forward movement.
MOVE	BACKWARD	0.7	Back backward movement.
MOVE	LEFT	0.4	Sidestepping to the left.
MOVE	RIGHT	0.4	Sidestepping to the left.
TURN	LEFT	0.8	Turning by predefined number of degrees to the left.
TURN	RIGHT	0.8	Turning by predefined number of degrees to the right.
TURN	STRAIGHT	1.0	Movement straight ahead, no turn.
SPECIAL	KICK	0.25	The speed penalty for kicking the ball. The penalty will last only one update, so it is not considerable, but it does allow others chasing you to come closer, so abusing the kick method is prevented by this penalty.
SPECIAL	BLOCK	0.1	Speed penalty for blocking. Blocking will reduce impact of others bumping into you, and it allows you to catch balls at higher speeds.
SPECIAL	NONE	1.0	No special moves - default.

An *Orientation* object is used for storing the rotation and position information, and for allowing easy conversion between the different coordinate systems. Table 5.11 illustrates the methods of the *Orientation* class that can be used for retrieving coordinates in either coordinate system.

The *team number* is either a 1 or a 2, depending on whether your team is playing on the left or on the right side. The team number is a property of all *RobotObjects*. The relative to self coordinates are calculated by the framework every *view-update*¹¹ milliseconds by the *SoccerAgent*.

5.5.2 Movement Model

The *SoccerAgent* is responsible for handling the Movement, and it will receive guaranteed frequent updates from all *RobotAgents* with their *MoveTuples*. A pseudo-code implementation of the algorithm for updating the

¹¹See Appendix B

Table 5.10: Three Different Coordinate Systems

Type	Description
FIELD	Field Coordinates. These correspond to the screen coordinate system, and are used for calculations in the framework. Rotation is, alike the unit circle, counterclockwise. See Figure 5.6 for an example. Width and Height refer to the parameters as defined in Appendix B.
TEAM	Team Coordinates. During team creation, it is impossible to know whether your team will play on the left or on the right. Team coordinates serve as a side-independent coordinate system, as they reason from the perspective of <i>your</i> team. See Figure 5.7 for an example.
RELATIVE	Relative Coordinates are very useful for many short-term calculations. They take your robot to be at the origin (0,0), and the viewing direction to be the X-axis at 0 degrees. Moving 90 degrees counterclockwise gives the Y-axis. All coordinates are defined in this X,Y coordinate system. See Figure 5.8 for an example.

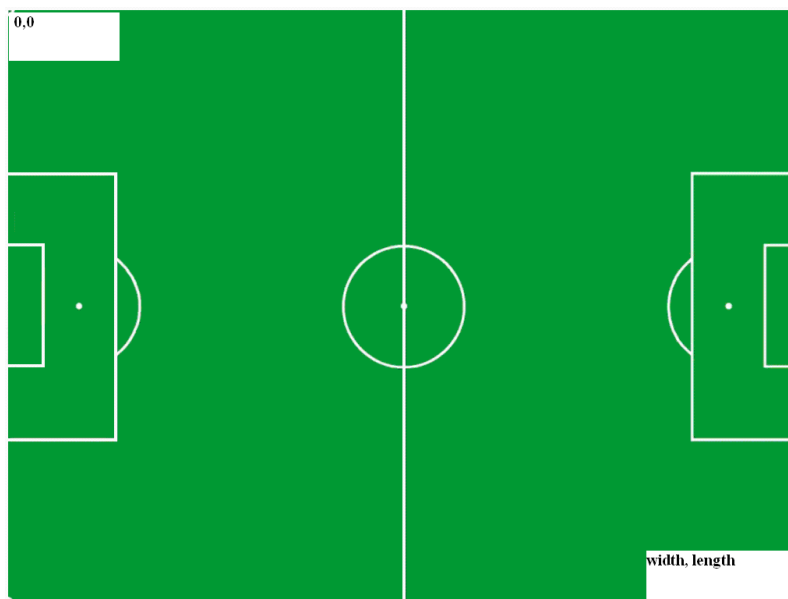


Figure 5.6: Field Coordinate System

positions is the following:

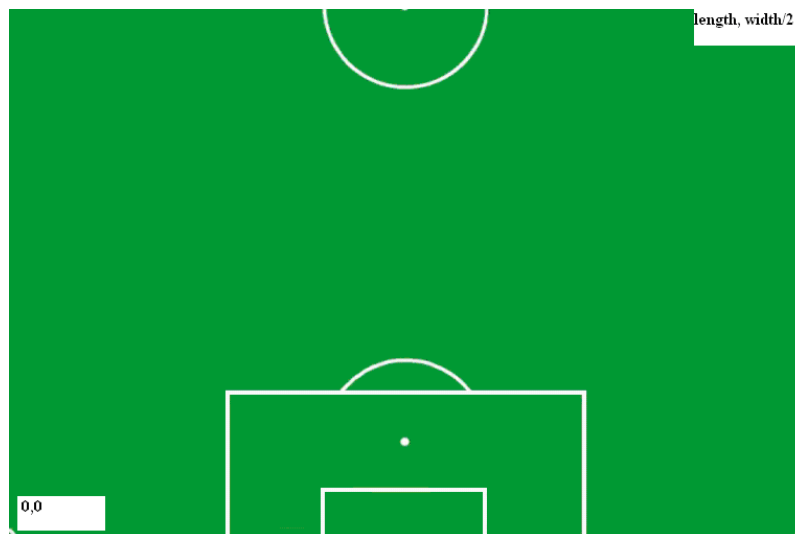


Figure 5.7: Team Coordinate System

Table 5.11: Using the Orientation Class for Retrieving different Coordinates

Type	Methods
FIELD	<i>getAbsolutePoint()</i> and <i>getAbsoluteRotation()</i>
TEAM	<i>getRelToTeamPoint(int pTeamNumber)</i> and <i>getRelToTeamRotation(int pTeamNumber)</i>
RELATIVE	<i>getRelToSelfPoint()</i> and <i>getRelToSelfRotation()</i>

Listing 5.5: The Movement Model

```

1
2 do every 1/refreshrate seconds while in GAME status {
3   update_ball_position();
4   last_MoveTuple[i] = Player i most recent MoveTuple;
5   for all players i {
6     calculate_robot_speed(penalties);
7     calculate_new_orientation(robots[i], speed,
8       last_MoveTuple[i]);
9     for all other objects j (so excluding i) {
10      distance_matrix[i][j] = distance_matrix[j][i]
11      =
12      calculate_distance(i, j);
13    }
14  }
15 }

```

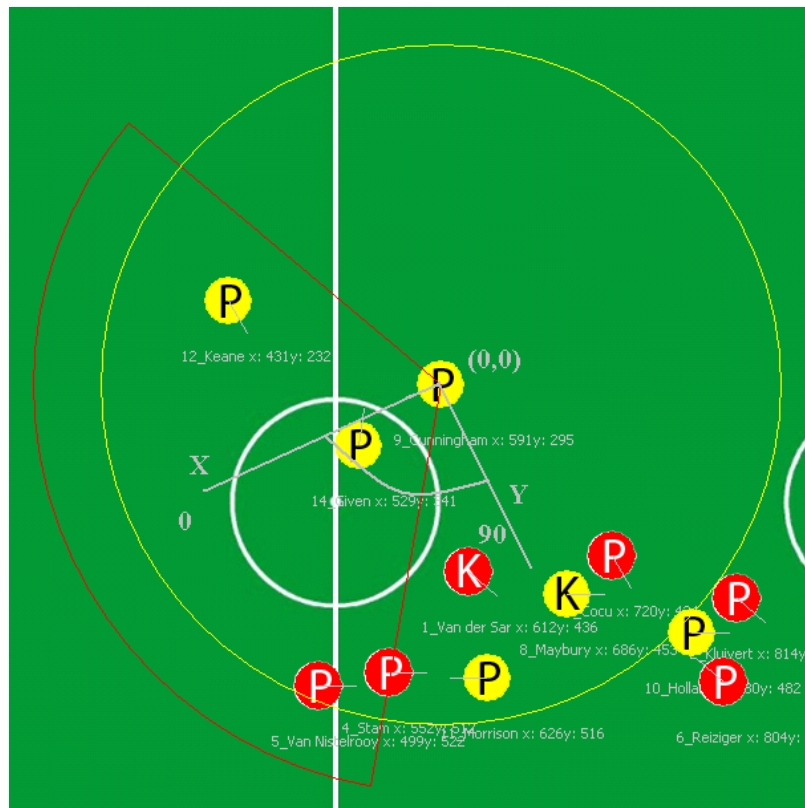


Figure 5.8: Relative Coordinate System

```

13   check_and_resolve_collisions ();
14 }
15
16 calculate_robot_speed(penalties) {
17     robot_speed = base_robot_speed * move_penalty *
18                 turn_penalty *
19                 special_move_penalty * shout_penalty *
20                 collision_penalty
21 }
22
23 calculate_new_orientation(robot, speed, last_MoveTuple)
24 {
25     if ( turn_left )
26         direction = direction + rotation_increment
27         last_MoveTuple-turn = straight
28     else if ( turn_right )

```

```

27     direction = direction - rotation_increment
28     last_MoveTuple.turn = straight
29
30     robot-direction = direction
31
32     if ( move_left )
33         direction = direction + 90 degrees
34     else if ( move_backward )
35         direction = direction + 180 degrees
36     else if ( move_right )
37         direction = direction + 270 degrees
38
39     distance = ((speed * (1 / refreshrate)) / field-
40                metric-length) *
41                field-width
42
43     x-change = cos(direction) * distance
44     y-change = sin(direction) * distance
45
46     robot-x-coordinate = x + x-change
47     robot-y-coordinate = y + y-change
48
49     validate(robot-position)
50 }
51 validate(position) {
52     if ( outside field-width or outside field-length )
53         put back on field
54 }

```

Verbally, the algorithm will update all robots according to their latest MoveTuple. It does this by first calculating the robot's speed and direction. Using the tournament parameters the (pixel) distance is calculated, and the position is changed. It is then validated to check if it's still on the playing field. Once this has been done for all robots, the collision detection method will detect and resolve collisions.

As the positions are calculated at a (much) higher frequency than the view update frequency, demanding from robots that they specify their desired movement every update would slow the system down, and add difficulty to implementations. The last MoveTuple received by the SoccerAgent will be used indefinitely, except for the turn part of a MoveTuple, which is reset to *straight* after being used once. Not resetting this would mean that an arbitrary number of turns would be executed before the next MoveTuple

would arrive to set it back to straight again.

The reason why only the last MoveTuple is used is to discourage players from flooding the system to get better movement (denial of service attack). In general, reassessing your robot's course every *view-update* milliseconds will be more than enough to get smooth movement.

5.5.3 Collision Model

The collision model is very straightforward. In a nutshell, it checks the distance between all objects. When two objects are too close, they are both separated so they do not collide anymore. The pseudo-code in Listing 5.6 illustrates this.

Listing 5.6: The Collision Model

```

1 for all objects i = 1..n {
2   for all objects j = (i+1)..n {
3     if distance(i,j) < 2 * player-radius
4       collisions.add(i,j)
5   }
6 }
7
8 for all collisions {
9   if collision with ball
10    if distance(i,j) < player-radius + ball-radius)
11      ball_collision(player, ball)
12  else {
13    distance = calculate distance both players need to
14      be moved so they
15    do not collide anymore
16
17    move player1 distance / 2 away
18    move player2 distance / 2 away
19
20    for all objects i excluding player1
21      distance_matrix[i][player1] = distance_matrix[
22        player1][i] =
23        calculate_distance(i, player1);
24
25    for all objects i excluding player2
26      distance_matrix[i][player2] = distance_matrix[
27        player2][i] =
28        calculate_distance(i, player2);

```

```

27     set last-collision-time for players 1 and 2 to
        current time
28     }
29 }

```

5.5.4 Ball Model

5.5.4.1 Ball Movement

The ball plays a central role in the simulator. Its model is very similar to that of the robot's movement model. The main differences are that when the ball enters certain areas, this is considered as a goal, and catching, kicking or blocking a ball are special. The pseudo-code in Listing 5.7 describes the ball model.

Listing 5.7: The Ball Model

```

1 update_ball_position() {
2
3     ball-speed = ball-speed * ball-friction
4
5     distance = ((ball-speed * (1 / refreshrate)) / field
        -metric-length) *
6     field-width
7
8     x-change = cos(direction) * distance
9     y-change = sin(direction) * distance
10
11    ball-x-coordinate = x + x-change
12    ball-y-coordinate = y + y-change
13
14    if ( in either goal area )
15        score()
16        reset ball to middle of field
17    else
18        validate(ball-position)
19 }

```

5.5.4.2 Ball Collisions

Following is a pseudo-code implementation of dealing with ball collisions. The parameters that are not covered by Appendix B are explained in Table 5.12. The ball collision model is straightforward. When a player is close to the ball, it will automatically 'catch' it, and the ball will auto-rotate

towards the player's direction. Kicking the ball gives it a certain (high) velocity and a weighted random direction error. A ball will bounce when the player is neither blocking, it is incoming at a relatively high velocity and it comes from outside the maximum catch angle. The ball collision model is illustrated in Listing 5.8.

Listing 5.8: The Ball Collision Model

```

1 ball_collision (player, ball) {
2   if distance(player, ball) < player-radius + ball-
   radius
3   if ( ball-speed > #threshold
4     and ball-incoming-angle > #max-catch-angle
5     and not special_move_block )
6
7     bounce_off
8   else
9     catch_ball
10
11   if catch_ball {
12     ball_direction = player_direction
13     if block
14       ball_position = new point at distance player-
   radius +
15       ball-radius + #catch-width at original angle
16     else
17       ball_position = new point at distance player-
   Radius +
18       ball-radius - #catch-width, at an
19       angle #degree-increment closer towards player
   -direction.
20       (auto-rotate towards player-direction)
21   }
22   else if bounce off
23     'reflect' the ball. Calculate incoming angle
   and send it back
24     with reduced speed in reflected direction
25
26   if ( kick )
27     ball-speed = #kick-speed
28     ball-direction = player-direction + (random *
29     kick-randomness)
30   else if ( block )
31     ball-speed = #block-speed
32   else

```

```

33 ball-speed = robot-speed
34 ball-direction = robot-direction

```

Table 5.12: Parameters dealing with Ball Collision

Name	Value	Description
Threshold	25	The number of meters per second above which catching of the ball is no longer possible.
Catch-width	10	The amount of pixels that the x and y change of the ball's position will be altered. This effectively puts the ball 'within' the player radius, causing 'collisions' to occur every update, ensuring the ball will stick to the player.
Max-catch-angle	45	The maximum angle where the ball is still catchable. If the ball comes in at an angle higher than this, it might bounce if it is too fast.
Degree-increment	45	Number of degrees that the ball will move towards the player-direction each second. This means that every update, the ball moves $1/\text{refreshrate} * \text{degree-increment}$ degrees towards the player-direction.
Kick-speed	45	The velocity in meters per second of the ball straight after receiving a kick.
Block-speed	0	The speed of the ball after an effective block.

5.6 User Interface Design

Attractive things work better, Donald Norman.

The user interface has only one primary purpose: *To visualize simulations between soccer teams*. As this is the most important aspect, there should be little distractions from this primary goal. The interface has to be intuitive, and visually appealing.

As can be seen in Figure 5.1, there are 4 different use cases for the simulator:

1. View and modify framework parameters.
2. Play a match.
3. Play a tournament.
4. Create or modify team.

The first will generally only be used by professional users, or prior to starting a (large) project. The tournament parameters will as such remain constant for a longer period of time, and they should not be an integral part of the user interface. Changing them from a pre-specified central location is sufficient.

Simulating a match is by nature very different from specifying teams or creating / modifying a team. The simulator requires all corresponding agents to be loaded and in the proper state, whereas the other uses can run independently of these. Although both types will be displayed in the same window, they are discussed separately.

5.6.1 Settings

5.6.1.1 Main Screen

Upon starting the simulator, the dominant task will be to play a match between the *challenge* team¹² and the (recently improved) team. Prior to being able to execute this task, the two teams that are to play a match have to be specified.

To make this as easy and intuitive as possible, there should be one *Main* screen, that is optimized for starting a game. The buttons - and their functionality - that will be present on this main screen are described in Table 5.13. This is visualized in Figure 5.9.

For user convenience, when the *Save MatchSetup* button is used - and the desired paths are automatically entered in their textfields, the process of starting a game is reduced to starting the framework, and clicking once on

¹²The reference team that has to be beaten



Figure 5.9: The Main Screen

the *Start Tournament* button. This one-click interface makes debugging a team through simulating a game after making a small change a satisfactory approach.

5.6.1.2 Team Creation Interface

The team creation interface performs the following tasks. This approach makes creating and modifying teams a lot easier than through editing their XML files:

- display (the base positions of) the robots,
- move robots around the field to give appropriate base positions,
- select a name for each robot,
- select an icon for each robot,
- select an agent for each robot,
- select a team name.

For these tasks, the interface will be displayed in two equally sized, vertically split regions. The left hand side visualizes the soccer field with the currently available robots, their positions, icons and names. The right hand side allows for the modification of these. The buttons and their corresponding functions that will be available on the right hand side are summarized in Table 5.14. The interface is provided in Figure 5.10.



Figure 5.10: The Team Creation Screen

Table 5.14: Interface Elements of Team Creation / Modification

Name	Type	Function
Start MatchSetup	Button	This opens a File Prompt to determine the desired name and location of the team XML.
Cancel	Button	Return back to the main screen
Team name	TextField	This textfield contains the team name that is displayed in game mode
Continued on next page		

Table 5.14 – continued from previous page

Name	Type	Function
Player name	TextField	Represents the name of the currently selected robot. Changing this will automatically change the corresponding robot's name on the field on the left hand side of the interface.
Agent Path	List	List displaying all agents that can be selected. Will highlight the agent that is used for the robot that is currently selected. Selecting another agent from this list will automatically change the robot's behavior. When the team is stored, the agent path list is also written to a property in Fleeble and loaded whenever the team creation screen is used. This avoids laboriously re-adding the agents after every team.
Add Agent Path	Button	This button opens a File Prompt that will allow users to manually add new agents to the Agent Path list.
Remove Agent Path	Button	This button will remove the currently selected agent path from the list. Robots that had this agent will take the next agent on the list as their behavior.
Icon Path	List	List displaying the names of all icons that can be selected. Will highlight the icon that is used for the robot that is currently selected. Selecting a different icon from this list will automatically change the corresponding robot's icon on the field on the left hand side of the interface. The paths to the icons are also stored using properties.
Add Icon Path	Button	This button opens a File Prompt that will allow users to manually add new icons to the Icon Path list.
Remove Icon Path	Button	This button will remove the currently selected icon path from the list. All robots that used this icon will use the next icon in the list.
Delete Agent	Button	This button deletes the currently selected robot from the team, and automatically from the field.
Continued on next page		

Table 5.14 – continued from previous page

Name	Type	Function
Hint	Label	This label instructs first time users how robots can be added to the screen, and reads: "Double-click somewhere on the field to create a new robot..."

Dragging a robot around the field is easily done through selecting and dragging the mouse over the robot. Releasing the mouse will place the robot at the desired location. The XML files created by this interface are exactly the same format as the ones described in section 5.4.7.

Adding robots is done by double-clicking on the field. When the permissible number of robots, as allowed by the framework parameters, has been reached, and the user is trying to add more robots, a note will appear to the user to inform him that his team is already full. No more robots can be added at this time.

5.6.1.3 Tournament Creation Interface

A tournament is a series of matches. Testing the performance of a team against one other team can be done by setting the game time arbitrarily high. Testing the team against a larger number of teams is a laborious process if a human operator has to wait for the current game to finish prior to being able to select another team. To automate this, and allow for a predefined series of matches to be simulated, the tournament creation interface is included in the system. The elements required to accomplish this are described in Table 5.15. A tournament can be composed of an arbitrary number of games, and the results of the games are stored and printed after each individual game, for convenient retrieval purposes. The interface is shown in Figure 5.11.

5.6.1.4 Error Handling

All settings deal with human input, and care should be taken to prevent the program from crashing. As the amount of options is still very limited, it is straightforward to prevent all possible input-related errors from corrupting the game.

In dealing with all lists¹³, adding elements that are already present is ignored. Removal of an element, if there is only one item on the list remaining is not permitted in the team creation, as this would automatically mean the

¹³These lists include the icons list, agents list, the match list and the team lists



Figure 5.11: The Tournament Creation Screen

removal of all created robots. The tournament creation menu allows for the removal of all elements.

Input through the file prompts and the input text fields is another probable cause of faulty human input. This is handled by catching the errors that will occur from not being able to successfully load or process the data, and providing feedback to the user with regards to why their request is not carried out successfully. In the case of an incorrect path to the team XML file, the text of the input fields is changed to *INVALID*, as can be seen in Figure 5.12.

To prevent confusion about the interface, and to provide adequate direct to-the-point help, tooltips are used throughout the interface. These pop up when a user keeps the cursor over a button with his mouse for over a second, and explain to the user what the function of the button is. Tooltips are added to most buttons.



Figure 5.12: Error Message from Incorrect Input

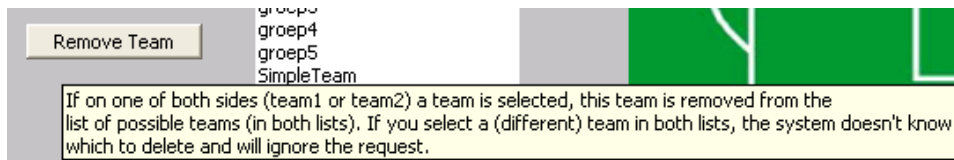


Figure 5.13: Example of the Tooltip Help

5.6.2 Game

The interface for displaying the game is naturally centered around the playing field. The only user input that is available is for controlling the states. A user can either pause the game or end it. To provide visual feedback regarding the visual and aural input areas of the robots, an extra feature is included such that clicking on a robot will mark these sensor areas with a red and yellow line, respectively. The interface is shown in Figure 5.14, the red and yellow lines marking the visual and aural sensor areas of the robot are displayed in Figure 5.15.



Figure 5.14: The Soccer Simulator Interface

Table 5.13: Interface Elements of the Main Screen

Name	Type	Function
Start Tournament	Button	This button takes a central location, and is the most important button on the main screen. It will start the sequences defined in sections 5.4.7 and 5.4.3.
Team 1 Path	TextField	This textfield contains the full path to the XML file describing the properties of team 1. Team 1 will play on the <i>left</i> side of the field.
Team 2 Path	TextField	This textfield contains the full path to the XML file describing the properties of team 2. Team 2 will play on the <i>right</i> side of the field.
Load Team 1	Button	This button will open a File Prompt, and allows users to browse through the hard drive to specify the location of team 1's XML.
Load Team 2	Button	This button will open a File Prompt, and allows users to browse through the hard drive to specify the location of team 2's XML.
Edit Team 1	Button	This button will start the team creation interface, and attempt to load the team at the location specified in the Team 1 Path.
Edit Team 2	Button	This button will start the team creation interface, and attempt to load the team at the location specified in the Team 2 Path.
New Team	Button	This button will start the team creation interface.
Tournament	Button	This button starts the tournament interface, where teams can be added in a list to start a series of sequential games.
Save MatchSetup	CheckBox	This is a checkbox that, when checked, will store the locations to the team XML files in Fleeble using Properties (See section 3.4.9) Even after shutting down Fleeble and restarting it, the preferred team paths will be stored and automatically entered to the corresponding textfields.

Table 5.15: Interface Elements of Tournament Creation

Name	Type	Function
Start Tournament	Button	This will start the (sequence of) matches as defined in the Match List.
Cancel	Button	Return back to the main screen.
Match List	List	List containing all matches that are to be simulated.
Add Match	Button	Will add a match consisting of the selected team from both team selection lists to the match list.
Remove Match	Button	Removes the match that is currently selected in the Match List.
Team 1 List	List	This list shows all teams that can be selected. This list is stored in a property so that upon restarting the simulator the teams are automatically added to the list.
Team 2 List	List	Same as Team 1 List.
Add Team	Button	Adds a team to both the Team 1 and Team 2 Lists.
Remove Team	Button	Removes the team that is selected, provided that either the same team is selected in both team lists, or that only one of the two lists has a team selected. If both team lists have the same selection, the remove button is ignored.



Figure 5.15: The Visual and Aural Sensor Areas of a robot

Chapter 6

Implementation

Ninety-ninety Law: The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time. Tom Cargill.

This chapter describes the implementation of the soccer simulator. The soccer simulator is based on the model that is described in Chapter 4, and is implemented in accordance with the design that is specified in Chapter 5. The chapter begins by discussing the approach that was used for implementing the simulator. Subsequently, the separate phases for the implementation of the system are described. The next section describes how to run the simulator, and will illustrate some features. Subsequently the testing of the framework is described. This chapter concludes with a number of suggestions for future work on the simulator.

6.1 Approach

Work on the graduation project started in early March 2006. The final deadline was May 15th, when the soccer assignment in the MKT-2 project would start.

The implementation of the system was done *incrementally*, in 4 phases. The first phase consisted of creating a basic framework that would load all required agents. In the second phase, the basic sensors, effectors and the movement model were implemented. In the third phase the GUI was implemented, and the functionality of the second phase was tested. The fourth phase consisted of adding a number of complex features, such as collision detection and ball movement. After the fourth phase, the MKT-2 project started to work with the simulator. To prevent students from having to download new versions and deal with new functionality all the time, the

development was discontinued. After the MKT-2 project, the team creation and tournament creation interfaces were added to the simulator.

6.1.1 Tools

During the development of the system, Eclipse [56], CVS [57], SWT [60], and Fleeble [9] were thoroughly used. The MKT-2 project already had a CVS-repository assigned on a server at Delft University of Technology. The software for the soccer simulator was added to this repository as a number of separate packages. To distribute the simulator, an installer was created using an Ant [58] script that would call an NSIS [68] script. The NSIS script basically describes what files are necessary, puts these in an executable container that will distribute the files to predefined (relative) locations. This executable will also contain the assignment manual and is distributed to the students.

6.2 First Phase: Initializing the Framework

The first implementation featured a very basic SoccerAgent, SoccerVisualizer, RobotAgent, and a SimplePlayer. The SoccerVisualizer displayed the options screen prompting for the two team XML files, and passed this on to the SoccerAgent. The SoccerAgent would then load 14 RobotAgents, and use the mechanism defined in the system design¹ to provide these RobotAgents with their name, and the PlayerInfo object containing the name of the PlayerAgent. The RobotAgents would subsequently load the PlayerAgent, and assign the corresponding virtualhost. The TournamentParameters class was also included and used from the start.

Once this basic framework, responsible for parsing the team XML files, and loading all corresponding robot and player agents, was functional, the remainder of the functionality was gradually added.

6.3 Second Phase: Sensors, Effectors and Movement

Starting with the basic framework, the second phase added basic implementations of the aural, visual, and movement models that were described in the design.

¹Using the Soccer.Robotloaded and Soccer.Nodename channels

6.3.1 Position Tracking

Prior to implementing the movement, aural or visual model correctly, the framework had to keep track of the current state of all objects on the field. The `VisibleObject` class was implemented, and a list of visible objects was created - one for each robot, and one for the ball. These were then placed on the base positions that were defined by the team's XML files.

6.3.2 Aural Model

The aural model was relatively easy to implement. First the `MessageTuple` class had to be implemented, and the `RobotAgent` had to wrap a `MessageTuple` with corresponding time and sender around all messages that were sent by a certain robot. Since the positions of all robots was known, checking which robots were within the shouting range was a straightforward task. The next step dealt with replying² the `MessageTuple` to all agents within the shouting range.

6.3.3 Visual Model

The visual model was significantly more difficult than the aural model. The most difficult aspect of the visual model was working with the angles. Calculating which objects were visible was relatively easy, but the design specified that the *relative coordinates*³ had to be calculated for all objects that were visible. On hindsight this is mathematically trivial, but during the implementation this was one of the pitfalls.

6.3.4 Movement Model

The `MoveTuple` was implemented, and integrated with the player, robot, and soccer agents. Calculating the correct translations and rotations was a tedious exercise, and a lot of debugging was done during the later phases before this was correctly implemented.

6.4 Third Phase: Graphical User Interface

The second phase concluded with a framework that would load the corresponding robots, player agents, and supported basic movement, vision and communication. Testing whether this functionality worked according to specification was rather difficult however, without the GUI. In the third phase, the GUI (`SoccerVisualizer`) was extended.

²A reply is a message over a certain channel directed at a specific agent rather than at all subscribed agents.

³See Section 5.5.1

The first step was implementing the `TournamentStatus` class, and establishing a basic structure for displaying and updating the field and all visible objects. There was some difficulty in coordinating the threads and synchronizing the data between the `SoccerAgent` and `SoccerVisualizer` initially, but this was resolved after debugging.

Then, the robots had to be visualized. Visualizing the robot's individual icon (user-specified!), name, position and the line representing the heading of the robot was relatively easy.

Now that the GUI was functional, the functionality from the second phase could be tested and debugged. At the end of the third phase, the simulator featured robots with fully functional visual and aural sensors, that were able to move across the field using `MoveTuples`.

6.5 Final Phase: Added Features

The third phase finished with a system that performed most of the tasks that were specified in the design, but was far from complete.

6.5.1 Collision Detection

The robots were able to move around the field, but there was no collision model implemented yet. The list of `VisibleObjects` that was created in the first phase was used to create and update a two-dimensional matrix containing all distances between the different objects. The collision model was implemented in accordance with the design (See Section 5.5.3).

6.5.2 Ball Movement

After the movement model and collision model were implemented, the ball movement model was implemented. The ball had to stay within the bounds of the field. It had to check every update whether it was inside the goal.

6.5.3 Change Behavior

The change behavior functionality was added to the soccer, robot and player agents, without any problems.

6.5.4 Other Features

A number of other important features were also added to the system:

- Scoring. The framework has to increment the score, stop play, reset the ball and robot's location to their base positions, and resume play again.
- Visualizing the aural and visual range.
- Pause / Play buttons. The (semi-transparent) pause, play and stop buttons were added to the field as the only buttons that should be visible throughout the game.
- Entering / Leaving field. To make the game appear more natural, the robots will enter the field from their corresponding sides, and move towards their base positions. When the game has ended, the robots move from their current positions towards the sides.

The framework was released when all these features were implemented and thoroughly tested. The team and tournament creation functionality was not included yet, as this was added after the project had ended.

After it had ended, the team creation and tournament creation interfaces were added. The team creation and modification interface was developed as a separate GUI layer that received its data from the SoccerVisualizer. The MatchSetupGenerator class was developed for this purpose. The interface was intensively tested before being released, to prevent faulty input from crashing the system. The TournamentGenerator was developed in an alike manner.

6.6 Running the Simulator

This section describes how to run the simulator, and illustrates some of the features. Prior to running the simulator, the user must first install the software. The installer will ask the user to specify a directory, and extract all required files there. Then, the user has to start Fleeble, and load the SoccerAgent⁴

Entering / Exiting the Field Figure 6.1 illustrates how robots will enter the field⁵, and Figure 6.2 shows the process of exiting the field⁶.

⁴The SoccerAgent is located in `$Installation_directory/mkt2/soccer/SoccerAgent.java`

⁵This process starts after the team XML files have been submitted by the user, and the required RobotAgents have been loaded.

⁶This occurs when the match is over, or when the user ends the game.

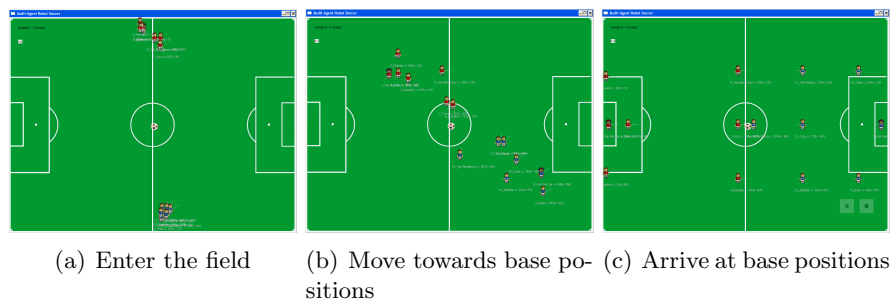


Figure 6.1: Entering the Field

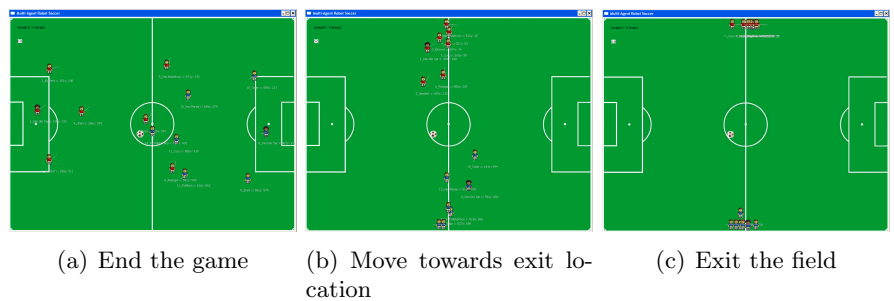


Figure 6.2: Exiting the Field

Play, Pause, and Stop buttons Figure 6.3 illustrates the semitransparent buttons in the right-hand bottom corner of the user interface that can control the state of the game. When the game is playing, the left button displays the pause logo. When the game is paused, it displays the play logo.

Score and Game Time Figure 6.4 illustrates the names of both teams, the corresponding scores, and the tournament time. The tournament time may differ from the actual time, as the tournament time is linked to the amount of movement updates that have been processed. This is explained in Section 5.4.4.

Robot Visualization Figure 6.5 shows a typical moment in the game. The red line determines the visual range of the robot. The yellow circle defines its aural range. The grey line indicates the current heading of the robot. The name of the robot and the coordinates are displayed beneath the icon. Channels on Fleeble can also be viewed by humans, and Fleeble attempts to put the data in a human-readable format. Figure 6.6 illustrates what the robot in Figure 6.5 will receive on the See channel. This is an Ar-

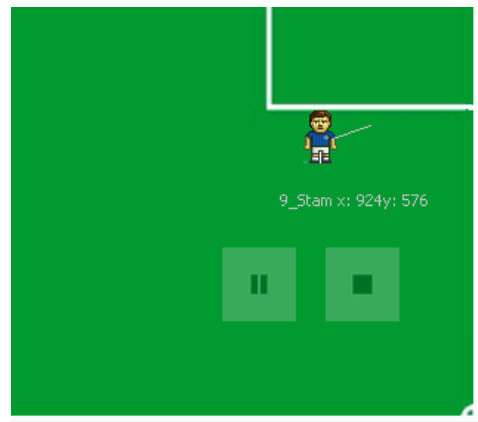


Figure 6.3: Pause / Stop Buttons

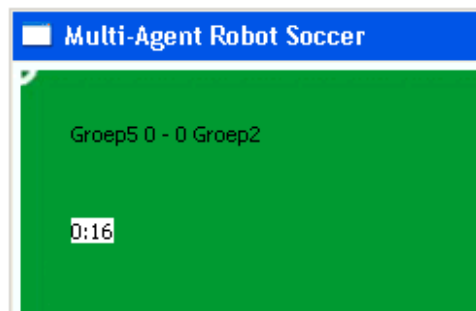


Figure 6.4: Game Time, Score and Team Name

rayList of RobotObjects⁷, and for each RobotObject, the properties such as team name, team number, radius and name are displayed. This visualization makes debugging players a lot easier.

⁷RobotObject is a subclass of VisibleObject, and ArrayList is a subclass of the List interface. As such, the input is a List of VisibleObjects, as described in the design



Figure 6.5: Robot Visualization



```
Connections See X
modCount: 0
]
mkt2.soccer.RobotAgent@2006-08-17 22:26:57: [object of class: java.util.ArrayList
  elementData: [Array of type: java.lang.Object and size 10
    item #0: [object of class: mkt2.soccer.tuples.RobotObject
      fTeamName: Groep5
      fTeamNumber: 1
      fOrientation: [Object of type 'mkt2.soccer.tuples.Orientation']
      fName: 6_Reiziger
      fRadius: 19.0
    ]
    item #1: [object of class: mkt2.soccer.tuples.RobotObject
      fTeamName: Groep5
      fTeamNumber: 1
      fOrientation: [Object of type 'mkt2.soccer.tuples.Orientation']
      fName: 7_Cocu
      fRadius: 19.0
    ]
    item #2: [object of class: mkt2.soccer.tuples.RobotObject
      fTeamName: Groep2
      fTeamNumber: 2
      fOrientation: [Object of type 'mkt2.soccer.tuples.Orientation']
      fName: 8_Van Der Sar
      fRadius: 19.0
    ]
    item #3: [object of class: mkt2.soccer.tuples.RobotObject
      fTeamName: Groep2
      fTeamNumber: 2
      fOrientation: [Object of type 'mkt2.soccer.tuples.Orientation']
      fName: 9_Stam
      fRadius: 19.0
    ]
    item #4: [object of class: mkt2.soccer.tuples.RobotObject
      fTeamName: Groep2
      fTeamNumber: 2
      fOrientation: [Object of type 'mkt2.soccer.tuples.Orientation']
```

Figure 6.6: The See Channel

6.7 Testing

Testing the framework was done throughout all stages of the development. There is a distinction between testing the functionality, and testing the usability. The functionality tests the compliance with the system model, whereas the usability tests whether the users of the system will interpret the offered functionality as it was designed by the developer.

The functionality of the system was debugged and tested during the development. Following the addition of a new feature, such as the aural model, visual model, or movement model, all possible uses of the feature were thoroughly tested. This testing occurred in one or more of the following manners:

- Use the "*System.out*" and "*System.err*" channels to print certain (status) information.
- Use the debug mode of eclipse to run through the program step by step to determine the exact location of an error.
- Read the human-readable representation of the data that is sent over channels.
- Look in Fleeble at the list of agents and their corresponding agent states.
- Analyze the behavior of the robots on the user interface⁸.

The usability of the user interface was initially evaluated by the other teachers of the MKT-2 project. After several small adjustments to the interface, it was distributed to the students. As the only interaction that the user has with the system lies in the entering of the path to the team XML (after the project also with creating the team / tournament), very few things can go wrong. The tooltip texts were added to prevent any possible confusion. Since the data that is entered can be stored in properties, the system has to be set up correctly once, and it will start playing a match with a single click from then onwards. There were no remarks or bugs submitted by the students.

6.8 Future Work

The simulator that is developed during this project has no known issues or bugs and is considered stable. However, a number of possible technical improvements do exist. These improvements are listed below:

⁸The user interface also displays the coordinates of the robots, and their heading. When a robot is supposed to turn left and it does not turn left, something is obviously wrong.

1. Move the 'DisplayThread', the thread that presently resides in the SoccerVisualizer and is responsible for calling the methods in the SoccerAgent that process the visibility and movement, towards the SoccerAgent. The SoccerVisualizer should be a thin interface layer that can be used to display the game, and the game should not be dependant on the game. This would also allow simulations to run without the GUI.⁹
2. Implement network option. Users should be able to join or host matches and tournaments with other users. How this should be implemented depends on the exact required usage. Setting up one central soccer team database and connecting to this, downloading the desired team and playing against this team would be a possible implementation.
3. Extend the model with uncertainty, to model various states of technological progress, and use this to test the impact of this progress on the performance of a team.
4. Add more human soccer features, such as specifying the speed and direction of a kick, modeling the goalkeeper as a robot with special physical capabilities, or kicking the ball with effect.
5. Add another layer over the player layer that contains pre-implemented functions for, amongst others, movement. This would effectively reduce the process of creating a team to defining a good rulebase.

⁹The simulation will not be able to run faster however, since the time that is required by Java and Fleeble to process all input and output by all robots is independent of the thread that implements it, and the user interface is very basic and does not require a lot of processing power.

Part III
Research

Chapter 7

Agent Strategies

You may not be interested in strategy, but strategy is interested in you. Leon Trotsky.

7.1 Introduction

The simulator that has been introduced in the previous part of this thesis is suitable for educational purposes and for research on high-level strategic behavior. This chapter illustrates the possibilities of the simulator as a tool for research on these high-level AI techniques.

The chapter begins by identifying a number of low-, medium-, and high-level player skills, that map high-level commands to the basic action commands available in the simulator. The next section describes a number of inference steps that map the basic percepts to high-level information about the environment. Subsequently, agent and team strategy is discussed, adaptive behavior is introduced, and the rule-based action selection mechanism is explained. The next section describes how a player could be implemented using a multi-agent system. The chapter concludes with an overview.

7.2 Player Skills

Skill (noun): Ability to produce solutions in some problem domain

The behavior of an agent is determined by its reasoning mechanism, but it is restricted by the player skills. The reasoning mechanism can use the player skills as building blocks. Prior to discussing what reasoning mechanisms exist, it is important to determine the player skills that are possible in the simulator.

In his Master's thesis, Jelle Kok [66] described a number of low-, medium-, and high-level skills that were used in his team for the soccer server. These skills can also be implemented using the simulator. Certain skills are pre-implemented, and some are not relevant in the simulator. The other skills are described in this section.

7.2.1 Low-level Player Skills

The following low-level player skills are all defined in terms of the basic action commands that are possible in the simulator. A number of trivial low-level skills are not described in detail as they are already contained in the implementation of the basic action commands. Examples are *kicking the ball*, *catching the ball*, *moving with the ball*, *turning with the ball*, *blocking an incoming player*, *moving / turning in a certain direction*, *communicating a message*.

7.2.1.1 Turn the Robot towards a Point

This skill enables a robot to turn towards a predefined point. Turns in the model are defined using a fixed angle. As such, it is only sensible to turn when the angle towards to goal position is larger than this angle. The relative rotation of any objects are already available, with an angle of 0 corresponding to the heading of the robot.

Listing 7.1: Turn the robot towards a point

```

1 turnToPoint(Point p) {
2   if ( cosine ( relative_rotation(p) ) < rotation-
      increment )
3     if ( sine ( relative_rotation(p) > 0 )
4       turn left
5     else
6       turn right
7   else
8     turn straight
9 }
```

7.2.1.2 Scout the Area

This skill enables a robot to look around a larger area. As robots have a small field of vision, opponent robots may get by if robots look in the same direction. This skill will continuously turn the robot, such that the robot will observe the full 180 degrees in the direction of the opponent side of the field. It will alternate between turning left and right. This skill can be used to search for the ball or opponents while remaining in the same position.

Listing 7.2: Scout the Area

```
1 boolean left // true means we are turning towards left
   , false
2 towards right
3
4 scoutArea()
5   if ( left = true & cosine(team_coordinates_heading)
6     < cosine(PI / 2 - view_angle / 2))
7     & sinus(team_coordinates_heading) > 0 )
8     left = false
9   else if ( left = false & cosine(
10    team_coordinates_heading)
11    < cosine(PI / 2 - view_angle / 2) & sinus(
12    team_coordinates_heading) < 0 )
13    left = true
14  if ( left )
15    turn left
16  else
17    turn right
```

7.2.2 Intermediate-level Player Skills

7.2.2.1 Turn the Robot towards an Object

This skill enables a robot to turn to a certain object.

```
1 turnToObject(Object o) {
2   turnToPoint(o.position)
3 }
```

7.2.2.2 Track the Ball

This skill enables a robot to track the ball.

```
1 trackBall() {
2   turnToPoint(ball.position)
3 }
```

7.2.2.3 Move to a Position

This skill enables a robot to move to a certain position. The agent will have to decide whether to sidestep to the left or right, move forward or backward. The agent can also decide to turn the robot. This has as its disadvantage

however, that the move to a position will override any turning skills¹. As such, this pseudo-code implementation keeps the notions of turning and movement separated. The implementation of this skill determines for a large part the performance of the player, and some complexities can be added. An example would be taking into account all known positions of other robots, and rule out those moves that will cause imminent collisions. This particular implementation will use its predicted speed to determine which movement gets the robot closest to the goal.

Listing 7.3: Move to a Position

```

1 move(Position goal_pos)
2 {
3   moves = { FORWARD, BACKWARD, LEFT, RIGHT, IDLE }
4   for all moves {
5     calculate predicted new position //(assuming no
6       collisions, and
7     //taking the penalties into account)
8   }
9   do the move where the resulting predicted position
10    has the
11    smallest distance to the goal position

```

7.2.3 High-level Player Skills

7.2.3.1 Intercept a Ball

This skill enables the robot to intercept a ball. The robot will have seen a ball that has a certain velocity and a certain heading. The position of the ball can be predicted exactly throughout time, if there are no other objects moving the ball. The robot will always catch a ball when it is less than player-radius pixels away. For implementing this skill, a number of different approaches exist:

1. Predict the path that the ball will follow, and try to move towards the point that is the closest to the ball, as to minimize the risk of other robots taking the ball
2. Predict the path that the ball will follow, and try to move as little as possible from your own position while still catching the ball
3. Predict the path that the ball will follow, try to ensure that you catch it, while anticipating the movement of other nearby robots, and hence

¹For instance turn towards a position, turn towards an object, track the ball

including the probability that you will get a collision and the corresponding penalty

4. Predict the path that the ball will follow, ensure that you will catch it, while already turning and moving towards the position that is best for passing the ball to others or moving towards the goal

Using `hadCollision(self)` the robot can determine whether he had a collision penalty, and anticipate on this. The ball will always move in a straight line with a speed that reduces with a constant factor. The following pseudo-code illustrates an implementation of the first of the aforementioned approaches. The player will intercept the ball as fast as it can.

Listing 7.4: Intercepting the Ball

```

1 intercept() {
2   if ( smallest distance to predicted ball trajectory
3     < player-radius )
4     //we will catch the ball unless someone else
5     //intercepts the
6     //intercept
7   else
8     // determine optimal point
9     Tmax = the time that it takes for the ball to come
10    to a halt
11    // this is when ball.speed * 0.96^Tmax) < 0.1
12
13    for i = 1...Tmax {
14      T_required = calculate the amount of time it
15      would take to move to the
16      position of the ball at time i
17
18      if ( T_required <= i )
19        optimal_point = position of ball at time i
20    }
21    move(optimal_point)
22  }

```

7.2.3.2 Passing the Ball

This skill enables a robot to pass the ball to a teammate. Before a successful pass can be given, the player has to have the correct heading. The closer the recipient is, the higher the chance that he will catch the ball, due to the small randomness that is added to the direction of the kick. When the

robot has to turn first, it is assumed the pass method is called again after the next visibility update.

Listing 7.5: Pass the Ball

```

1
2 pass(Position p) {
3   if ( relative_rotation(p) < rotation_increment )
4     kick
5   else
6     turnToPoint(p)
7 }

```

7.2.3.3 Give a Leading Pass

This skill enables the robot to give a leading pass to a teammate. A leading pass is a pass aimed in front of the player. The distance ahead of the receiver can be specified as an argument for this function.

Listing 7.6: Leading Pass

```

1 leadingPass(RobotObject o, distance d)
2 {
3   goal_position = position at distance d directly in
4   front of robot o
5   pass(goal_position)
6 }

```

7.2.3.4 Give a Through Pass

This skill enables the robot to give a through pass. This is a type of pass that is into the open field, 'through' opponents, such that a team mate can take it and (hopefully) score. There are many implementations for this type of pass, as many assumptions have to be made and a lot of risk has to be taken. For instance when an opponent robot is facing the other way, it could be assumed that the robot would not be aware of the kick of the ball, and will not intercept it, and a smaller safety line is required. A more conservative implementation could calculate whether the opponents will be able to intercept the ball.

Listing 7.7: Through Pass

```

1 throughPass(RobotObject o ) {
2   range = calculate the 'safe' range in between (
3   through) the opponents
4   // turning is always with 20 degree increments, so '
5   all possible '

```

```

4 // angles in the range is usually very few
5 for all possible angles in range {
6     trajectory = calculate the path of the ball if
        kicked
7     //depending on the amount of turns required for
        the shot,
8     //add visibility-update milliseconds per turn
9     best_angle = if there is intersection of robot o,
        assuming it moves
10    at maximum speed forward, with the trajectory,
        then this is
11    best angle
12 }
13 best_point = the intersection point using the best
        angle
14 pass(best_point)
15 }

```

7.2.3.5 Clearing the Ball

This skill enables the robot to kick the ball in a 'safe' direction of the field, if he is unable to make a (safe) pass. This can win the defensive side some time to re-organize and relieve some of the pressure of the opponents. The goal is to shoot the ball in the direction of the opposing team.

```

1 clearBall() {
2     if ( heading opposing side ) //prevent making an own
        goal
3     kick
4 }
5 }

```

7.2.3.6 Move towards Opponent Goal

This skill enables the robot to move towards the opponent's goal. To prevent opponents from being able to block the movement easily by moving directly in between the ball and their own goal, this skill can be implemented to consider the position on the field.

Listing 7.8: Move towards Opponent Goal

```

1 moveToGoal() {
2     if (distance from self to opponent goal is larger
        than
3     threshold)

```

```

4     move along the line defined by the x coordinate of
        own base
5     position towards the goal
6     //this means flank players will attack over the
        flank
7     else
8         move towards goal
9 }

```

7.2.3.7 Move Free from Teammate

This skill enables the robot to move free from its teammate. When a teammate has the ball, the robot should certainly not obstruct the teammate with the ball. Depending on the relative location of the teammate, the implementation of this method will attempt to proceed forward along the flank that is still free.

Listing 7.9: Move Free from Teammate

```

1 moveFree() {
2   if (x difference between ball and self > threshold)
3     move along the line defined by the x coordinate of
        own base
4     position towards the goal
5   else
6     if ( ball to left of center )
7       move along line a certain distance to the right
        towards the
8       goal
9     else
10      move along line a certain distance to the left
        towards the goal
11 }

```

7.2.3.8 Avoid Opponent

This skill enables the robot to avoid a certain opponent. This skill is required when the robot has a goal position and another robot is in the direct line between the current and the goal position. The avoid skill takes an argument indicating which opponent should be avoided. Depending on the heading of this opponent, the robot will either try to move left or right around it.

Listing 7.10: Avoid the Opponent

```

1 avoidOpponent(RobotObject closest_enemy) {

```

```

2  if ( sinus(closest_enemy.relative_heading) > 0 ) //
    opponent is left
3  if ( sinus(closest_enemy.team_heading) < 0 )//
    looking towards
4  right
5    // move right
6  else // looking towards left
7    // move right
8  else // opponent is on right side
9    if ( sinus(closest_enemy.team_heading < 0 )//
    looking towards
10   right
11     // move left
12   else // looking towards left
13     // move left
14 }

```

7.3 Inference

The mapping from percepts to actions is done through reasoning. Skills describe how the possible actions can be used to accomplish certain goals. These goals have to be formulated based on the percepts. This section describes a number of facts that can be inferred from the environment, and can be used for reasoning.

7.3.1 Determine if an Object's Position is known

This skill enables the robot to find out whether an Object's position is known. The object's position can be known through direct visual perception, but also through other robots communicating the position of the object. The position is considered to be known when it was observed or received less than a fixed number of time ago.

Listing 7.11: Determine if an Object's Position is Known

```

1  timeout = 7 seconds
2
3  isObjectknown(VisibleObject object) {
4    if ( history.contains(object) & now - object.time <
5        timeout )
6      object_known = true
7    else
8      object_known = false
9  }

```

7.3.2 Determine if an Object is Visible

This skill enables the robot to find out whether a certain object is currently visible. When the last observation of the object was less than a fixed number of time ago, it is considered visible.

Listing 7.12: Determine if an Object is Visible

```

1 timeout = visibility - update
2
3 isObjectVisible(VisibleObject o) {
4   if (objectKnown(o))
5     if (self.distance(o) < view-distance &
6         cosine(o.relative-rotation) > cosine(view-angle / 2)
7         & now -
8         o.time < timeout )
9       object_visible = true
10    else
11      object_visible = false
}
```

7.3.3 Determine if the Ball is Visible

This skill enables the robot to find out whether the ball is currently visible. When the last observation of the ball was less than a fixed number of time ago, it is considered visible.

Listing 7.13: Determine if the Ball is Visible

```

1 timeout = visibility - update
2
3 isBallVisible() {
4   if (ball-known)
5     isObjectVisible(ball)
6 }
```

7.3.4 Determine if the Ball is Kickable

This skill enables the robot to find out whether the ball is currently kickable. A ball is kickable when it is directly in front of the robot (visible), and it is close enough.

Listing 7.14: Determine if the Ball is Kickable

```

1 isBallKickable() {
2   if (ball-visible)
```



```

3   if ( self.distance(ball) < ball.radius + robot.
      radius )
4     ball_kickable = true
5   else
6     ball_kickable = false
7 }

```

7.3.5 Determine the Closest Opponent

This skill enables the robot to find the closest opponent.

Listing 7.15: Determine the Closest Opponent

```

1 findClosestOpponent() {
2   for all opponents
3     if (isObjectVisible(robot))
4       if (self.distance(robot) < closest_opponent)
5         closest_opponent = robot
6 }

```

7.3.6 Determine whether an Opponent is up Ahead

This skill enables the robot to find whether there is an opponent coming up ahead. This is the case when there is an opponent that is less than a certain distance away from the robot

Listing 7.16: Determine if an Object up Ahead

```

1 isOpponentUpAhead() {
2   if ( self.distance(closest_opponent) < threshold )
3     opponent_up_ahead = true
4   else
5     opponent_up_ahead = false
6 }

```

7.3.7 Determine whether there is a Teammate Standing Free

This skill enables the robot to find whether there is a teammate standing free. There are a number of approaches to determine this. The most important restriction to a correct implementation of this is the limited field of vision. The following pseudo-code implementation will result in accurate results, but is unfortunately only useful for small distances. This skill is useful when reasoning about passing the ball to a teammate. To prevent the ball from bouncing off the teammate, a certain minimum threshold distance is enforced.

Listing 7.17: Determine if Teammate is Free

```

1 for all robots {
2   if (time.now - robot.time < timeout & visible)
3     if (teammate & self.distance(robot) <
4         closest_t teammate.distance)
5       closest_t teammate = robot
6     else if (opponent & self.distance(robot) <
7               closest_o opponent.distance)
8       closest_o opponent = robot
9   }
10 if ( closest_t teammate.distance < closest_o opponent.
11     distance &
12     closest_t teammate.distance > threshold)
13   teammate_free = true
14 else
15   teammate_free = false

```

7.3.8 Determine whether a Teammate has the Ball

This skill enables the robot to find whether a teammate has the ball. If this is the case, the robot may want to move to a free place on the field.

Listing 7.18: Determine whether a Teammate has the Ball

```

1 teammateHasBall() {
2   for all teammates
3     if ( team_mate.distance(ball) < ball.radius +
4         robot.radius)
5       team_ball = true
6     else
7       team_ball = false

```

7.3.9 Determine whether the Ball is Free

This skill enables the robot to find whether the ball is on a free location in the field. When the closest player to the ball is more than a fixed distance away from the ball, it is considered to be free

Listing 7.19: Determine whether the Ball is Free

```

1 isBallFree() {
2   for all robot {
3     if ( robot.distance(ball) < free-distance)

```

```

4     ball_free = false
5   }
6   ball_free = true
7 }

```

7.3.10 Determine if the Ball Position is known

This skill enables the robot to find out whether the ball position is known. The ball position can be known through direct visual perception, but also through other robots communicating the position of the ball.

```

1 isBallKnown() {
2   isObjectKnown(ball)
3 }

```

7.3.11 Determine if there is Chaos around the Ball

This skill enables the robot to find out whether there are at least 3 robots that are close to the ball.

Listing 7.20: Determine whether there is Chaos around the Ball

```

1 isChaos() {
2   for all robots {
3     if (robot.distance(ball) < threshold)
4       list.add(robot)
5   }
6   if list.size > 3
7     chaos = true
8   else
9     chaos = false
10 }

```

7.3.12 Determine if Player had Collision

This skill enables the robot to find out whether a certain player had a recent collision, and is still suffering from the speed penalty. This information is more informative than the player speed, since sidestepping to the left will also yield a lower speed, but the robot will be able to move faster. Knowing that a robot had a collision is useful for determining whether he can for instance intercept a ball.

Listing 7.21: Determine whether there was a Collision

```

1 hadCollision(RobotObject p)

```

```

2  speed = determinePlayerSpeed(p)
3  //determine from the t-1 position and heading,
4  //and the new position and heading what movement
   penalties
5  //were issued
6  speed = speed * 1 / movement_penalties
7  if ( speed >= shout-slowdown * player-speed )
8      collision = false
9  else
10     collision = true

```

7.3.13 Determine Player Speed

This skill enables a robot to determine the speed at which a player is traveling. All robots have a basic speed, but may be affected by shouting and / or collision penalties, or by a movement penalty². The ball speed is always given as a property of the BallObject. The player speed can be determined only if the player has been observed in two consecutive visibility updates, in the following way:

```

1  determinePlayerSpeed(RobotObject p) {
2      p_last_update = history(t-1, p)
3
4      speed = distance ( p_last_update , p ) / visibility -
   update
5  }

```

7.4 Strategy

The previous sections have described how the basic percepts can be translated to derive high-level information about the environment, and how high-level commands can be executed in terms of the low-level effectors using several player skills. The strategy of a team combines these two by taking the high-level percepts as its input, reasoning about these, and deriving the high-level output.

This section begins by introducing social laws and describing their importance in the simulator. Then formations are discussed. First the in-game change of base positions is discussed, then the change of role allocation, and how the positioning and role allocation changes can be accomplished without any central authority are discussed. Then, the rule-based action selection mechanism is explained.

²All movement that is not straight ahead and forward is penalized in some way

7.4.1 Social Laws

The soccer simulator presented in the previous chapters penalizes direct communication through reducing the robot's maximum speed for a certain amount of time. Because of this penalty, the decision to communicate becomes an integral part of strategic design. Although at certain times communication, such as shouting "*help me out here, I have the ball and there are 2 opponents coming*" could prove useful, the speed penalty that is inflicted on the communicating robot would give opposing robots the opportunity to come closer and steal the ball.

Social laws are widely adopted conventions, such as driving on the proper side of the road. The strategies of the agents are preprogrammed, and as such reasoning about how other³ agents reason can easily be done. Reasoning about how other players will reason, and anticipating on this action removes the necessity for communication.

Throughout the design of a strategy, the inclusion of social laws should play an important role. Robots can either play an active or a passive role in the game. A robot with the ball will always play an active role. Robots that are 'locked' in the game because they are covered by one or more opponent robots, and they do not have the ball, can be considered to take a passive role at that time. The speed penalty for passive robots is less troublesome than for active robots. Robot strategies should give a complete definition of when robots are active and when they are passive.

7.4.1.1 Social Laws versus Communication

The decision whether to communicate, or to reason about the environment is defined inside the player agent. Social laws function as conventions that are agreed upon by all individual player agents. An example social law would be the following: *If a teammate has the ball, do not attempt to steal the ball from him.* The design and implementation of these social laws throughout a team is an important task. The different player behaviors are expected to act differently, but comply with the same social laws.

The main goal for including such social laws in strategies is to reduce the amount of communication (or: derive as much information⁴ from the environment as possible). The communication is restricted by the range

³This only goes for teammates, the opponent's strategy is a black box

⁴Through observing the behavior of teammate agents and knowing how these will respond to certain events, these events themselves can be derived. For instance when a teammate that has the ball is turning towards you, the information you can derive is that he is probably going to pass the ball towards you. Depending on the social law, he may try for a regular pass, a through pass, or a leading pass.

and by the speed penalty, and especially for robots taking an active role in the game, these restrictions are severe.

7.4.2 Communication

The soccer model was specifically designed to encourage users to consider the autonomous reasoning (social laws) versus communication tradeoff. Communication plays an essential role in any highly cooperative team however. The very limited field of vision for robots drastically reduces the amount of information they can reason about. Since any amount and type of data can be communicated through shouting, visual information from neighboring robots can be transmitted and used to draw better conclusions.

To illustrate how powerful communication is in strategy, Figure 7.1 shows *you*, with the ball, unaware of the presence of two opponents, and the teammate. The teammate sends a message with this information, and suggests a plan⁵. *You* will reason about the likelihood of this plan succeeding (based on the newly gained information), and if the plan is better than the previous plan, execute this. The teammate will be able to observe by the actions of *you* whether or not the suggested plan is being executed. The teammate will suffer from the communication speed penalty, but the robot with the ball (*you*) does not need to communicate.

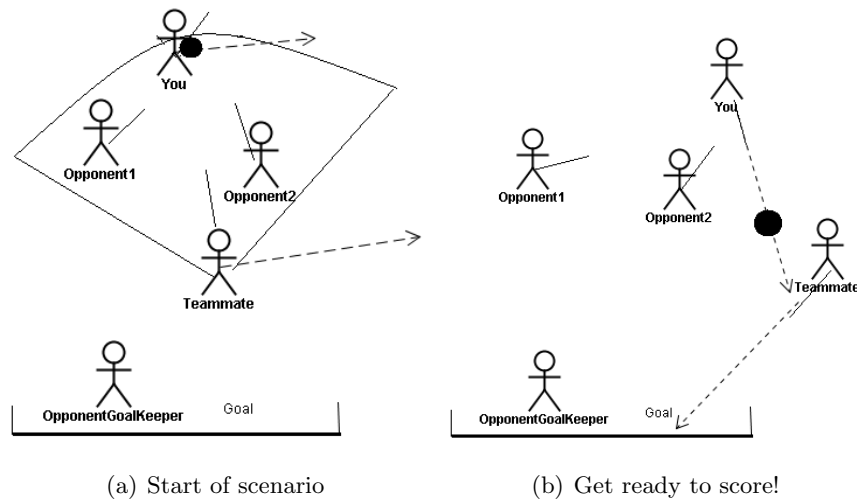


Figure 7.1: Example use of Communication to Outplay Opponents

⁵This particular plan consists of the teammate moving towards the flank, and *you* passing the ball to the teammate.

In general, deciding on what plan to execute should be decided by the players taking an active part in the play, as they have the visual information of nearby objects that other passive robots might not have. When it may seem like a good plan to pass the ball to a player by a passive observer, this plan should be *suggested* to the active player. As communication by the active player has a large cost, the passive players should attempt to anticipate based on the active player's movement. After an active player has passed the ball, it can safely communicate and inform neighboring teammates about the plan it has attempted.

7.4.3 Formation

An important aspect of team performance is the formation. In the soccer simulator, all aspects dealing with formation are described in the team's XML document that specifies the base position, behavior, icon and name for all robots. How the robots are distributed among the field and how the roles are divided plays an important role in the strategy.

7.4.3.1 Positioning

The base positions that are defined in the team XML file define the positions on the field where all individual robots will start. The strategy of the robots can be such that the robots will return towards this position when they take a passive role in the play. Because the robots will be distributed among the field, this prevents the phenomenon known as 'kiddie soccer', where all player cluster around the ball, from occurring. The definition of when a robot is considered to be passive should be an integral part of the strategy, but a simple definition would be when the player is not close to the ball⁶. A team can have multiple strategies, and the base positions for the individual robots are an important part of strategies. When playing extra defensive, there will be more robots located around the goal, whereas an offensive strategy will use extra robots close to the enemy goal.

7.4.3.2 Role Allocation

The other key aspect of the formation is the role allocation. The simulator allows individual robots to have individual behaviors. The distribution of these behaviors over the players is inherently a big part of the strategy. Players that are always close to the enemy goal will exhibit different behavior than the player that is defending the own goal. A technique that is frequently used to simulate this, is to define a robot's behavior based on his current position on the field. The field is divided in several areas, and the robot

⁶However, when complex plans involving multiple agents for scoring a goal are used, all these robots could take an active role without necessarily being close to the ball.

acts differently in each area. A problem occurs with this approach when, for example, there are multiple agents in the goal area. All agents will exhibit goal tending behavior. To prevent this, the different behaviors for the players are introduced. When an evaluation mechanism in the agents recognizes that the strategy that is currently executed is not very effective, it can be a good idea to change the roles of certain agents. An offensive player could be transformed to a defensive player, and improve the team's performance.

7.4.3.3 Changing the Strategy

An important aspect of the game is the lack of any centralized control. Changing the positioning or the role allocation throughout a game will have to be done by the robots themselves. Various approaches for accomplishing this exist. Changing the strategy is a measure that can be taken when it is expected that the new strategy will yield a better team performance. Changing the role allocation⁷ is generally done over a longer term, whereas the positioning of individual robots can change rapidly during play. Teams can employ for instance a start-, defensive-, and an offensive strategy. When the ball moves forward, all robots will change their base positions more towards the front. The following list illustrates some possible ways of dealing with distributed control:

1. Appoint one of the robots as a captain, and have the captain inform all other teammates with specific instructions. The disadvantage of this approach is that it is difficult to ensure that the captain will reach all other robots.
2. Make the decision to change strategies dependant on globally known variables, such as the score⁸. All robots will autonomously reason that they will switch strategies. The preprogrammed strategies can explicitly state which robots should fulfill which task, and at what position. This has as its disadvantage that it is predictable, but it does not require any communication.
3. When a team has a preprogrammed strategy (positioning) change when a teammate has the ball and is moving towards the opponents side, a passive robot that observes the teammate moving towards the front with the ball can 'broadcast' the strategy change, ordering all other robots in the vicinity to move towards the front.

⁷This is the equivalent of a substitution in human soccer

⁸For example, when your team loses by more than 10 points, switch to a defensive strategy

7.4.4 Adaptive Behavior

This section introduces adaptive team behavior and individual adaptive behavior.

7.4.4.1 Adaptive Team Behavior

The previous section illustrated that both the roles (behaviors) and the positions of agents can be changed during a game. There is a distinction between short-term and long-term adaptive behavior. The first deals with, for instance, when a teammate with the ball moves forward to start an attack. When the other robots adapt their strategy (base position) in accordance with the 'attack' mode, this is beneficial for the team for the following reasons:

1. There will be more players in the front of the field, increasing the possibility of success.
2. The robot can anticipate that there will be other robots on / around certain new base positions. Although this assumption is a big one, it will know for sure that there will be teammates trying to move towards certain predefined locations. This can be useful for, for instance, giving through passes beyond the field of vision.

There are several means to establish a strategy change, and there is a distinction between long-term and short-term strategy change, where the most important difference is that the role allocation will only change for long-term strategy changes. This is illustrated in Table 7.1.

To establish the long-term changes, using a captain or using the global variables approach that was outlined in the previous section is feasible. For the short term changes, they are not. This problem can be overcome by defining a social law where for given observations⁹ the observer is obliged to broadcast the strategy change. A problem with this approach is that multiple agents could reason simultaneously that the strategy should change, and all would shout and receive the speed penalty. An optional solution is that the defensive players - that are less important in the front of the field - will broadcast the strategy change upon the observation. The offensive robots will then be able to move forward rapidly. The optimal strategy change mechanism will depend on the situation, and only an empirical approach will determine which is the best.

⁹For instance, while using the short-term defensive strategy, an observation of a teammate with the ball

Table 7.1: Adaptive Team Behavior

Long / Short term	Role and / or Positioning	Example
Long term	Role	When losing by more than 10 goals, change an attacker to a defender.
Long term	Role and positioning	When losing by more than 20 goals, use a defensive formation and change more robots to defensive mode.
Short term	Positioning	When a teammate has the ball and is moving forward, change the strategy to offensive, and have all robots take a new (more offensive) base position.
Short term	Positioning	When the opponent is making an attack, switch to defensive more with more robots near the goal area.

7.4.4.2 Individual Adaptive Behavior

The team behavior can be altered throughout the game, but also the individual behavior of the agents can be made adaptive. The learning agent that was introduced in Figure 2.2 could serve as a model to establish adaptive behavior. Individual learning in a team faces several problems however:

1. The performance measure of an agent is the score of the team (compared to the score of the opponent). As such, it is not possible to measure the influence of a certain modification on the performance.
2. Teammates will not be able to predict each others behavior, social laws may be broken.
3. Improved¹⁰ methods of dealing with certain observations will only be used by a single member of the team, rather than by all.

The first argument makes it prohibitively difficult to have agents learn optimal behavior from scratch. What can be learned however is the optimal value of certain parameters - while playing against the same opponent. There are many parameters in the execution of tasks (skills), and in the inference of knowledge from the environment, that could be modified to have a significant influence on the team performance.

¹⁰Assuming that there is some way of determining the effect of a modification

In order to learn the 'optimal' value for certain actions, the agent requires a performance measure. When the action deals with scoring, the performance measure is trivial. When the action deals with, for instance, a through pass, the performance measure is nontrivial. A possible approach for dealing with this is the following:

1. Store all actions, the time at which they were performed, and all percepts, as 'knowledge-base'.
2. Whenever communicating a message, send your knowledge-base.
3. Upon receiving a message, merge your knowledge-base with the knowledge-base that was received¹¹.
4. When the information between the last kick-off and the last goal¹² is complete¹³, evaluate the performance in the following way:
 - (a) Keep a mapping from actions to desired outcomes. For instance a through pass has as its desired outcome that a teammate catches the ball.
 - (b) Look at the point in time that the action (through pass) was made. Was the desired outcome achieved? (ball received?).
 - (c) When the desired outcome was achieved, this can still not be attributed entirely to the modification of the variable. Keep a list for each action that was performed, and the success percentage for a given value.
 - (d) Using this list, determine statistically which value yields a higher chance of success.

This empirical approach to performance evaluation has not been tested in the simulator¹⁴ but it is expected to yield interesting results. An other essential ingredient for the learning agent is the problem generator. This could be triggered when the performance of a particular action falls below a certain threshold, or triggered randomly in any other case¹⁵. The conclusions regarding the performance evaluation of certain variables can be merged between the agents to steepen the learning curve. The agents could even negotiate that certain agents would attempt very high values,

¹¹The knowledge base consists of the position of all objects that were visible at any point in time.

¹²This is one period; Although a match runs for a certain amount of time, the game effectively restarts after every goal.

¹³Complete means that all teammates have merged their information

¹⁴The mechanism was implemented, but has not been thoroughly tested yet

¹⁵The problem generator has to see if there are better ways of performing the task, even if the current approach is functional

and others very low, and the resulting performance evaluation will conclude which value works better. In particular for very long simulations, this approach will adapt the behavior of individual players, and in term improve the performance of the team.

The pseudo-code of Listing 7.22 illustrates the learning mechanism that has been described in this section:

Listing 7.22: The proposed learning mechanism

```

1 //we have a complete knowledge-base prior to running
  this method
2 learn ()
3 {
4   for all actions in last period {
5     performance = evaluatePerformance(action)
6     evaluation_results.add(action , performance ,
7       current_value , time.now)
8   }
9   for all evaluation results {
10    if ( evaluation_result.size >
11      statistical_threshold &
12      evaluation_result < performance_threshold )
13      changeValue(evaluation_result.action)
14    else if random() > 0.98 // 2% chance
15      changeValue(evaluation_result.action)
16  }
17 }

```

The empirical evidence regarding the optimal values for certain actions is gathered by playing against the same team. Since the opposing team can be expected to show adaptive behavior, and tournaments will feature multiple opponents, the agents should have some way of storing their knowledge.

```

1 knowledge = list of { action , performance , value , time
2   } pairs
3
4 if (END)
5   store(knowledge)
6 if (BEGIN)
7   load(knowledge)

```

This list is identical to the list that was created in the previous algorithm. The time aspect is important, because when the opponent team would adapt itself, more recent values in the list are more relevant. Also, when the

actions yield different performance using the same values on different teams, a weight can be added to the time, to ensure that values that recently caused good performance are preferred over older values.

7.4.5 Rule-based Action Selection

The reasoning step between the high-level knowledge inferred from the environment and the high-level player skills is the most important in determining the player behavior. Reasoning is implemented using rule-based reasoning, to include human expertise about the domain. The rule based action selection mechanism decides based on the observations from the environment what action should be done. Since the environment of soccer is dynamic, the fitness of the current plan should constantly be reevaluated. The rule base will reason for every single visibility update what plan is best, given the current observations. The pseudo-code in Listing 7.23 is a simple rule base determining what the agent should do. A more detailed example of a rule base is provided in the next chapter.

Listing 7.23: A Simple Rule-based Action Selection Mechanism

```
1 if ball_known
2   if ball_kickable
3     if near_goal
4       if turned_towards_enemy_goal
5         kick
6       else
7         turn towards goal
8     else
9       move towards goal
10  else
11    move towards ball
12 else
13   scout for ball
```

7.4.6 A Multi-Agent Approach to Strategy

The soccer simulator and the strategies that have been discussed so far are technically *multi-agent systems*, as there are multiple agents involved in solving a problem that is difficult to achieve by an individual agent.

On a smaller level, the players are all modeled by single individual agents, dealing with solving a difficult problem. This problem could easily be split up in a number of different modules - and corresponding agents. This multi-agent system would be involved in handling sensors, reasoning, evaluating,

and acting. An example MAS to replace the player agent would consist of the following agents.

- Sensor Agent, to receive the sensors, and parse some low-level information.
- Knowledge Base Agent, to receive the newly updated information and store all (relevant) information from the past.
- Environment Analysis Agent, that uses the inference rules to derive all high-level information about the current and past environment.
- Reasoning Agent, to reason about the high-level percepts, and transform these in high-level action commands.
- Strategy Evaluation Agent, to evaluate the current strategy's performance and decide whether to adjust it.
- Action Agent, to transform the high-level action commands in the low-level actions that are possible.

The implementation of such a proposed MAS is relatively simple. The agent that is mentioned in the team XML will be loaded by the framework. If this agent would load the other required agents

7.5 Summary

The agent strategies described in this chapter explore some possibilities of the soccer simulator as a tool for AI research. It described how an agent can map its percepts to the basic action commands that are available. This is done using three steps; (1) a number of inference rules to determine high-level information about recent observations, (2) a rule-based action selection mechanism that uses the high-level information and returns high-level action commands, and (3) a set of low-, medium-, and high-level player skills that map the high-level action commands to the basic action commands of the simulator. A distinction was made between passive and active players, and the importance of social laws in reducing the amount of communication at critical moments during the game was described. The importance of the formation - based on individual robot positioning and role allocation - was described, and several approaches towards dynamically adapting these were discussed. A distinction was made between long-term and short-term strategy changes. The implications of the distributed control in the simulator with regards to the implementation of strategy change mechanisms was described and several approaches were suggested. The difficulty of finding a good performance measure for determining the influence of a single

parameter on the game was described, and a solution was proposed. The solution uses a mapping from actions to desired outcomes, and analyzes the joint history of all teammates to determine the actual performance. This solution allows robots to learn the optimal values for the execution of their actions.

Chapter 8

Development of a Team

"None of us is as smart as all of us." Japanese proverb.

8.1 Introduction

This chapter describes the development of a team¹ in the soccer simulator. The team was designed for usage in the MKT-2 project. The project requires groups of students to develop a team that is capable of convincingly defeating the reference team. As it was the first year that the assignment was introduced, and there were only 3 weeks available for the assignment, and the participating students were first-year undergraduates, the complexity of the team should be carefully considered. Another important requirement for the reference team is that it should not use any form of communication. The implementation should be of such complexity that students will be able to defeat the reference team through cooperation and communication, to illustrate the importance of these elements in successful teams.

The first section describes the design process of the SimpleTeam. First, the roles in the team are defined, then the formation is determined. Subsequently, the skills and inference rules that were going to be used were selected. The next part discusses the design of the rule-based action selection mechanism. The next section discusses a more advanced team. The final section of this chapter deals with the implementation details of the SimpleTeam. The template that was used is described, and certain implementation details are illustrated. The chapter concludes with a brief summary of the design and implementation of the SimpleTeam.

¹The team is called SimpleTeam, despite its reasonably complex behavior. This name was given to illustrate the possibilities of the simulator.

8.2 Design

The design of the team is in accordance with the general outline for agent strategies that was described in Chapter 7. The individual agents will separately reason each visibility update about the action that should be taken given the current environment. This section begins by describing the roles in the team, followed by the player skills and inference rules that were used. The next section describes the rulebase that is used for the action selection mechanism.

8.2.1 Roles in the Team

The design and performance of the team depends for a large part on the role allocation to the individual robots. A distinction that can be made is that of the goalkeeper and the other players. Although all robots are physically identical, one of the robots will always guard the goal and behave different from the regular players. Further specifications of the behavior can be made, such as a division between offensive and defensive players, and even left- or right- wing players, but these would unnecessarily complicate the reference team. A *KeeperPlayer* and a *SimplePlayer* were included in the design. When this would not result in sufficient complexity after the implementation, more players (defensive / offensive) would be included.

8.2.2 Formation

The positions of the players in the team is another important aspect of the design. In order to keep the complexity low, there is no adaptive behavior included, and there is only one formation for the SimpleTeam. The formation is illustrated in Figure 8.1. The formation is 1-2-3-1. It intentionally keeps a small gap in between the two defenders. If there would be a solid defense line, most attacks would occur over the flanks. Through providing an alternative route to the goal, users can be stimulated to consider center attacks, and make well-performing implementations of the avoid opponent action.

8.2.3 Player Skills

A number of low-, medium-, and high-level player skills are described in Chapter 7. The player skills that will be implemented in the SimplePlayer are the following:

1. Low-level
 - (a) Turn the robot towards a point.
 - (b) Scout the area.



Figure 8.1: The SimpleTeam Formation

2. Intermediate-level

- (a) Turn the robot towards an object.
- (b) Track the ball.
- (c) Move to a position.

3. High-level

- (a) Clearing the ball.
- (b) Avoid opponent.
- (c) Move towards opponent goal.
- (d) Passing the ball.

8.2.4 Inference

The following functions were used for the SimplePlayer to transform the low-level percepts towards high-level information about the environment, that was subsequently used for the rule-based action selection. The last two functions were not described in the previous chapter, as their implementation is trivial.

1. Determine if the Ball Position is known.
2. Determine if the Ball is Visible.
3. Determine if the Ball is Kickable.
4. Determine whether the Ball is Free.
5. Determine if there is Chaos around the ball.
6. Determine whether a teammate has the ball.
7. Determine whether an opponent is up ahead.
8. Determine whether there is a teammate standing free.
9. Determine whether the ball is in the enemy goal area.
10. Determine whether the ball is near the base position.

8.2.5 Rule-based Action Selection

The rule-based action selection takes as its input the high-level information derived from the percepts, and determines the best action given the current observation. Listing 8.1 illustrates the reasoning mechanism of the SimplePlayer. Instead of executing the actions straight away, the resulting low-level actions are stored in an intermediate MoveTuple. When multiple actions 'fire', the last action that fires will actually be executed.

Listing 8.1: SimpleTeam's Rule-base Action Selection Mechanism

```

1 MoveTuple move
2
3 if (ball_position_known)
4 /** always track ball, except if we have it and want
5 to kick */
6 move.turn = trackBall()
7 if (ball_visible)
8 if (ball_kickable)
9 /** we have the ball: If enemies are coming,
10 dodge or pass to teammate. Otherwise, move
11 for goal! */
12 move.position = moveToGoal()
13 move.turn = turnToPosition(goal)
14 if(ball_in_enemy_goal_area)
15 if (turned_towards_opposing_goal)
16 move.special = kick
17 else if(team_mate_free)

```

```

15     move.special = kick
16     else if(chaos & turned towards opposing goal)
17         move.special = kick
18     else if(enemy_up_ahead)
19         move.position = avoidOpponent()
20         move.turn = turnToPosition(move.position)
21     else if(ball_with_teammate)
22         move.position = moveFree()
23         move.turn = turnToPosition(move.position)
24     else if(not chaos)
25         move.position = ball.position
26     else if (ball_near_base_position)
27         if (ball_free and not chaos)
28             move.position = ball.position
29         else
30             move.position = ball.position
31     else
32         move.position = base_position
33     else
34         move.position = base_position
35         move.turn = scoutArea()
36
37 moveToGoal(move)

```

8.3 Implementation

The implementation of the SimpleTeam started with a basic skeleton that was extended to merge the known information in the knowledge base, to infer the high-level information about the knowledge base, and to reason about and execute the actions. The implementation of the SimplePlayer is discussed. The KeeperPlayer is structured in a similar manner, but the actual action selection mechanism is different, and several goal tending skills have been included.

8.3.1 Template

The code in Listing 8.2 is the template that was used and extended for the implementation of the SimpleTeam.

Listing 8.2: Code template for SimplePlayer

```

1 @Publishes({ "Robot.Move", "Robot.Shout", "Robot.
   Changebehavior" })

```

```
2 @Subscribes({"Robot.Init", "Robot.Parameters", "Robot.
   Status",
3 "Robot.See", "Robot.Listen"})
4
5 public class SimplePlayer
6     extends Agent
7 {
8     /**
9      * some 'static' information about game rules,
10      * status and our own info.
11     */
12     private PlayerInfo fPlayerInfo;
13     private TournamentParameters fParameters;
14     private TournamentStatus fStatus;
15
16     /**
17      * This method is called after we are loaded,
18      * with a PlayerInfo containing (amongst others)
19      * our BasePosition.
20      * @param pInfo
21     */
22     public void handleInit(PlayerInfo pInfo)
23     {
24         fPlayerInfo = pInfo;
25         fKnownInfo = new HashMap<String, RobotInfo>();
26         fRobotNames = new HashSet<String>();
27     }
28
29     public void handleParameters(TournamentParameters
30     pParameters)
31     {
32         fParameters = pParameters;
33     }
34
35     public void handleStatus(TournamentStatus pStatus)
36     {
37         fStatus = pStatus;
38     }
39
40     public void handleListen(MessageTuple pMessage)
41     {
42         // SimplePlayer does not use any communication!
43     }
44 }
```

```

42  /**
43   * This method is called every view-update
44   * milliseconds
45   * supplying us with the latest information.
46   * determine strategy according to what we see
47   * here
48   */
49  public void handleSee(List<VisibleObject> pVisible
50  )
51  {
52   updateKnownInfo(pVisible);
53   analyzeSituation();
54   doStrategy();
55 }

```

8.3.2 UpdateKnownInfo

The UpdateKnownInfo method will merge the information that is observed with the information that was already known.

Listing 8.3: Update the known info

```

1  /**
2  * will update the list of known info
3  */
4  public void updateKnownInfo(List<VisibleObject>
5  pVisible)
6  {
7
8  /** The first object in the list is always the own
9  robot */
10 fLastPosition = (RobotObject) pVisible.get(0);
11 long lTime = System.currentTimeMillis();
12 /** skip ourself -> start at 1*/
13 for (int i = 1; i < pVisible.size(); i++) {
14 VisibleObject lObj = pVisible.get(i);
15 RobotInfo lInfo = fKnownInfo.get(lObj);
16 if (lInfo != null) {
17 lInfo.object = lObj;
18 lInfo.time = lTime;
19 } else {
20 RobotInfo lNewInfo = new RobotInfo();
21 lNewInfo.object = lObj;

```

```

21     lNewInfo.time = lTime;
22     fKnownInfo.put(lObj.getName(), lNewInfo);
23     if ( !lObj.getName().equals(fParameters.
24         getBallDescription()) )
25         fRobotNames.add(lObj.getName());
26     }
27 }
28
29 /**
30  * wrapper class for information about a particular
31  * robot.
32  * Will now only store the latest VisibleObject and
33  * time at
34  * which it was last seen, but can be extended to keep
35  * track of
36  * everything we know about this player!
37 */
38 class RobotInfo
39 {
40     VisibleObject object; /** the last known
41         visibleobject of this robot */
42     long time; /** last seen at $time */
43 }

```

8.3.3 AnalyzeSituation

The AnalyzeSituation method will derive the high-level information from the environment. The implementation of the individual methods is described in pseudo-code in Chapter 7. The actual implementation is a compact concatenation of all different methods.

8.3.4 DoStrategy

The DoStrategy method is responsible for selecting the actions that will be executed, and executing these. The implementation of this method is very much like the pseudo-code action selection mechanism described in Section 8.2.5.

8.4 Advanced Team

The team that was discussed in previous discussions exhibits reasonably complex behavior without communication using a simple set of rules. Chap-

ter 7 has described a number of more advanced techniques. The characteristics of such an advanced team are discussed below:

- Use multiple agents, each with their own task, to execute the player tasks. An example of this is given in Section 7.4.6.
- Keep a number of behavior patterns. Have one of the MAS agents analyze the behaviors of the individual opponent robots, and find the best matching behavior. Use this approximated behavior in predictions of opponent robot behavior.
- Implement and integrate the self-learning behavior explained in Section 7.4.4.2 in the MAS. An important prerequisite for this is the design and testing of the functions that determine the performance measure of the action. An intrinsic element of this self-learning behavior is the short-, medium-, and long-term storage of (relevant) information in the knowledge base. The *short-term* information is of particular importance for determining how to execute a strategy. *Medium-term* can reflect on the performance of a strategy and adapt this if necessary, whereas *long-term* allows the optimization of the variables that determine how actions are carried out.
- Design and implement a model that determines when a player is considered active or passive, and includes as many social laws as possible. The model should also determine exactly when to communicate, bearing in mind the speed penalty².

8.5 Summary

This chapter has described the design and implementation of the SimpleTeam. The requirements of the SimpleTeam state that groups of first-year students should be able to defeat it after a three week assignment, and that the team does not use any form of communication. In accordance with these requirements, the team was designed. First the different roles within the team were decided on, and the formation was defined. The different skills, inference rules and the rule-based action selection mechanism were defined. The implementation of the SimplePlayer features three methods that together start with perceptual input, and end with basic action commands. A number of potential elements of an advanced team have been described. The design and implementation of such a team is expected to yield interesting results.

²For instance, shouting position information of robots that you know the targeted recipient will already have observed is spreading redundant information and not worth it

Part IV
Education

Chapter 9

MKT-2 Project

More important than the curriculum is the question of the methods of teaching and the spirit in which the teaching is given,
Bertrand Russell.

The work on this graduation project was initiated by the demand for a new assignment on multi-agent systems for the MKT-2 project. This chapter introduces the project, and reflects on flaws in previous assignments, to prevent these from occurring in the new assignment. The first section describes the educational goals of the project. The next section briefly introduces the other programming assignments that are a part of the project, and provides a detailed discussion on the two previous ad-hoc network assignments. This chapter concludes with the setting of the project in 2005-2006, the year that the multi-agent soccer assignment was introduced.

The educational approach of the MKT-2 project is very different from conventional practical assignments and projects. An in-depth description of the history and background of the project is provided in Appendix C. The next chapter discusses the soccer assignment.

9.1 Educational Goals

The MKT-2 project is a mandatory project for first-year undergraduate Media & Knowledge Engineering (MKT)¹ students at Delft University of Technology. The project is their first introduction to artificial intelligence, and has the following two aims [7]:

1. To introduce the basic concepts of knowledge engineering and the relevant AI techniques, including search algorithms, knowledge representation techniques, rule-based reasoning algorithms, and agent technology.

¹This is a variant of computer science

2. To explain and instruct on issues related to AI programming in general and intelligent (multi-) agent applications in particular.

9.1.1 Approach

In order to achieve the aforementioned goals, several educational approaches exist. The conventional approach is based on the principles of behavioral psychology. This approach is frequently employed in today's higher education. The knowledge that is transferred from the teacher to the student is passively accepted. This approach alienates many students, reducing motivation and performance. Other approaches use cognitive psychology, or constructivism.

"Constructivism proposes that knowledge or meaning is not fixed for an object, but rather is constructed by individuals through their experience of that object in a particular context." [69]

An example of a constructivist, vague problem definition would be "*Design and implement an intelligent system*". An example objectivist assignment would be "*Implement the A* algorithm, and calculate the route from Amsterdam to Berlin*". Using a pure constructivist approach for first-year students would however face several problems, as they lack the design and implementation experience that is required to adequately deal with vague problem definitions.

Since neither method is suitable for this project, a synthesis between objectivist and constructivist approaches is applied. Guidelines towards combining these approaches have been proposed in the literature [70] [71] [72] [73] [74]. These guidelines have been considered while designing the assignments. Throughout the years, this approach has been thoroughly evaluated and refined.

9.2 MKT-2 Assignments

There are 4 assignments; A,B,C and D. Together these assignments fulfill the educational goals described in the previous section. This section begins by briefly describing assignments A,B and C. Subsequently, it shall describe the educational goal of D, describe the two previous assignments that were developed, and discuss their strengths and weaknesses. This section concludes with the setting of the MKT-2 project in the year 2005-2006, the year that the soccer assignment was introduced.

All group assignments require that an analysis, design and implementation report are produced. A data flow diagram illustrating the general structure

of the system they will develop - to ensure a thorough understanding of the assignment, and two Gantt[75] charts - to ensure a fair labor division - have to be produced.

9.2.1 Assignment A: Roshambo

The first assignment is an individual assignment where a Roshambo (Rock-Paper-Scissors)[76] agent has to be programmed that defeats the reference agents. The goal of the assignment is to introduce the students to the principles of agents, verify that the programming level of the students is adequate, and learn students to design and implement an "intelligent" algorithm. The reference agents each have their own strategy for deciding whether to choose Rock (R), Paper (P) or Scissors (S). These strategies vary in difficulty from simply repeating R-P-S, to a pattern-recognition mechanism that detects the longest pattern match, and anticipates the opponent's move based on this. The students have to defeat all but the most difficult and the random player convincingly² in order to pass the assignment. It is an individual assignment, and will take some students a couple of hours, whereas the less experienced programmers may need several days. Students that pass the assignment have shown to be capable of programming Java at a reasonable level. All further assignments are in groups of 5-6 students. The game is illustrated in Figure 9.1.

²After 100 games, the player has to have won at least 30 games more than the opponent.



Figure 9.1: Assignment A: Rock, Paper, Scissors

9.2.2 Assignment B: Rule-based Reasoning

The second assignment is a group assignment that introduces students to rule-based reasoning. In the assignments, students create a multi-agent system that translates a questionnaire³, filled out by all participating students, into a ranked list depicting the suitability of each participant for being a part of your team. Several agents have to be developed; one agent parses the survey data, another uses this data to generate the new facts using forward chaining [77]. Such facts would be characteristics derived from the data, such as *leader*, *nerd*, *arrogant*. Then, a score agent that assigns a score to these characteristics has to be built. Finally, the 'Lister agent' has to sort the scores provided by the score agent. An overview of these agents is provided in Figure 9.2.

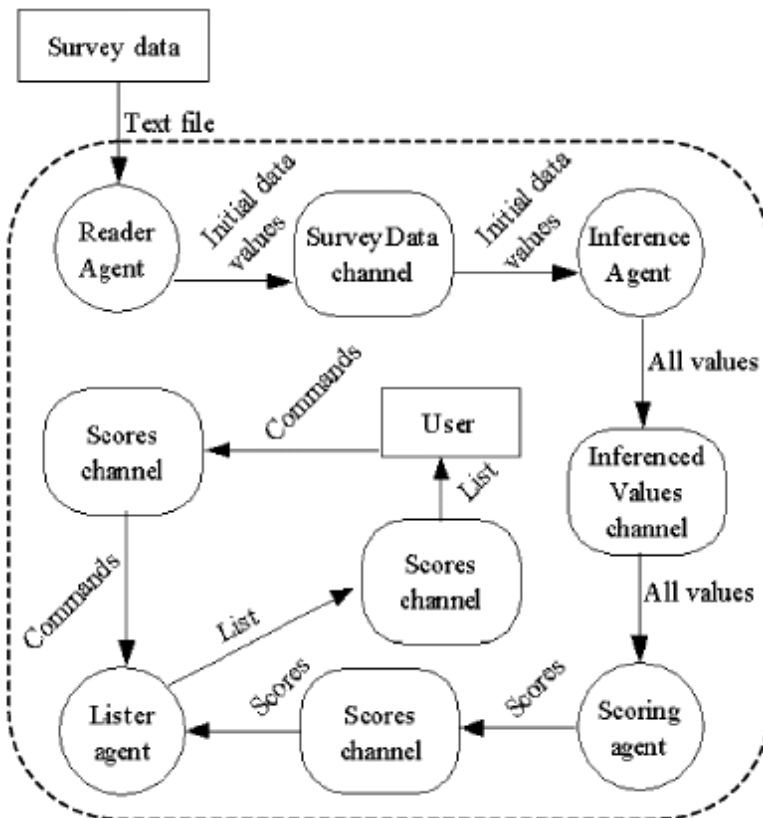


Figure 9.2: Assignment B: Rule-based Reasoning

³Some example questions of this questionnaire are *Do you often take the initiative in a project team?*, *How old are you?*, *How often do you read books?*

9.2.3 Assignment C: Hollywood

In the third assignments, the IMDB [78] database is used and transformed to a semantic network. This semantic network is made up of nodes (actors, actresses, movies, directors), and connections (played in, directed). This is illustrated in Figure 9.3. The students have write an agent that builds the semantic network, and an agent that determines the (minimum) amount of "degrees" a person X is from person Y. Then, an agent has to be built that determines the entire "Hollywood" network, by taking a seed and propagating several degrees. Finally, the group's ultimate hero is determined by designing and implementing an intelligent algorithm that assigns points to certain activities, such as directing or acting, but also through including the "rating" of the people this person has worked with. The assignment is algorithmically very challenging, and works with real data, which provides a strongly motivates the students.

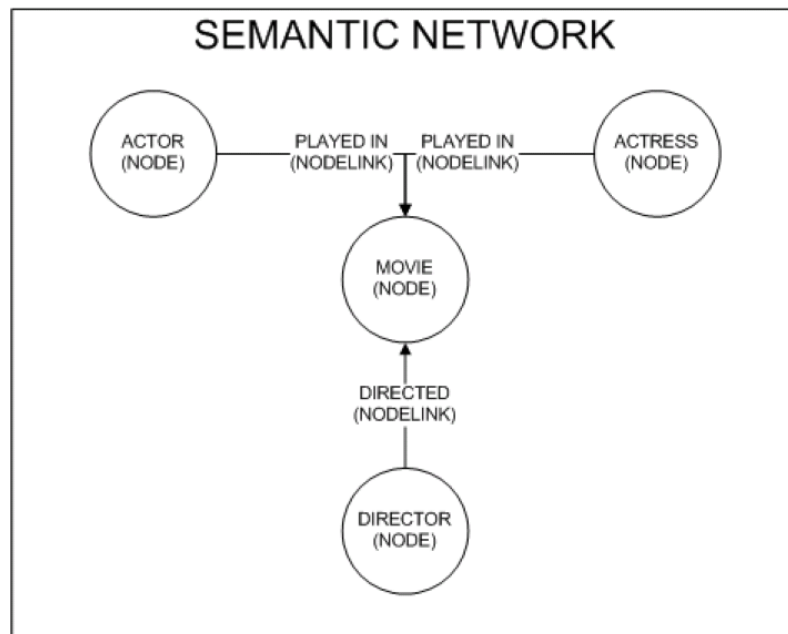


Figure 9.3: Assignment C: Hollywood

9.2.4 Assignment D: Multi-Agent Systems

Assignment D is the largest assignment of the project. It aims to introduce multi-agent systems, ad-hoc networks, and distributed problem solving to the students in a useful way. In both assignments, the trade-off between autonomous reasoning and communication was present, with the intention of making students aware of this problem and consciously designing an intelligent algorithms that combines both.

Development on assignment D started in 2003-2004, but this was not very successful due to software and conceptual errors, as shall be seen in the first part. The second part introduces the assignment as it was introduced in 2004-2005, which was an improvement, but still faced conceptual problems. This section concludes with a description of the problems that were encountered in both assignments.

9.2.4.1 2003-2004: Peer-to-peer Networks

The 2003-2004 assignment dealt with peer-to-peer networks. Every node in the network had a certain probability of having a connection with any other node, with a given (random) latency, bandwidth, for a certain duration. An example network is given in Figure 9.4. Nodes have audio files hosted on them, and a search agent would be loaded on a random node with a list of desired audio files. The goal was to get all desired audio files to this node in as little time as possible. Agents could be sent over the connections, with for instance the list of desired tracks. Although the concept of the assignment was found to be challenging, the mobility that was required for simulating the nodes, and moving agents between nodes, did not function yet in Fleeble. Another issue was the concept of the assignment. Random connections, with random latency and bandwidth bear no relation to reality. The constraint that connections are often alive for a very short amount of time made it a surrealistic environment. Although latency and bandwidth may vary between certain connections, this is often due to the distance to an access point, the network traffic, or local restrictions. Random connections that will live only 5-20 seconds are highly improbable as well. With no relation to the reality, the validity of the problem and plausibility of the existence of an "intelligent solution" become questionable.

9.2.4.2 2004-2005: SMS Assignment

Building on the lessons learned from the year before, in 2004-2005, a new assignment D was introduced in an effort to teach students about MAS using a metaphor that was more realistic. Random SMS messages were initiated on random nodes, with random targets. All nodes were moving randomly across the screen, with a certain broadcasting range around them. They

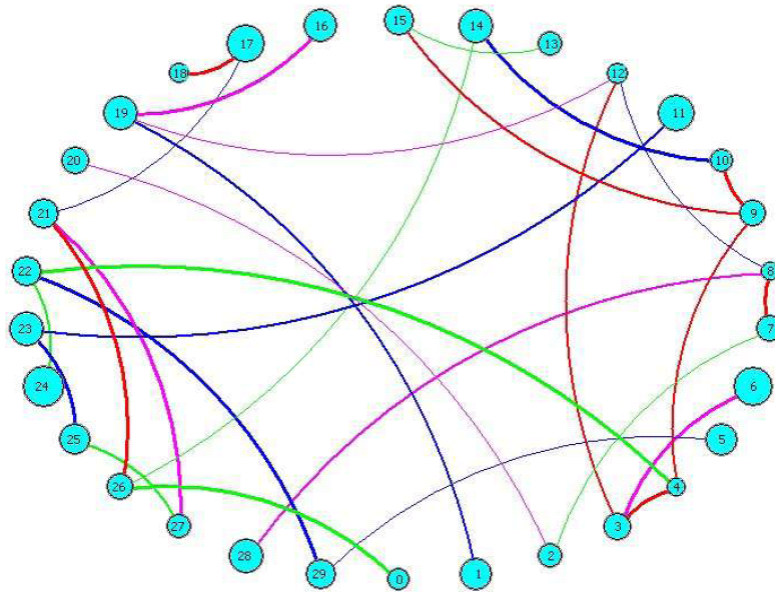


Figure 9.4: Assignment D 2003-2004: Peer-to-peer networks

were able to see who was in this range, and at what position. This is illustrated in Figure 9.4. Through sending messages across channels, the agents on the nodes were able to communicate position information. Communication costs a certain number of points, whereas point rewards were given for quick delivery of the SMS. With a relatively large number of SMS operations, sending a message and including all positional information that is available, will have a high chance of helping other SMS operations succeed faster. As such, the goal was to get cooperative intelligent solutions where the cost for an agent to share information was outweighed by the potential benefit of other agents gaining this new information. The main problem with the assignment was the artificial nature of the point distribution. Because of this, very simple strategies like always broadcasting (cooperating), or never broadcasting unless next to the goal position (defecting), would prevail. Attempts at more intelligent solutions yielded too high broadcasting penalties to be effective. Changing the point distribution to generate a lower penalty for broadcasting would allow simple always-broadcast strategies to become successful. A possible solution to the problem would have been to create one 'optimal' strategy, and set the threshold such that this strategy would get the highest score. As there is only a limited amount of time for implementation, this strategy would have to be pre-specified, and the assignment would not be as challenging or motivating. The perceived freedom in designing an approach, combined with group brainstorming sessions are

important ingredients for the learning experience of the students.

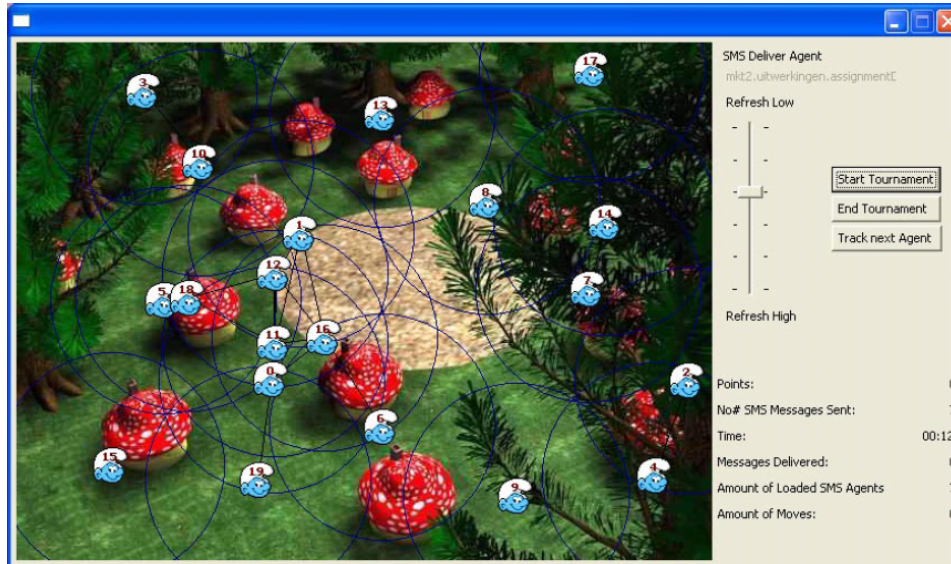


Figure 9.5: Assignment D 2004-2005: SMS Assignment

9.2.4.3 Discussion

The two previous ad-hoc networking assignments have shown that not only software problems, but also conceptual flaws in assignments have a large impact on how intelligent solutions will be, and on the skills acquired by the students. Both assignments faced the problem of being too artificial. The problems that were handled in both assignments were artificially created for teaching ad-hoc networks. As a result of this, the performance measures that were used in the assignments were also artificial. An arbitrary point distribution for rewarding and punishing certain types of behavior was used, causing the 'intelligent' solutions that solved the general problem to lose to very simple strategies. This is also described in Section 4.6.1. Another problem that was encountered in the design of many groups, was that they would attempt to make the network deterministic through intense communications - allowing agents to reason about complete information, which essentially simplifies the problem such that it will be solvable using conventional brute-force methods, rather than finding a synthesis between autonomous reasoning and communication.

From this experience, the most important lessons that were learned are: Do not create an artificial problem with an artificial point distribution. Use

actual problems, simplify these, and put clear goals with natural rewards and penalties

9.3 Setting

In the year 2005-2006, the MKT-2 project started with 34 first-year undergraduate students. 27 of these finished the project with a passing grade. The 7 that did not were either removed from the project during the individual (first) assignment, or throughout the project. This resulted in 5 groups of 5-6 students.

The undergraduate curriculum at Delft University of Technology was changed significantly in 2005-2006. Because of this, the entry level that was anticipated by the teachers - based on the level of the students from the old curriculum - was different. The students had slightly different background knowledge and programming capabilities⁴. This explains the relatively high number of students that did not meet the requirements set out in the first assignment.

In their introductory Java courses, the students acquired some basic Java skills, but not on the specific features of Java 1.5 (See Section 3.3.4). The students had gained some experience with group work during the first MKT project, but not much.

The MKT-2 project was held in the fourth quarter, from early April until early June. There were 8 scheduled weeks, and one week May holiday in between. There were biweekly afternoon sessions scheduled, lasting from 1 PM until 5 PM. In these sessions, the teachers were actively involved in helping students resolving bugs in their code, answering questions about the design and implementation of the assignments. The project rooms featured large tables that could be used for group meetings and brainstorming, and a number of computers for programming and debugging.

The last assignment of the project - the ad-hoc networking assignment, was scheduled to last 4 weeks, but because the Hollywood assignment took a little more time, there were 3,5 weeks available for the soccer assignment. The soccer simulator was finished when the assignment started, but several small software bugs were encountered and fixed in the first week. The assignment and the parameters of the game did not change, and the students were told that they had to anticipate the implementation that was described

⁴Several students blamed the high workload on students in their first year of the new curriculum for not having had enough time to acquire adequate programming skills.

in the manual, and that the final implementation would work according to those specifications. The bugs that were reported were usually resolved the same afternoon or the day after.

SimpleTeam, the reference team that had to be defeated, was finished and given to the students after the first week of the assignment. The source code of this team was not distributed. The RandomTeam⁵ was included in the first release, and the source code was available to serve as an example for the students.

⁵Players in the random team will all chase the ball when they see it, and kick it when they have it. When they don't see the ball, they move across the field randomly.

Chapter 10

Soccer Assignment

”The whole art of teaching is only the art of awakening the natural curiosity of young minds for the purpose of satisfying it afterwards”, Anatole France.

This graduation project started in response to the call for a challenging programming assignment that would replace the existing mobile ad-hoc networking assignment in the MKT-2 project (See Section 9.2.4). During the design and implementation of the model, soccer simulator and reference teams, the educational goal was always considered.

This chapter describes the design, use, and evaluation of the soccer assignment that replaced the previous ad-hoc networking assignment. The first section describes the educational goals that the assignment intends to accomplish. Then, the actual assignment is described. The next section describes how the assignment was evaluated and what the evaluation results were. This chapter concludes with a discussion about potential alternative educational applications of the soccer simulator and reference teams.

10.1 Educational Goals

The educational goals of the soccer assignment are the following:

- Teach students to work in groups.
- Teach students to develop an implementation from a *vague, high-level* problem description, including:
 - analyzing the problem
 - designing a solution
 - identifying and distributing tasks between group members
 - implementing the individual tasks; learning about the difficulties of integrating several parts of a system, and learning to cope with unexpected results from their implementation.

- Teach students about:
 - distributed artificial intelligence,
 - multi-agent systems,
 - mobile ad-hoc networks,
 - decentralized control,
 - cooperative agents,
 - rule-based reasoning
- Advance the programming skill level of the students.

After having completed the assignment, students will be aware of the problems related to multi-agent soccer. They will understand the issues related to dealing with a dynamic problem environment that demands real-time performance on incomplete information, without any centralized control. They will understand the importance of cooperation and communication for achieving teamwork. They will be able to design and implement a cooperative soccer team in the soccer simulator that is capable of convincingly defeating the moderately intelligent reference implementation.

10.2 Assignment

The assignment was given to the students in Dutch, as the undergraduate program is also in Dutch. The following is a translation of the assignment text:

Introduction

Athletic and programming qualities are two completely different concepts. In this assignment we will attempt to combine these qualities. Strategic reasoning is important both for a professional athlete and for programmers. The goal of this assignment is to combine the best programming and athletic strategies to score as many goals as possible.

Assignment

In this assignment, every group is required to design a number of PlayerAgents that cooperate in order to win a match. A reference implementation shall be provided that has to be defeated by your team convincingly in order to receive a passing grade for this assignment. This reference implementation does not use any form of communication, and will display only a minimal amount of cooperation.

A team consists of 7 players. A player on the field is a RobotAgent (the "hardware") that can be controlled by a PlayerAgent (the "brains"). The PlayerAgent has certain inputs, and can execute certain commands. He can "shout", move, turn, kick, or substitute the PlayerAgent (i.e. a new behavior) on a robot. There is however no central authority in the game. The user does not have any influence on a game once it has started. The focus of this assignment is on cooperation. Controlling the RobotAgent is very intuitive. During the design phase, the approach, strategies, and the weaknesses of the chosen approach will have to be considered extensively.

After the assignment text, a detailed explanation of all input and output channels, and the three coordinate systems was provided.

10.2.1 Scenarios

The following is a translation from the Dutch manual. Prior to designing and implementing their soccer team, groups had to finish their scenarios and have these approved by the teacher.

*Following are a number of scenarios. Explain for each scenario what strategy an individual player should follow, in order to accomplish the goal of winning the match. Give a thorough description of the actions that must be taken, and use a diagram or illustration when necessary. This assignment evaluates whether the assignment and the concept of ad-hoc networks has been thoroughly understood by the group members, and whether they are ready to proceed to designing their team. The global strategy that your soccer team will use should be defined in these scenarios. Every scenario has a short description that can provide clues to possible solutions. Do not let these hints stop your imagination however. It is **strongly recommended** to analyze the behavior of the reference teams, to understand the problems and scenarios that will occur. This assignment must be checked and approved by the teacher before your group may proceed to the design phase!*

10.2.1.1 Player Level

- *Scenario 1: The ball is free*
The ball is lying somewhere on the field. This could be somewhere close to you, or close to a team mate. In the worst case scenario the ball is not close to any player of your team.
- *Scenario 2: You have the ball*
Are the opponents coming dangerously close? Can you move around freely, or are you close to your own or the opponent's goal?
- *Scenario 3: You do not have the ball*
You do not have the ball. The ball can be free, a team mate could have the ball, or an opponent could have it.

- *Scenario 4: Someone else is on your base position*
It is possible that a team mate or opponent is on your base position.
- *Scenario 5: You do not know where your team mates, opponents, or the ball are*
Possibly fellow team mates within your shouting range do know the information you would like to have, or that you have to look around.

10.2.1.2 Team Level

- *Scenario 6: Your team has the ball*
Should you defend? Should you try to stand clear of opponents?
- *Scenario 7: The opponent has the ball*
Are you in the area, or are there team mates in the area? Is the opponent in front of your goal?
- *Scenario 8: The game has just started, is half way, or is near the end.*
What do you do in each situation? Describe for each situation what strategy you will use.
- *Scenario 9: Your team is losing, winning or the score is a draw*
What do you do in each situation? Describe for each situation what strategy you will use.¹

10.2.2 Design

The scenarios that were approved in the first part of the assignment serve as a guideline for the design of the soccer teams. The following is a translation from the Dutch manual regarding the design part of the assignment:

*Your group will now have to produce a detailed design of their PlayerAgents. **All possible** scenarios have to be thoroughly explained, including the scenarios described in the first part of the assignment (Have a brainstorming session with your group about which scenarios are possible.) Make a rough sketch of the implementation. Describe how your group expects to solve the most difficult aspects of your implementation?*

Gannt [75] charts had to be made prior to implementation and after implementation that visualize the planned work load and task distribution between individual group members.

¹Scenarios 8 and 9 hint at the possibility of making a team adaptive, through evaluating your team's performance, and changing the behavior of several robots to deal with the new situation.

The design had to be approved by the teachers before teams could begin working on their implementation. Students were frequently reminded of the importance of a good design, as the problem they had to solve was big and little time was available.

10.2.3 Implementation

The most time consuming part of the assignment was the implementation. The resulting student team implementations are discussed in Section 10.3. The scenarios and design parts of the assignment did help students in understanding the problems that occur in the environment, but the challenge of the implementation is a lot bigger than any programming assignment they have previously encountered. As such, a number of questions were posed to the groups during implementation to keep them on the right track.

10.2.3.1 Implementation Questions

Following are the questions that were posed during the implementation phase of the assignment. The *HINT* is the kind of answer that was desired by the teachers.

1. *In multi-agent soccer you could decide to determine your strategy again every visibility update, and execute this accordingly. What is the disadvantage of this approach? How could you solve this? HINT: Long-term strategy, planning.*
2. *How can you prevent deadlocks? HINT: Look at the long term results of your actions. If your robot has hardly moved during the last 10 seconds while you did attempt to move, it could be that something is blocking you.*
3. *When and how should you change strategy? HINT: There have been X sequential goals by the opponents, so change? How to change? Who decides this? If you do not have a captain, how do you make sure that a particular player will know that he has to change - without changing the behavior of other players?*
4. *How can you make sure that a player is 'free'?*
5. *Most passing algorithms assume that the receiving player is standing still, and will shoot in that direction. Sometimes, the receiving player is moving and the pass will miss. How can you prevent this? HINT: Look at the orientation of the recipient. If he can see you, assume that he will reason 'intelligently'. Otherwise, reason how he will reason (He doesn't see the ball, so he is probably not doing anything highly sophisticated, so shoot the ball in front of him - if there is room for*

that. He will either see the ball coming, or receive it if he is indeed moving.

10.2.4 Extra Questions

As a reflection exercise, a number of extra questions for the final report were given. The following is the translation of these questions from the Dutch manual.

1. Explain how your players cooperate. Consider direct communication, but also indirect communication (i.e. social laws, such as "you are closer to the ball, so you get to pursue it"). Explain which scenarios include cooperation and give a general description of how this works.
2. Sometimes it is a good idea **not** to shout, whereas sometimes it can be extremely helpful. Where did your team draw the line? Motivate your decision!
3. Assume that you would have to develop a team that would defeat your own team. How would you do this? Explain this in no more than 200 words.
4. When is it useful to change the behavior of a player? If your team uses this; when do you use it, and to what extent did it help your soccer team's performance? Motivate your decisions!

10.2.5 Competition

To conclude the project with a fun event, and to provide an extra incentive to the students, a competition with a small cash prize was held among the student teams. Rather than the default 10 minute game time that was used throughout the project, 30 minute matches were used in the competition. These longer matches reduce the impact of the randomness on the score.

During the competition, multiple matches were held at the same time. The random team and simple team were included in the tournament as well.

10.2.5.1 Competition Results

Table 10.1 shows the exact results from the competition held at the 6th of June, 2006 at Delft University of Technology.

Table 10.1: Results from the 2006 MKT-2 soccer competition. The number indicates the group number, S = SimpleTeam, R = RandomTeam. Win = 3 points, Lose = 0 points, Draw = 1 point

Group #	1	2	3	4	5	S	R	Total
Group 1	x	6-0	1-4	0-10	1-4	7-5	13-3	9
Group 2	x	x	3-15	9-19	4-22	2-20	30-5	3
Group 3	x	x	x	4-9	6-3	22-9	18-7	15
Group 4	x	x	x	x	8-4	21-9	34-3	18
Group 5	x	x	x	x	x	28-5	36-2	12
Simple	x	x	x	x	x	x	32-2	6
Random	x	x	x	x	x	x	x	0

10.3 Implementations

This section describes the different approaches that were taken by the students in their implementations of the soccer teams. For each part of the assignment - scenarios, design, and implementation - the classroom observations and reports produced by the different groups are described. First a description of the general approach is given, and then the specific differences are explained.

10.3.1 Team Setup

During the scenarios and design phase, all groups recognized the importance of having different PlayerAgents (different behaviors) within a team. The general approach was to distinguish between a keeper, a defensive player and an offensive player. Teams would then be built up from 1 keeper, 3 defensive players and 3 offensive players. Some groups started and finished with a more advanced setup, whereas some others did not manage to successfully implement the different agents, and ended up with different setups. The setups that resulted after the implementation were the following:

- one toolkit of functions (skills) and variables that all agents share. 1 keeper, 3 defenders, 1 captain and 2 attackers. The captain plays a central role in determining the strategy.
- one 'super' agent that has a number of basic skills, and player-specific reasoning in the keeper, offensive, and defensive agents.
- 1 keeper, 2 defensive players, 1 mid-fielder, 3 attackers.

- 4 defensive players (one in front of the goal)², and 3 offensive players.
- 1 keeper player, 6 offensive players.

The position of these players on the field is displayed in Figure 10.1.

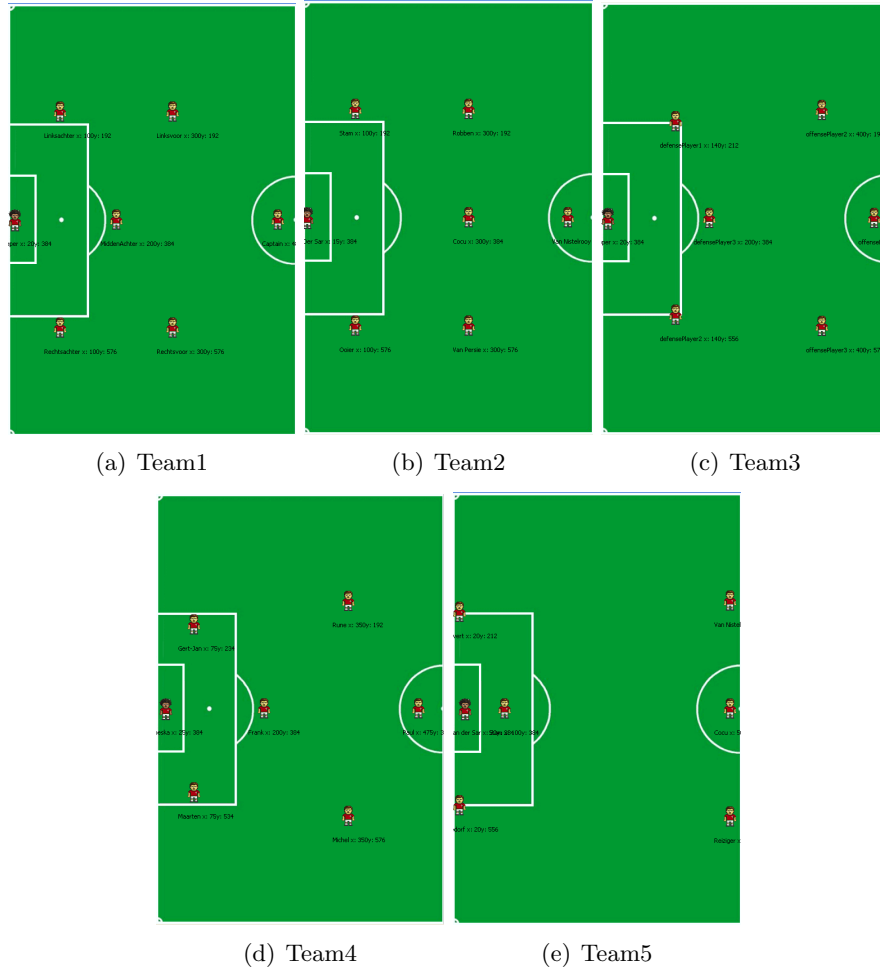


Figure 10.1: The Setups that were used by the Different Teams

²The original design of this group included a keeper, but the group struggled with the implementation. The keeper was left out, and a defensive agent replaced it.

10.3.2 Reasoning

The higher level reasoning model of the agents was very similar for all student groups. This model is illustrated in Figure 10.2. The model is similar to the simple reflex agent described in Figure 2.1, but it is more specific. The following sections will explain the elements of the model.

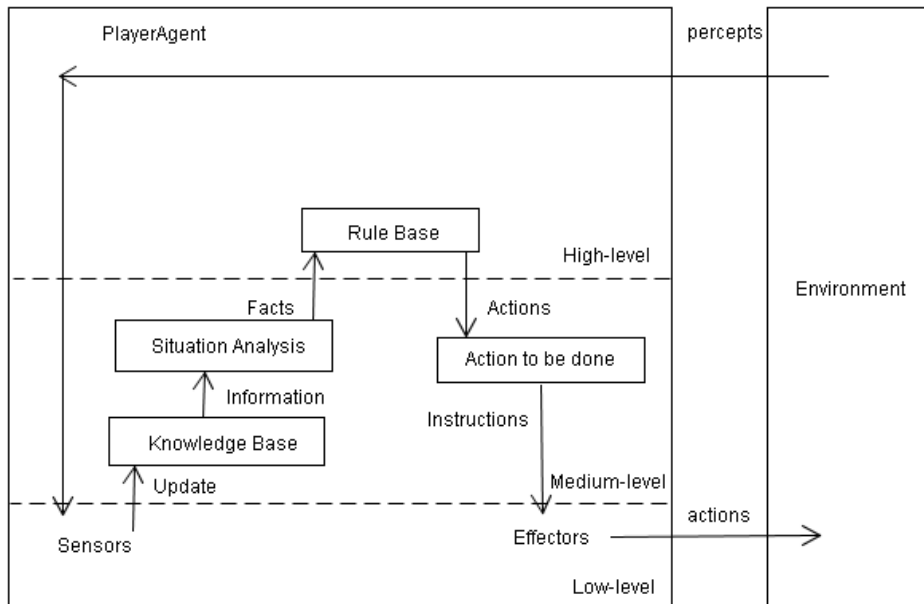


Figure 10.2: A Typical PlayerAgent

10.3.2.1 Knowledge Base

The knowledge base is the collection of facts that have been acquired from previous (low-level) observations³. All student groups keep a history of certain observations. Everything that is observed in the environment is combined with the existing knowledge in the knowledge base. Several student teams keep the time at which certain observations occurred, and use this to reason about the reliability of the information. Other groups ignore the time, and assume that the last known information is the truth for their reasoning⁴.

³Both visual and auditory observations. These observations are relatively high-level compared to those of other simulators, but in the PlayerAgent, they are the lowest level.

⁴The teams that did not keep the time for the latest observation had adjusted their reasoning such that the influence of the objects that were not directly visible was either small or not present at all.

10.3.2.2 Situation Analysis

The situation analysis module takes as its input the information from the knowledge base. This is generally all information about the objects that are currently visible, objects that were (recently) visible, and sometimes time-related information, such as the last time that a particular player or the ball was seen.

The knowledge base can be in an infinite number of different states⁵. As such, reasoning about the information contained in the knowledge base using a set of rules can not be done without establishing some higher-level facts first. Attempts to do so without these high-level facts would include too many possible scenarios for a rule base to be effectively manageable.

The situation analysis will analyze the situation (as captured in the knowledge base), and derive a number of high-level facts. These facts are generally either values or booleans⁶. An example of a value would be "the number of visible enemy robots", and an example boolean "do I see the ball". All groups defined about 10-20 of these high-level booleans to describe the current situation.

10.3.2.3 Rule Base

The rule base is the central behavior-specific part of the PlayerAgent. Two out of five teams used a superclass or a toolkit to define all behavior-invariant methods and functions, and determined the behavior (rule base) in the specific agents (i.e. defensive, offensive, keeper agents).

The rule bases that were used were all integrated in the source code, and as such the values that were required in the reasoning were simply taken from the knowledge base. The boolean facts derived in the situation analysis are the most important for determining the final strategy.

The facts that were derived, and the values that were used were specific to each team. The strategy that was defined in the rule base was also specific for each group (and to the different players within the groups), but were in general the same. A distinction was made between knowing where the ball is, and not knowing where the ball is:

⁵There is a predefined number of robots, and a predefined number of possible positions on the field, but the possible amount of different messages that can be received - and are also a part of the knowledge base, is not limited. The temporal (time) domain is infinitely long, and all observations could be added to the knowledge base.

⁶A boolean value can be either TRUE or FALSE

- In the first case, the agents reason about whether to interfere, move to a free location, shout, or to move ignore it. When the ball is visible but a team mate has the ball, the general strategy is to move towards a free position, or to advance forward. If an opponent has the ball, defensive players will attempt to move in between the ball and the goal, and bump the offensive player to slow him down and allow other team mates to help in defending the goal area.
- In the second case, the players would generally scout the area around their base positions. The base positions guarantee that the robots are spread across the field and will be able to cover the entire field. Reasoning that allows players to move away from their base position too far or too long will result in clustered groups of robots, and causes large open spaces in the field. This undesirable effect is generally prevented by sticking close to the base position, unless the player is with the ball, or close to the ball and aiding in an attack.

The rule base will return the action that should be performed. This is the best action possible given the current knowledge. This action is high-level, and will have to be translated to the lower level of the effectors. When multiple actions would be returned, the most specific action is executed⁷.

10.3.2.4 Actions

The final element of the implementations deals with the execution of the actions that are decided on in the rule base. An example of such an action would be *dodge opponent*. In order to successfully execute this action, the module requires information from the knowledge base. In order to dodge an opponent, the opponent that should be dodged has to be located. All other known opponents and team mates in the area will have to be considered. The goal position of the robot, and all other variables that are important for determining the optimal low-level instruction are also required.

The robots have three effectors; (1) movement, (2) shouting, and (3) changing behavior. The latter is not used by any student soccer team. Shouting is relatively straightforward. The group has to decide on a certain format, and ensure all communication is in accordance with this format. Movement is still rather complex, as finding the "optimal" movement is not possible. The environment will react on your movement. As such, the predicted shortest path at any point in time may be obstructed by opponents and take more time than an other, longer path would have taken.

⁷Due to the implementation in code, the 'action' is overwritten by the most specific rule.

A more detailed justification of the movement model is given in Section 4.6.2. Dealing with movement is one of the hardest parts of this module.

10.4 Evaluation

The assignment was evaluated in a number of ways, using different criteria. The most important criterium is whether the educational goals for the assignment were met. Another important aspect of the evaluation is what things should be improved for future use.

The first part describes the evaluation results regarding both criteria for the classroom observations that were gathered by the teachers. The second part describes the results from the survey that was distributed among participating students of the MKT-2 project.

10.4.1 Classroom Observations

The classroom observations that are described in this section are a result from observations by the teachers, remarks from the students, the final documentation and implementation, and from several students that participated in the MKT-2 project in 2004-2005 and shared their opinion on the new assignment.

10.4.1.1 Educational Goals

First and foremost, the motivation of the students was extremely high throughout the 3,5 weeks that the assignment lasted. Many students worked on their team during the weekend, day, night, and some students even admitted to missing lectures in order to work on their team.

Group work Because of the high motivation of the students, and the realization that the deadline was only 3,5 weeks away, all student groups recognized the importance of effective task distribution and group communication. The groups had already collaborated during the previous two assignments, and the roles within the group were clearly defined. Throughout the assignment, all group members were present and working together on the different agents. Most groups had (bi)weekly team meetings to discuss the progress.

Developing an implementation from a *vague, high-level* problem description The assignment was divided in an analysis, design, and an implementation phase. An important aspect of the analysis and design is

to distinguish between must-haves, should-haves, could-haves, and would-haves. The limited time that was available for the assignment caused students to consider what elements of their design could realistically be implemented, and what elements should be left out.

Several groups spent a lot of time on the analysis and design, whereas other groups started on their implementation before doing so. The student group that spent the most time on their design won the competition and had the best soccer team. Their initial task distribution, and the time they had planned to spend on the individual tasks corresponded more or less to the actual time that was spent. The groups that started immediately on their implementation struggled to complete everything on time. Integrating the different parts of the system was difficult, and their final implementation was very different from the initial scenarios. These groups also mentioned after the project that they should have spent more time on their design.

Many groups divided the implementation of their team over the group members such that one or two members would work together on a single player. These group members would then be responsible for finishing this player according to the design. Since the programming skill of the different students varies a lot, this resulted in several players that did not meet the specifications of the design, and several students that made their agent oversophisticated. Once all elements were working, the challenge of integrating the separate parts of the system such that the cooperation that was part of the team design functioned according to specification. The debugging and optimizing the system took place in the last week of the assignment prior to the competition.

Learn about distributed AI, MAS, mobile ad-hoc networks. The student groups learned about all these topics throughout the design and implementation of the soccer team. During the design phase, students would frequently reason that, for instance, their offensive agent would walk towards the ball if he would see it. This approach stems from the mindset where problems are solved in a centralized manner. The teachers and certain group members would question the initial simple approach by asking simple *what if* questions. One common trivial scenario is illustrated in Figure 10.3.

In this scenario, the player will see nothing but the ball. Most initial strategies would immediately move towards the ball. In this particular scenario however, there is a team mate that is much closer to the ball, and is likely to intercept it. This team mate is outside the scope of the player, but through communication, and pre-programmed knowledge of the base

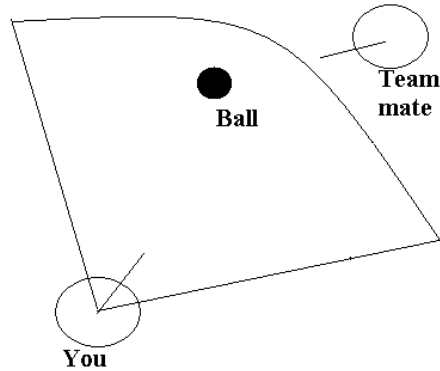


Figure 10.3: An Example Scenario

positions of the team mates, intelligent reasoning could reason about the likelihood of other team mates being closer, and anticipate on this.

Decentralized control Another example is the question about adaptive team behavior. Many student groups had initially included the idea to switch one robot from offensive to defensive when the score would exceed a certain limit. In their detailed design, they realized the difficulty of making this decision without centralized control. Several approaches were thought of. Several included some type of captain, that would reason for the team. An other frequently seen approach would use a predefined order⁸ to overcome the problem of not having any centralized control

Cooperative agents, rule-based reasoning In the design phase, students realized that communicating everything was not a feasible strategy. In order to develop cooperative agents, the students defined a number of social laws in their agents⁹. These laws were formulated during the design, and integrated in the agents in the rule base mechanism. The students also used explicit communication to establish the cooperation. Students used rule-based reasoning in their agents, but this was integrated entirely in their code. As such, the implementations were lengthier and less organized.

⁸For example, when the opponent have a lead of more than 10 goals on your team, the left offensive player becomes defensive. When the lead is more than 20, the right offensive player comes defensive.

⁹This was subconsciously done in their design. An example of such a social law would be saying that when you see a ball, and a team mate can see the ball, and you are further away from the ball, that the team mate will always take the ball.

The students recognized during their analysis that the SimpleTeam that had to be defeated was predictive in many ways. They anticipated on this behavior in their strategies. The extra question that was posed at the end of the assignment, *Assume that you would have to develop a team that would defeat your own team. How would you do this? Explain this in no more than 200 words.* helped students to reflect on the strategy that they have used, and recognize the weaknesses of this. Most teams are relatively predictable, and an analysis of when the opponent team manages to score will give feedback to the weak points in the strategy. This real-time feedback about the strong and weak points of the rule base helped the students improve their implementations. Unfortunately, because of time-related problems, the students were not able to play matches between their teams until the tournament, where many flaws in the rule bases that had gone unnoticed in the practise matches against the SimpleTeam appeared. This experience provided students with insight in the strengths and weaknesses of a rule-based approach.

Increase programming level The soccer assignment is the biggest programming assignment that the students have ever done, and also the most challenging one. The high motivation, and the instant feedback (through playing a match) helped the students in successfully finishing their implementations. The most important lesson for the students during the implementation was that *complexity kills*. Many groups started on very difficult algorithms which then failed for mysterious reasons, rather than starting with a simple framework that works and gradually extending this. Because of the distribution of the players amongst the group members, they experienced first-hand the necessity to standardize the communication (and related objects) between their players, and to program in an organized manner for debugging.

10.4.1.2 Improvement

Based on the previous two assignments, most groups expected that the analysis and design phase would take a small amount of time. A remark that was frequently made was that the design of their team was a lot more challenging than they expected, and that in hindsight, they should have spend more time on it. The importance of the analysis and design should be emphasized, and the teachers should be more strict about not starting on the implementation before the design is approved.

The simulator was released the week before the assignment started. A number of minor bugs were resolved, and some features were added during the first week of the assignment. The reference team that had to be defeated was completed and distributed at the end of the first week of the

assignment. The example team (RandomTeam) used a very complicated and flawed method to determine the movement, that was initially copied by the student groups. This caused confusion among the students, and did not add to their understanding of the environment. All elements of the assignment should be finished well ahead of the start, and the reference team should be thoroughly documented and use well-implemented methods.

The students used rule-based reasoning in their agents, but this was integrated entirely in their code. As such, the implementations were lengthier and less organized. It is good coding style to separate the reasoning from the algorithms and code. This provides a better overview of the reasoning, making it easier to detect flaws. The assignment should force student to design a separate rule base for their players, and use this.

One of the disadvantages of the rule-based approach in soccer is its predictability. When training the team against the SimpleTeam, this is not a problem since the reference team is also highly predictable. As such, the adaptive team behavior that is possible in the simulator is neither required nor used. Two possible solutions exist:

- Develop a number of reference teams that all have to be defeated. The student team will not know beforehand which of the teams it is facing.
- Develop an adaptive team that has to be defeated.

Another interesting dimension of the assignment would be to have an (online) database of reference teams that could be defeated, and a ranking system. Student groups could submit their groups and play matches against implementations from previous years, or against other student groups.

10.4.2 Survey

A survey was distributed among all participating students. This survey is given in Appendix D.

Table 10.2: Survey Results

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Insight in distributed control & ad-hoc networks	0%	10%	20%	70%	0%
Trade-off communication vs autonomous	0%	5%	20%	60%	15%
Learned more than conventional assignment	0%	5%	20%	35%	40%
Assignment was at a good level for first-year students	0%	15%	15%	60%	10%
Increased programming skills	5%	15%	35%	45%	0%
Software is easy to use	15%	30%	35%	20%	0%
I like the high-level interface	5%	20%	35%	35%	5%
I like the focus on cooperation and team work	0%	5%	35%	50%	10%
Manual was useful	10%	45%	15%	30%	0%
Web tutorial for installation / configuration	15%	15%	40%	25%	5%
Web tutorial for making simple player	15%	15%	30%	25%	15%
Web tutorial for explaining assignment	20%	25%	20%	25%	10%
I like the tournament	5%	0%	10%	50%	35%
Teaching assistants were necessary	5%	30%	45%	20%	0%
Assignment was challenging	0%	0%	15%	65%	20%
I liked the assignment	0%	0%	25%	60%	15%
I am pleased with result	0%	5%	45%	50%	0%
Assignment stimulated creativity	0%	10%	15%	65%	10%
Assignment took too much time	0%	0%	25%	30%	45%
I would like to do more soccer assignments	0%	15%	40%	30%	15%

10.4.2.1 Survey Results

The results from the survey are given in Table 10.2. An important consideration regarding the interpretation of this data that has to be made is that there are two different *classes* of students - those that implemented the team, and those that did the presentation. Since both *classes* were involved with the design and analysis, and the educational goals for both *classes* were identical, the survey was handed out to all students. These different classes could be an explanation for some of the results in the survey.

5 things you learned from the assignment The following list contains an overview of the most common keywords that were mentioned by the students.

1. Teamwork (6x)
2. Programming (5x)
3. Ad-hoc networks (3x)
4. (High-level) problem solving (3x)
5. Creativity (3x)
6. Cooperation (2x)
7. Designing, planning, organizing code, working systematically, testing, trial-and-error, how to implement a (high-level) strategy

Manual Remarks

1. Incomplete, could be improved (4x)
2. Coordinate systems needs a better explanation (3x)
3. MessageTuple / VisibleObjects need a better explanation (2x)
4. Clear enough (2x)

Software Remarks The question about software remarks was often misinterpreted, and many remarks about Subversion, Eclipse, and Fleeble were made. There was only one specific remark about the software, which was *easy to use*.

Assignment Remarks

1. More time needed (10x)
2. Fun (5x)
3. Good to learn teamwork
4. Make assignments B and C like this assignment
5. Kick B and enlarge D
6. Include test-opponents, and some unfinished teams that will provoke certain scenarios.
7. More documentation, but (please) no screencasts¹⁰

10.4.2.2 Educational Goals

From these results, the following conclusions can be drawn with regards to the educational goals:

- Students were pleased with their group's final soccer team. Considering the limited amount of time that was available, and the complexity of the assignment, this implies good teamwork within the student groups.
- The students learned to develop an implementation from the high-level problem description.
- The assignment learned students about distributed control, ad-hoc networks, and about cooperation and teamwork.
- The programming skill of the students increased for many students.

Furthermore, with regards to the experience of the students, the following can be said:

- The assignment was challenging
- The assignment was fun
- Most students would like to do more soccer assignments
- The assignment stimulated the students to be creative
- The competitive element (tournament) was very much appreciated

¹⁰There were several 'screencasts' that explained CVS and eclipse produced for the project.

- The focus on team work and cooperation was embraced by most students
- Students learned more from this assignment than they would have learned from a conventional programming assignment
- The assignment was at an appropriate level for first-year students

10.4.2.3 Improvements

With regards to the possible improvements of the assignment, the following conclusions can be drawn:

- More time should be available for finishing the assignment
- The manual should be improved
- Web tutorials for the assignment / software might be helpful, but the opinions on this differ between the students
- The teachers' presence was not a necessary requirement for the assignments
- More documentation should be available
- Add some (unfinished) 'teams' that will provoke certain scenarios. Rather than visualizing the scenario on paper, having a library of preprogrammed scenarios and being able to use this to evaluate the performance would help in improving the implementations.

10.5 Conclusion

include hobbyist observation

10.5.1 Educational Goals

The educational goals that were defined in Section 1.3.3 have been met. Both the classroom observations and survey results unambiguously lead to this conclusion. The students have learned a great deal about problem-solving, group work, various AI techniques, and programming. The assignment was considered very motivating and challenging, and many students indicated they were interested in more assignments based on the soccer simulator.

An interesting conclusion that can be drawn from the evaluation is that the role of the teachers throughout the project was considerably smaller than in the previous assignments. No more than 20% of the students felt that the assistants were necessary for completing the assignment. Especially when taking into consideration that it was the first year that the assignment was given, and there was a lot of room for improvement, this conclusion is very important.

10.5.2 Improvement

It was the first year that the assignment was given, and a number of things can be improved. A common complaint was the lack of time for developing a good team. Next year, assignment B is going to be removed, and the soccer assignment will have two extra weeks.

A suggestion that was uttered by several students was to add some unfinished teams that serve to provoke a number of scenarios. Creating a library of these teams for practising particular aspects of the strategy could be a useful addition.

The reference team that had to be beaten was sufficiently complex for the three weeks that were available to the students this year. When the assignment is extended however, a more complex challenge is required.

Rather than challenging students to defeat one reference team, the assignment could be altered to resemble assignment A. Supply a number of reference teams, and develop a team that - without knowing which team it is facing - is able to convincingly defeat all reference teams.

Improve the documentation and the manual.

A small "problem" with this year's implementations was that most teams would communicate, and cast the object that was received directly to their own object. The reference team did not communicate at all, so during the practise sessions this went all right. When playing a tournament against other student groups however, this caused heaps of `ClassCastException`s. (One of) the reference team(s) should use some communication to ensure this undesirable effect is removed during the practise sessions.

Following the above improvements, using the assignment in distance-learning, or releasing it for hobbyists is expected to yield good results. Teachers were not a necessity during the assignment, as the simulator provides instant feedback (in a match) regarding the performance of a certain algorithm.

10.6 Discussion

This chapter described a soccer assignment for first-year undergraduate students. The soccer simulator is suitable for a wide body of educational environments and levels however. Depending on the educational goals of the project, a custom assignment can easily be tailored. The layered approach of the system model allows for the inclusion of environment noise, effectively allowing simulations of realistic robots.

The ease-of-use of the simulator and the accompanying framework, and the low threshold for developing a team make it an interesting project that is suitable for students without real programming experience. The possibilities of creating adaptive self-learning teams provide a sufficient challenge for the very experienced, however.

Part V
Results

Chapter 11

Conclusions

"Great things are not done by impulse, but by a series of small things brought together." Vincent Van Gogh.

This chapter presents the conclusions of the project. A number of goals for this thesis project were formulated in Section 1.4. The structure of the goals is followed in discussing the realization of all individual goals.

11.1 Literature

The literature survey introduced (multi-) agent technology, the RoboCup Federation, and provided an overview of existing educational assignments based on robot soccer. The RoboCup Federation hosts 5 different leagues; small-sized robots, middle-sized robots, four-legged robots, humanoid robots, and the simulation league. The soccer server application that is used in the simulation league has a steep learning curve and requires an extensive low-level implementation before high-level reasoning can be conducted. The system provides a realistic simulation of the soccer environment, including uncertainty about the sensor readings. Existing applications of robot soccer in education are often involved with both hardware and software aspects. Several courses focus on the soccer server through distributing individual tasks among the students to develop one team, or through building on the work of students from earlier years to develop intelligent teams incrementally. There are no courses where a group of students will develop a team capable of reasoning at a high level from scratch.

11.2 Game

11.2.1 Model

An important part of the project was the design of the model. Rather than developing a realistic soccer simulator, the model is intended to be suitable

as a tool for AI research and education. The laws of the game were defined and compared to human soccer. The most important difference is that the ball is always in play in the model, whereas it can be taken out of play by the referee or leaving the field in human soccer. There is no centralized control such as a coach or a referee. Three types of users will use the system; students, hobbyists, and professionals. Flexibility and extendibility are important requirements of the model. The soccer model features three layers. The player layer, robot layer, and framework layer. The features of the game are defined in their corresponding layers. This layered approach makes the system extensible. Robots have aural and visual sensors with perfect information in a narrow field. Robots can change their behavior, shout, and move. Robot speed is instant and static, speed penalties are awarded for collisions and shouting.

11.2.2 Design

The design of the soccer simulator was based on the model, and influenced by the possibilities and limitations of the agent framework that was used. This agent framework, Fleeble, is a simple Java-based agent framework that allows dynamic loading of agents, and can be used to simulate multiple isolated environments through assigning virtualhosts. This feature is used extensively in the design to restrict player agents to communicating only with the robot layer. The user interface was designed to be simple, intuitive, and visually appealing.

11.2.3 Implementation

Based on the design the soccer simulator was developed incrementally in four phases. First, a basic working framework was implemented. The second phase implemented all basic sensors, effectors and the movement model. In the third phase the user interface was developed, and the robots were displayed. The functionality of the first and second phase was thoroughly tested. The fourth phase dealt with several complicated features such as the collision model and ball movement.

11.3 Research

11.3.1 Agent Strategies

Numerous low-, medium-, and high-level player skills, and a number of inference steps that map the basic percepts to high-level information about the environment were identified. The rule-based action selection mechanism is explained and illustrated. Adaptive team behavior can be divided in short-term and long-term strategic change. Several suggestions on dealing with

distributed control were given. Depending on whether a robot fulfills an active or a passive role in the play, the speed penalty issued for communication is increasingly troublesome. Using social laws to predict and anticipate behavior can reduce the required amount of communication. Self-learning soccer agents are not successful because their performance measure is hard to define. An approach is suggested towards defining accurate performance measures to achieve a self-learning soccer agent. The soccer simulator is a promising tool for AI research.

11.3.2 Developing a Team

The SimpleTeam was developed for use in the MKT-2 project. Groups of first-year undergraduate students would have to be able to convincingly defeat the team in three weeks time, and the team was not allowed to use any communication. The different roles within the team were decided on, and the formation was defined. The different skills, inference rules and the rule-based action selection mechanism were defined. The implementation of the SimplePlayer features three methods that together start with perceptual input, and end with basic action commands. The resulting team meets the requirements, and all student groups managed to convincingly defeat it. An approach is suggested towards an advanced soccer team that uses several of the aforementioned techniques.

11.4 Education

11.4.1 MKT-2 Project

There were two predecessor ad-hoc networking assignments, that faced conceptual and software errors. Based on a thorough analysis of these problems, several conclusions were drawn. It is hard to find a performance measure in ad-hoc networks. Using arbitrary performance measures results in agents that are particularly good at performing on these arbitrary areas, but are neither effective nor efficient. When students can - through a lot of communication - make the network deterministic, they will. This behavior is undesirable because it does not teach students about the distributed control mindset, nor does it teach about the synthesis between autonomous reasoning, social laws, and communication.

11.4.2 Soccer Assignment

The soccer assignment required groups of first-year undergraduate students to implement a team capable of convincingly defeating the reference team. Classroom observations and survey results conclude that the educational goals that were defined in Section 1.3.3 have been met. The students have

learned about problem-solving, group work, various AI techniques, and increased their programming skill level. The assignment focused only on the high-level details, dealing with teamwork and cooperation. The assignment was considered to be very motivating and challenging, and many students were interested in more assignments based on the soccer simulator. Most students felt they could have completed the assignment without the help of the teacher, making the assignment suitable for distance learning. As it was the first year that the assignment was given, several possible improvements were identified. The most frequent complaint was the lack of time for developing an intelligent team. At least two extra weeks will be added for the assignment of next year. The students were very enthusiastic about the assignment. The soccer simulator is suitable for both regular education and for distance learning.

Chapter 12

Discussion & Future Work

The more original a discovery, the more obvious it seems afterwards. Arthur Koestler.

12.1 Discussion

The work for this graduation project focused on three main areas: The game, AI research, and education. The soccer simulator was designed to be a suitable tool for AI research and education. The soccer model that is implemented in the simulator defines the possibilities and limitations of the simulator. The domain of soccer in AI is very broad, and it is impossible for a model to suit all potential users in a single model. For this reason, the model was designed to be extensible. The two chapters dealing with AI research aspects have provided an in-depth example of the possibilities of the simulator, and have shown that the soccer simulator is a suitable tool for AI research on high-level strategic behavior. The classroom observations and the survey results illustrate how the students embraced the assignment. The ambitious educational goals were achieved. These results show that the soccer simulator is a suitable tool for education.

12.2 Future Work

Throughout the individual chapters of this thesis a number of suggestions for future work have been given. This section summarizes these.

12.2.1 Model

The soccer model that was described in Chapter 4 is suitable for developing highly intelligent teams. The layered approach of the model allows advanced customization such as introducing uncertainties regarding the robot sensors and effectors to simulate a more realistic game. Including such uncertainties

allows for the comparison of the performance of a team in several technological states, effectively measuring the influence of enhanced technology on the performance of a team.

12.2.2 Soccer Simulator

The simulator that is developed during this project has no known issues or bugs and is considered stable. A number of possible technical improvements do exist. These improvements are listed below:

1. Move the 'DisplayThread', the thread that presently resides in the SoccerVisualizer and is responsible for calling the methods in the SoccerAgent that process the visibility and movement, towards the SoccerAgent. The SoccerVisualizer should be a thin interface layer that can be used to display the game, and the game should not be dependant on the game. This would also allow simulations to run without the GUI.¹
2. Implement network option. Users should be able to join or host matches and tournaments with other users. How this should be implemented depends on the exact required usage. Setting up one central soccer team database and connecting to this, downloading the desired team and playing against this team would be a possible implementation.
3. Extend the model with uncertainty, to model various states of technological progress, and use this to test the impact of this progress on the performance of a team.
4. Add more human soccer features, such as specifying the speed and direction of a kick, modeling the goalkeeper as a robot with special physical capabilities, or kicking the ball with effect.
5. Add another layer over the player layer that contains pre-implemented functions for, amongst others, movement. This would effectively reduce the process of creating a team to defining a good rulebase.

12.2.3 Agent Strategies

Chapter 7 suggested a number of approaches towards implementing certain player skills, inference rules, action-selection mechanisms, and social laws. Most of these could be made more effective through using other existing

¹The simulation will not be able to run faster however, since the time that is required by Java and Fleeble to process all input and output by all robots is independent of the thread that implements it, and the user interface is very basic and does not require a lot of processing power.

AI techniques, and thorough observations of the environment. An approach was also suggested that has a performance measure for every selected action, and optimizes the thresholds and values determining the execution of that action through analyzing the joint knowledge base of all teammates. This performance measure can then be used in a self-learning agent. The agent would reason with the same set of rules, but optimize the perceptual reasoning and skill variables. It would be interesting to see whether this approach would work and whether it would indeed enhance the performance over a longer period of time.

12.2.4 Soccer Assignment

It would be interesting to add a number of unfinished teams that serve to provoke a number of scenarios. Creating a library of these teams for practising particular aspects of the strategy could prove to be a useful addition. The reference team that had to be beaten was sufficiently complex for the three weeks that were available to the students this year. When the assignment is extended however, a more complex challenge is required. Rather than challenging the students to defeat one reference team, the assignment could be altered to resemble assignment A. Supply a number of reference teams, and develop a team that - without knowing which team it is facing - is able to convincingly defeat all reference teams. Furthermore, the documentation and the manual should be improved. Following the conclusion that the teaching assistants were not required to successfully complete the assignment, using the soccer assignment in distance learning is expected to yield good results. The time that will be available for the assignment of next year will be extended by at least two weeks.

Appendix A

Publication for Third E-Learning Conference on Computer Science Education

The following paper was submitted to and accepted for oral presentation at
ELCONF'06

Teaching Artificial Intelligence Techniques Using Multi-Agent Soccer

I.J.J. Borm
Delft University of Technology
Mekelweg 4, 2628 CD Delft,
The Netherlands
iweinb@gmail.com

L.J.M. Rothkrantz
Delft University of Technology
Mekelweg 4, 2628 CD Delft,
The Netherlands

L.J.M.Rothkrantz@ewi.tudelft.nl

ABSTRACT

This paper describes a method of teaching rule-based reasoning, ad-hoc networks, multi-agent systems and agent technology through multi-agent soccer. The coursework is a set of assignments that require students to implement intelligent agents that can control the soccer players. The players form an ad-hoc network that is utilized for communication and cooperation. Students have to beat the reference team to pass the assignment, where a competition between student teams provides an extra incentive for the students. We implemented a running version of the system. The system was tested in a classroom environment. The assignment, system and test results will be discussed in the paper. The coursework presented in this paper bridges the gap between theory and reality in a fun, motivating way.

Categories and Subject Descriptors

K.3.1 [Computers and Education]: Computer and Information Science Education – *computer science education*.

I.2.1 [Artificial Intelligence]: Applications and Expert Systems – *games*

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – *coherence and coordination, intelligent agents, multi-agent systems*.

General Terms

Algorithms, Documentation, Design, Experimentation

Keywords

Agent Framework, Ad-hoc Networks, Distance Learning, Multi-Agent Soccer, Multi-Agent Systems, Programming Assignments.

1. INTRODUCTION

Multi Agent Systems (MAS) are an increasingly popular domain in AI. A special example of MAS are robot or embodied agents. One of the challenges of this domain is “By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team” [4].

Researchers are looking at the high-level and low-level aspects of building a team that can reason autonomously and cooperate with its team mates to achieve this ambitious goal.

The threshold for starting in this domain however, is rather high. A lot of low-level issues have to be resolved before reasoning on a strategic level can be done. Robots are primarily concerned with things they can determine about their environment – is this robot

friend or foe? Is that object a robot or a ball? Where am I? How fast is the other player moving? Although (probabilistic) answers to these questions can be given, the processing time, camera quality and position tracking are some of the problems that harm a proper strategic approach to a soccer team.

The coursework presented in this paper provides students with a multi-agent soccer simulator that has all low-level functionality preprogrammed. A simple interface for controlling players is provided, allowing students to start immediately on a strategic level, without being bothered by the low-level image processing and reasoning bottleneck. There is no central authority – to better resemble reality, and to keep the focus of the assignment on controlling the players.

Students have to program a team of intelligent agents that control the players. The team has to communicate and cooperate in order to beat the reference implementation. To further stimulate students, a competition is held. Through designing and implementing their teams, students learn about Multi-Agent Systems (MAS), Ad-hoc Networks, Rule-based Reasoning and Agent Technology.

The coursework uses a Java-implemented Agent Framework specifically designed for teaching (introductory) artificial intelligence.

A classical approach of teaching students AI is through (conventional) programming assignments. We designed a new method, and expect students to be highly motivated and to learn more.

In this paper we shall first provide the background information about the introductory AI course to which the coursework belongs. Then a brief overview of the simple agent framework on which the coursework is built is given. Then the soccer simulator is explained. The assignment forming the coursework will be explained next. Finally, survey results and classroom experience will be discussed

2. INTRODUCTORY COURSE ON AI

The coursework presented in this paper was used in an introductory first-year undergraduate course on AI. Participants are all from Media and Knowledge Technology, a variant of Computer Science studies at Delft University of Technology. The course started in 2001-2002 and aims to achieve the following [1]:

- Introduce basic concepts of knowledge engineering and relevant AI techniques including search algorithms, knowledge representation methods, rule-based reasoning algorithms, and agent technology.

- Explain and instruct in issues related to AI programming in general and intelligent MAS in particular.

Rather than pursuing these goals through conventional programming assignments, the project consists of a number of challenging (group) assignments.

The course consists of 20 hours of lectures – followed by an oral exam, and 80 hours of practical work.

The teachers are actively involved in supporting the students throughout the project. Through a kick-off lecture, the assignments are introduced. Groups are formed based on performance in the individual assignment, creating homogeneity in groups. Good groups are given more freedom and are stimulated to be more creative, whereas extra guidance is given to the groups that did not perform as well.

The teachers are always present during the lab hours, for answering questions and monitoring the groups. Every week, a brief group conversation with the teacher is held to monitor progress and recognize problems within groups.

The project consists of 4 assignments; A,B,C and D.

- The first is an individual assignment where a Roshambo agent has to be programmed that defeats our reference agents. This assignment filters out students that lack the programming skills required to succeed in further assignments. Throughout the rest of the project, it is assumed all students are capable of programming java at a reasonable level. All further assignments are in groups of 5-6 students.
- The second assignment focuses on rule-based reasoning. Based on questionnaire data, the ideal group member is found.
- The third assignment introduces students to semantic networks. A network of the International Movie Data Base (IMDB) is created and the ‘hero’ of Hollywood determined.
- The last assignment, D, has always been problematic. The goal of the assignment is originally to teach students about MAS and provide insight in the difficulties that are encountered when dealing with MAS. Various approaches have been attempted, but both software and conceptual flaws in the assignments have so far prevented this assignment from being a success. A new assignment was created in 2004-2005 [1]. Although perceived as enjoyable and motivating, the artificial point distribution of the assignment caused simple strategies to win from intelligent ones.

Throughout the project, students become more acquainted with the agent framework (Fleeble), programming agents in java and working together in groups. Students also learn to deal with increased amounts of freedom. For the first and second assignments, the path from goal to implementation is relatively straightforward. Students know what steps need to be taken and implement them accordingly. In the third and fourth assignments, a lot more freedom is given. The goal is

clearly specified, but several paths can be taken, none necessarily much better than the other.

3. FLEEBLE AGENT FRAMEWORK

The first-year undergraduate students participating in the project have only little experience in programming Java. To keep the focus of the project on the assignments rather than on learning a complex agent framework, a simple agent framework called Fleeble was developed in previous years of the project.

Fleeble is Java-based and provides all functionality required for the project. It allows concurrency (multi-threading), multiple agents (and easy communication between these agents) and namespaces. A thorough description of these (and all other) features can be found in [2].

Namespaces simulate different computers. When loading a child Agent, a certain name space can be given and Fleeble will lock the Agent’s communication to this namespace. This is particularly useful for multi-agent soccer, as it enables us to force namespaces on player agents, such that they can only communicate with the framework and not directly with each other. Fig. 1 shows the Fleeble GUI, with the soccer simulator running.

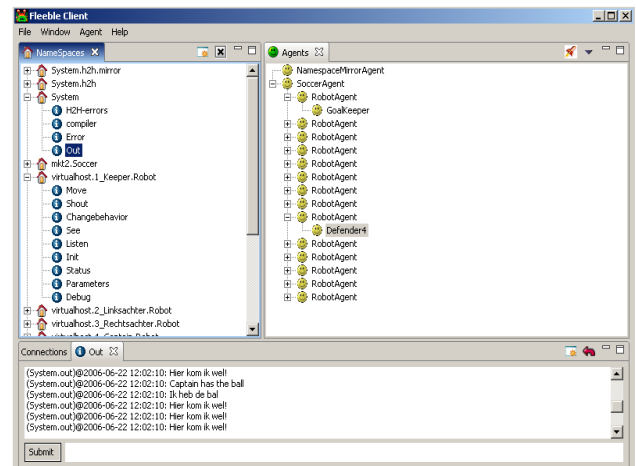


Figure 1. The Fleeble GUI

A comprehensive tutorial can be found in [2]. Templates for agents are included in the assignments, to give students a head

start. Fleeble will automatically compile agent code, so the only requirements for working with Fleeble are an editor, Fleeble, and Java 1.5 installed.

Because of its excellent documentation, tutorials and example code, only basic Java knowledge is required to start working in Fleeble.

4. MULTI-AGENT SOCCER SIMULATOR

The goal of the assignment is for students to learn about MAS, ad-hoc networks and rule-based reasoning. To help students in achieving these goals, while not distracting them from low-level problems and concerns, the soccer simulator was designed (See Fig. 2).



Figure 2. The Multi Agent Soccer Simulator

Teams consists of 7 players. A player on the field is a 'Robot Agent' (the hardware), which is controlled by a 'Player Agent' (the brains). The Player Agent has a number of inputs (Listen, See), and a number of outputs (Shout, Move).

4.1 Input / Output

Shouting can be done at any time by any agent. Agents listening within a predefined distance (See Fig. 3) will receive the message.



Figure 3. Part of the soccer simulator. The yellow circle defines the shouting distance, red arc is what the player sees

A shout can include any Java Object, so it can be very valuable to exchange information about positions and strategies of other players. Rather than restricting shouting by allowing only a certain number of shouts per time unit, or by setting a maximum message size, shouting is penalized by reducing the speed of the shouter by a certain percentage for a few seconds. This causes students to have very diverse strategies. Some will be hesitant to

use shouting often, whereas others will take the speed penalty for granted.

Players receive a visual update about 3 times per second, through the See channel. This consists of a list of all Visible Objects. A Visible Object contains position information, the name and team of the player. The player self, and all objects within a certain distance / angle are added to this list (See Fig 3.). This information, and what is gathered through listening are the only sources of input that a player has.

The position information contained in the visual updates comes in three flavors. Absolute (screen) coordinates, relative to team coordinates, and relative to self coordinates. The relative to team coordinates are very useful for providing orientation – a team has to work on both left and right sides of the field. The relative to self coordinates make life a lot easier for students for determining how far away objects are, whether they are on your left or right side, and several other useful facts.

Players can move their Robot Agent by telling it to move (Forward, Backward, Left or Right). A turn can be specified as well (Left, Right, Straight) and a player can block or kick. Although Player Agents are not restricted in the amount of move requests they send, only the last one that is received before a frame update is processed. Bumping into other players, or being bumped into is penalized by a fixed speed reduction lasting several seconds.

Since it is not possible to know beforehand the strategy of the team you are playing against, it would be nice to have some autonomous adaptive behavior. Therefore, an option to change the 'brains', the Player Agent on a Robot Agent is included. When a team is losing with 5-0, it is probable that proceeding with the current strategy is not going to yield better results. Changing an offensive player to a defensive one, or vice versa, is an interesting dimension that motivated students could explore.

4.2 Randomness

The simulator is mostly deterministic. All parameters are known to all players, and can be used to calculate for instance where a ball will come to a stop, how much time it will take to move to a certain position, etc. There are only two random elements in the game, causing every run to be different.

First, there is a random factor when kicking the ball. To prevent lucky shots from a very large distance, and to stimulate strategic behavior, a certain random distortion is added to every shot.

Secondly, there is "Java-induced randomness". There are 2x7 Robot Agents, 2x7 Player Agents and a Framework Agent. All Agents have their own thread, and thread handling in Java is not completely deterministic. Because of this, sometimes a certain player will get his See updated just before a frame update, and another player after. These random variations are equally distributed amongst both teams, and make every game unique.

4.3 Team creation

A soccer team consists of 7 players, each with a Player Agent, name, base position and an icon. The information is stored in an XML file. The XML file can be hand written, or generated using a visual editor (See Fig. 4).



Figure 4. Interface for creating team setup XML

5. MULTI AGENT SOCCER ASSIGNMENT

The assignment is formulated as follows:

Design and implement a number of Player Agents that cooperate in order to win the game. It is only possible to pass the assignment if the team convincingly beats the reference team. A team consists of 7 players. A player on the field is a 'Robot Agent' (the hardware), which is controlled by a 'Player Agent' (the brains). The Player Agent has a number of inputs (Listen, See), and a number of outputs (Shout, Move). There is no central authority. The user has no influence whatsoever once a game has started. The focus of the assignment should be on cooperation and communication. A lot of thought has to be spent on what approach and strategies are best, and what the weak spots of these would be.

To provide a guideline to the students, the assignment is split up into three parts. In the first part, students have to think about a number of predefined scenarios, and form a strategy for each. The second part requests students to design their *Player Agents*, to provide a clear picture of what their team will do. The third part is the actual implementation of the team.

5.1 Scenarios

In the first part of the assignment, students have to describe how their players will act in a number of scenario's. A differentiation may be made between types of players (i.e. a defense player will respond different from an offense player).

Scenario's are divided in micro (player) and macro (team) scenarios. An example micro scenario is: *'The ball is free'*. Students have to reason about how their agent reacts. Will it work towards the ball, get in between the goal and the ball, communicate to see if other players are closer to the ball? An example macro scenario is: *"Someone from your team has the ball"*. Do you try to stand in a free spot? Do you help him by communicating position information of enemies?

A lot of decisions have to be made in this part of the assignment. Letting students think as their player agents would, is a good

exercise for getting them to understand difficulties and tradeoffs that have to be made.

Before being allowed to proceed to the next part of the assignment, the teacher has to approve of the scenarios.

5.2 Design

During the design phase, students will have a good idea of what is possible and what is not. They use the scenarios from the first part to come up with a full description of their system. This includes additional scenarios (the scenarios from part one only included basic, trivial events), different players (i.e. goalkeeper, defense, captain, offense,...) and a plan for how communication and cooperation between these shall occur.

5.3 Implementation

The final phase deals with the actual implementation of the agents. Considering that Fleeble and the soccer simulator are still (relatively) new to the students, it is impossible for them to estimate beforehand exactly how much time a certain task will take, and whether everything will have the expected outcome. Even for experience programmers, this is a hard task, but for novice programmers, it is a serious problem. Even when their design is perfect, and time constraints seem reasonable, the result will still depend on the individual skills of the group members. It is important to use their design as a guideline, but students will need to learn to iteratively refine their design as the implementation continues.

In this phase, students start working with rule-based reasoning. Although during the year 2005-2006, it was neither obligatory, nor stated in the manual, all groups used a rule-based approach to programming their agents. The rules primarily defined the behavior of the agents. How a rule-based approach is incorporated in soccer agents is shown in 5.4

5.4 Reference Team

To get a passing grade, a student team has to convincingly beat the reference team. Students do not get to see the code for the reference team, but they can observe how it works by playing at it.

The reference team consist of one goalkeeper agent and six *Simple Players, spread over the field*. The reference team uses a rule-based approach to reason about its environment. Every time input arrives through *See* or through *Listen*, the list of positions of all objects is updated. Every time a *See* comes in (3 times per second), the known info is updated and a number of Boolean values are derived. These Booleans deal with: Is the ball position known? visible? kickable? free? in enemy goal area? near base position? with a team player? Is there a team mate standing free? An enemy up ahead? Is it total chaos?

The values of these Booleans are then put into a Rule base, which

```
IF ball_kickable AND NOT in_enemy_goal_area AND team_mate_free
THEN pass_ball
```

reasons about the situation. For example:

The `pass_ball` fact will then cause the player to kick the ball.

The success of any given strategy depends on their analysis (what Boolean values were derived and how well they were

recognized), the Rule base, and the implementation of the actual actions.

The reference team does not communicate at all. All players reason autonomously about what to do.

The keeper will try to stay between the goal and the ball. It will try to take the ball when it is free, and when it has the ball, shoot it into the field. When it doesn't see the ball, it will stay on its base position and scout for it.

The Simple Player will also scout for the ball when it doesn't see it. When the ball is visible and it is no total chaos, it will hunt for the ball. When a team mate has the ball, it will try to stand in a free spot. When it has the ball, it will try to avoid enemies and attack over the flank at which it was positioned (a left back will attack over left flank), moving directly towards the goal when it is getting close. When a ball was recently visible, but isn't anymore, it will try to find out where it is by turning and moving towards it.

The resulting team was neither very smart nor very stupid. It was a challenge for students to convincingly beat the reference team.

6. RESULTS

Evaluation of the assignment occurred through classroom observations and through a survey. After a discussion of the results coming from these, a comparison is made with previous assignments.

6.1 Classroom Results

The expected result was that students would find the assignment enjoyable, motivating and very educative. Based on classroom observations and several conversations with participating students, this is also the actual result. A few things were remarkable, however.

First of all, there were hardly any questions about the assignment all. With previous assignments, and especially with the introduction of new assignments, there have always been loads of questions. The only occasion the teacher was really necessary was for approving scenarios and designs. This leads to believe that the assignment is very well suitable for distance learning.

Second of all, it was remarkable that a wide variety of approaches were attempted. The creativity and skill of individual group members have been put to good use in designing and implementing the agents, and gave diverse results.

A third remarkable observation was that several groups gave the players the group members names. Rather than referring to a certain player as 'Right back', or 'Defense Player' – as expected, they felt affinity for their players. It is also interesting to note that many groups were so enthusiastic about the assignment that they worked on the implementation far beyond the lab hours. Even after knowing their team was good enough to beat the reference implementation, many groups made a big effort to try and win the competition between the groups.

Many students were pleased with their implementations, but regretted to see that a lot of their hard work had turned out useless as it was not used. Getting a complicated strategy to work may look trivial on paper, students gained first-hand experience in the hard reality that it isn't.

Another remarkable result is the results of the tournament. Since groups were formed based on homogeneity in performance, the

expected result would be to have group 1 as champions, group 2 as number 2, etc. Rather than this happening, groups 4,3 and 5 (out of 5 groups) finished 1st, 2nd and 3rd. Although group 1 could have won, their approach was too complicated, causing it not to work. Most groups encountered similar problems, but the urge to win the competition caused some groups to invest a lot more time.

Finally, at the competition between the groups, it was remarkable to see how enthusiastic students were about winning, and how much they enjoyed seeing their strategies at work.

Regarding the educational goals for the assignment, these were achieved through all parts of the assignment. Students gained a lot of insight in the problem of dealing with decentralized control in ad-hoc networks through the scenarios and design. Through implementing the players, most groups found that complexity kills. A clever combination of rather simple methods yields a better result than a poorly implemented ingenious approach.

Students used a Rule-based approach in their implementations, but did so without using separate rule base software. Separating the rules from the code would force students to program in a better organized manner. Changing the assignment to explicitly include this element would probably help students a lot during the implementation.

6.2 Survey Results

After the competition, a survey was distributed among the students to get their opinion of the assignment, the manual and the software. It was remarkable to see several questionnaires with a (very) positive rating, and a lot of useful feedback in the *remarks* section on the one hand, and several very negative results without any remarks.

Because the survey is anonymous, it is not possible to know which students gave what kind of feedback, but a plausible explanation for the result is that within groups, there are two subgroups: Those that did a large share of the programming work, worked with the simulator and read the manual, and those that prepared the group's presentation, wrote reports, helped the programmers and took part in the design process. The first group generally has a positive opinion (this is in agreement with classroom observations), whereas the second group does not.

One of the questions was "*Describe in 5 keywords the things that you learning from the assignment.*"

Some common answers were: "*Teamwork, planning, ad-hoc networks, problem solving, cooperation (software), how to implement a (high level) strategy*"

45% of the students indicated that they would like to do more assignments with Multi-Agent Soccer.

80% (strongly) agreed that they learned more from this assignment than a '*conventional*' practical that asks to implement a certain algorithm.

75% felt that the assignment gave them insight in the problems you encounter in environments without any centralized control, specifically ad-hoc networks.

85% (strongly) agreed that the assignment was challenging.

80% liked the assignment (very much).

85% liked the competitive element (very much)

65% felt that they would have been able to finish the assignment without the teacher's help

In the remarks, the most commonly heard complaint was the lack of time. Students only had 3 weeks to finish the entire assignment. Some even proposed to remove assignment B and enlarge this assignment. Another negative remark was the coordinates (relative to team, relative to self) were not explained properly in the manual.

6.3 Comparison

In the past two years, other assignments were given to the students with similar educational goals [1]. It is not possible to compare survey results, as the questions and the way they were posed differ too much. It is possible however to compare the classroom observations.

Two years ago, the assignment was flawed both conceptually and by software. To combat these problems, the *Smurf* assignment came into existence [1]. Although the software was working reasonably well, the artificial point distribution system caused trivial solutions to beat highly intelligent ones. Because of this, no 'good' implementation could be made.

With the lessons learned from previous assignments, the soccer assignment tries to stick close to a realistic environment. Speed penalties are given for communication, rather than an artificial point distribution. Because the assignment was closer to reality, students could identify better with their agents and this in turn further increased their motivation.

"*It actually works*" is a common utterance by students that participated in the course in previous years, after having seen the soccer assignment.

7. CONCLUSIONS

In this paper we have described a method of teaching Ad-hoc Networks, Rule-based Reasoning, Multi Agent Systems and Agent Technology using a multi agent soccer system. We implemented a running version of the soccer system and tested the assignment in a classroom environment.

We gathered results through anonymous questionnaires and classroom observations. Both methods show that students enjoyed the assignment and felt motivated by it. The educational goals were achieved, and many group work related lessons were learned.

Since the assignment only focuses on high-level strategic decisions, rather than the low-level issues currently being looked at in the robotic soccer field, students had more freedom for creative, original solutions. Students learned while implementing

that complexity kills. A clever combination between simple methods yields far better results.

Adding the competitive element to the assignment gave students extra incentive for hard work, and the competitive element was greatly appreciated by the students.

8. FUTURE WORK

The assignment is a good challenge for first-year undergraduate computer science students. Due to the level of these students and the short period of time allocated for the assignment, the resulting teams were not spectacular.

It would be interesting to extend the system with various other (more difficult) reference implementations, alter the assignment to defeat these teams and give it to graduate students.

Adding a self-learning reference implementation would add an interesting dimension to the analysis phase of the assignment, as well as provide a real challenge for graduate students.

Most students felt that they could have done the assignment without the help of the teacher. This leads to believe that a modified version of the assignment would be particularly useful for distance learning.

9. ACKNOWLEDGMENTS

The authors would like to thank all MKT undergraduates who evaluated the course in 2005-2006. Special thanks go to Cristiano Betta, Ilyaz Nasrullah and Paul van den Haak for their help in managing the course in 2005-2006.

10. REFERENCES

- [1] Pantic, M.; Zwitterloot, R.; Grootjans, R.-J., "Teaching ad-hoc networks using a simple agent framework," *Information Technology Based Higher Education and Training, 2005. ITHET 2005. 6th International Conference on*, vol., no.pp. S2A/6- S2A11, 7-9 July 2005
- [2] <http://www.fleeble.net/> (last visited: June 23, 2006)
- [3] Pantic, M., Grootjans, R.J., Zwitterloot, R., "Fleeble Agent Framework for teaching an introductory course in AI", *Proc Int'l Conf. Cognition and Exploratory Learning in Digital Age*, pp. 525-530, Lisbon, Portugal, 2004.
- [4] <http://www.robocup.org/> (last visited: June 23, 2006)

Appendix B

Tournament Parameters

The TournamentParameters contain a number of variables defining the rules of the game. These are passed on by the SoccerAgent to the SoccerVisualizer, all RobotAgents and all PlayerAgents. As such, all rules are always known by all agents. Knowledge of these rules may be beneficial for determining strategies.

Table B.1: An overview of all Tournament Parameters

Parameters	Default	Description
player-startnumber	1	Robots will always get a number prefixed to their names to make sure the names are unique - the name is frequently used as an identifier. This startnumber determines the first number. The prefix is incremented by one for every subsequent player that is added.
num-player	7	The amount of players on each team
refreshrate	25	The amount of times per second that the system will attempt to redraw the screen. If, due to heavy computation or some other reason, this refreshrate can not be met, the highest possible refreshrate is used.
player-radius	19	The radius of a player in pixels. Custom icons may be used for players of different proportions, but internally the size of every player will be kept at this value.
field-width	1024	The width of the field in pixels. On the screen this is the <i>horizontal</i> part of the field.
field-length	768	The length of the field in pixels. On the screen this is the <i>vertical</i> part of the field.
Continued on next page		

Table B.1 – continued from previous page

Parameters	Default	Description
field-metric-length	100	The metric <i>length</i> of the field ¹ . This is used in conjunction with field-width in calculating for instance pixel speed from metric speed.
view-angle	120	The viewing angle in degrees of the robots ² . On the GUI, upon clicking on a robot, the area within the red arc represents what is in the viewing angle.
view-range	30	The number of meters within which objects can be observed by robots. This is represented on the GUI by the red arc.
view-update	300	The amount of milliseconds between sequential updates on the <i>Robot.See</i> channel.
rotation-increment	20.0	The number of degrees that a turn will rotate the robot ³ .
robot-speed	8.0	The velocity of the robot in meters per second. ⁴
shout-penalty	2500	The amount of milliseconds that the speed penalty inflicted for shouting a message will last.
shout-slowdown	0.75	The <i>robot-speed</i> ⁵ will be multiplied by this factor if a robot has shouted during the last <i>shout-penalty</i> milliseconds.
shout-range	25.0	The amount of meters that a shout can be heard.
collision-penalty	2000	The amount of milliseconds that the speed penalty inflicted for taking part in a collision between two robots will last.
collision-slowdown	0.4	The <i>robot-speed</i> will be multiplied by this factor if a robot has shouted during the last <i>shout-penalty</i> milliseconds.
Continued on next page		

¹Length refers to the longest side of the field, which is described by the parameter field-width

²As Java uses radians as a default in its implementation of Math functions, the simulator always uses radians. The parameters are in degrees because they are easier to visualize for students. Conversion to degrees is easily done using Java's Math.toDegrees() function.

³Movement is described in section 5.5

⁴The actual speed might differ from this in a number of situations, such as recent collisions or shouting.

⁵With *robot-speed* we mean the current value of this speed. There are several parameters possibly affecting the robot's speed that do not rule each other out, but are all multiplied with each other

Table B.1 – continued from previous page

Parameters	Default	Description
ball-radius	12	The radius of the ball in pixels.
ball-friction	0.96	Every update ($= 1 / \textit{refreshrate}$ seconds) the ball's velocity will be multiplied by this factor, to represent the friction of the field.
ball-description	"Ball"	The name of the <i>BallObject</i> . This can be used to check whether a certain <i>VisibleObject</i> is in fact the Ball.
kick-randomness	22.5	The maximum number of degrees that a kick will be off center in either direction.
goal-width	156	The width of the goal in pixels. From the middle, the goal will stretch $\textit{goal-width} / 2$ to the left and $\textit{goal-width} / 2$ to the right.
game-time	10	The amount of minutes that a single game will last.

Appendix C

Detailed description of MKT-2 Project

This chapter describes the background of the MKT-2 project, the setting, and the educational approach. The first section describes the history of the project. The next section describes the background. The background knowledge of the students is described, and the various evaluation tools that are used by the teachers are introduced.

C.1 History

In the academic year 2001-2002, the Computer Science curriculum at Delft University of Technology was split up in two variants. The first being Software Engineering, the second Media & Knowledge Technology (MKT). The latter has as its main objective to create engineers who are able to design and develop intelligent systems for multimedia and multimodal information and knowledge processing, and who are able to design, realize and deploy properly working man-machine interfaces. For first-year undergraduate students, the main difference between the two variants is that two specialization courses, together with their corresponding projects, are different. The second project in the first year¹ for students doing MKT was developed in 2001-2002. This project (MKT-2) was an introductory AI course, with the following two aims [7]:

1. To introduce the basic concepts of knowledge engineering and the relevant AI techniques, including search algorithms, knowledge representation techniques, rule-based reasoning algorithms, and agent technology.
2. To explain and instruct on issues related to AI programming in general and intelligent (multi-) agent applications in particular.”

¹The project and course are given in the fourth quarter of the year.

The educational approach that was used in the project is described in detail in Section 9.1.1. It is very different from the conventional approach, and is based on a synthesis between objectivist and constructivist approaches.

The enthusiastic and highly committed teaching assistants have contributed to the project's success, and implemented the feedback of previous years. Fleeble [9][11] was developed, and many assignments were changed and improved. The educational approach was evaluated and also improved throughout the years.

C.2 Background

C.2.1 Students

Each year about 30-40 students participate in the course. These first-year undergraduate students have no previous experience with any AI techniques, but they have had three courses consisting of lectures and practical assignments teaching them basic Java programming skills. They have little experience in group work, as they have only had one group work project earlier in their first year.

C.2.1.1 Java Programming Skills

Although the three first-year undergraduate Java courses that the students will have passed prior to starting on MKT-2 do learn the basics of Java programming, the students will not have any experience with transforming an abstract problem definition to working code. The practical assignments corresponding to the courses test whether students can implement a highly specific assignment, but they can not guarantee that students thoroughly understand what they are doing, that students made everything by themselves (the assignments are individual, but group work is inevitable), and that proper coding style and thorough documentation is used.

C.2.1.2 Group work

The students have had one group work project before, MKT-1, where they were divided into groups of 5-6, and have as such gained some team work experience. Classroom observations suggest however that they still have a lot to learn in this area. In contrast to the first project, MKT-2 has serious deadlines and challenging assignments. There is a lot more time pressure and tasks have to be distributed among the group members. Due to the limited programming experience, estimating whether or not a programming assignment can be completed by a group member, and in what time, is a difficult process. To ensure all participants have a certain programming level,

and are capable of functioning under time pressure, an introductory assignment was devised that will remove potential freeloaders. This assignment shall be further discussed in section [check!](#).

C.2.2 Evaluation

C.2.2.1 Studentrate

Despite this effort to stop freeloaders, every year, freeloaders have been identified during the project. These people generally have made some effort, but are notoriously absent during team meetings, often miss their targets, and lack responsibility. To tackle the problem of freeloaders, a system called Studentrate was introduced. The system asks students to rate their peers anonymously, on job performance, attitude, leadership, management of resources and communication, on a 1 to 5 scale. See Table C.1, and Table C.2 (continued) for details.

Students are asked to fill it in halfway through the project, and at the end. They will be able to see how their peers rated them on average, but will not be able to see the individual ratings. The introduction of Studentrate provides a lot of feedback to the teaching assistants, as it will generally confirm suspicions of freeloaders, or signal problems that were unnoticed before. Some groups prefer to do most of their work at home, and without Studentrate, it would be hard to gain insight in what members are contributing enough. After several years of working with the system, it is also important to note that several students dislike rating each other, and will give all team members a score of 4 or 5 on everything. The interpretation of the results is at least equally important as the gathering. It has occurred several times that one or two students - in a group of 6 - do about 80% of the work. These people tend to avoid confronting their group mates with this problem, and will often refrain from informing the teaching assistants of the unequal work distribution. Through the anonymity of Studentrate however, they are free to spill their guts. It is important for teaching assistants to attach a certain value to the ratings of group members. It is obvious that the most active members have higher credibility in estimating how much work other members of the group have actually done.

When freeloaders are identified, the teaching assistants first call for a meeting with the group without the freeloader, then a meeting with the freeloader individually, and finally a meeting with the entire group and the teaching assistants. After these meetings, a decision is made on whether or not the freeloader gets a second chance, or if he is removed from the project. Only in extreme cases are people removed from their groups, but it has occurred several

Table C.1: Studentrate

	Job Performance	Attitude	Leadership
5 (++)	Consistently does more than required. Work is of exceptional quality.	Positive and professional attitude, which favorably influences other company members.	Takes initiative to seek out work, concerned with getting the job done. Very involved in the technical project.
4 (+)	Sometimes does more than required. Work is of high quality. A producer.	Positive attitude toward project and the team.	Readily accepts tasks, sometimes seeks more work. Gets involved in the project.
3 (+/-)	Performs all assigned tasks. Quality of work is acceptable.	Neutral attitude.	Gets involved enough to complete tasks. Does his/her share.
2 (-)	Performs all assigned tasks. Work must be redone or repaired to meet standards.	Negative attitude toward project and/or team.	Tends to watch others work. Gets involved only when necessary. Volunteers to help when it will look good.
1 (-)	Performs some assigned tasks. Work must be redone by others to meet standards.	Negative attitude which adversely affects other company members or project.	Lets others do the work; does the minimum he/she thinks is needed to get by.
NR	No rating For example, you never worked with that person, or you don't want to rate yourself.	No rating For example, you never worked with that person, or you don't want to rate yourself.	No rating For example, you never worked with that person, or you don't want to rate yourself.
NFI	Not filled in	Not filled in	Not filled in

C.2.2.2 Group Formation

The group formation for the project uses a new approach. A lot of the recent literature suggests using heterogeneous groups, as a diversification in the skills of students would result in more balanced groups, where the individual's strengths can be used to their fullest potential. Common group formation techniques used at Delft University of Technology are either random, self-selected or through using the Belbin test. One strong argument against allowing student to form their own groups is that they will often

Table C.2: Studentrate (continued)

	Management of Resources	Communication
5 (++)	Uses time effectively in and out of group and works to get others to do the same. All tasks completed on or ahead of schedule.	Oral and written skills excellent. Very effective within the group and to reviewers.
4 (+)	Uses time effectively in and out of group. Completes all tasks on time.	Usually effective
3 (+/-)	Wastes some time in group, but works hard when a deadline is near. Most tasks completed on time.	Generally gets the point across. Tries to improve in weak areas
2 (-)	Wastes most of group time. Seldom seen doing productive work. Some tasks completed late.	Skills ineffective. Makes an effort to improve.
1 (-)	Does little useful work in group or out; wastes his/her time and others. Work is constantly late.	Skills ineffective. Makes little or no effort to improve.
NR	No rating For example, you never worked with that person, or you don't want to rate yourself.	No rating For example, you never worked with that person, or you don't want to rate yourself.
NFI	Not filled in	Not filled in

stick together throughout their curriculum, and as such miss the opportunity of learning from other colleagues. In the MKT-2 project, a formation approach evolved from empirical evidence by the teaching assistants. Based on the quality of the code of the individual assignment, and the time it was delivered, homogenous groups are created, based on skill and enthusiasm. Several arguments support this approach: First of all, in groups with a lot of variance in the skill level, the most skilled members will do all the programming work - they are the best equipped, and will not want to submit products of the low quality that the less skilled members would produce. As such, it frequently occurs that the less skilled members end up only writing the reports. The distinction between these people and freeloaders is very hard to make. When they do get a programming task, they will need a lot of help and the most skilled group members will spend a lot of time and effort helping their peers. This is not a pleasant working environment for either the skilled or the unskilled group members. Furthermore, the quality of the final product is likely to be negatively affected by this. The expectations

that the teaching assistants have of the highly skilled groups will be higher than that of the lower skilled groups. The lower skilled groups will receive more assistance and help with programming, and more monitoring from the assistants, whereas the highly skilled groups will be given more freedom. They will receive challenging questions regarding their implementation, and stimulated to make highly intelligent solutions. The skilled groups will learn more, as they can function on their own level, whereas the less skilled groups will also learn a lot more, as they have to actually make the implementations, rather than simply writing the reports. It was found that members from all groups were quite happy with their group, and there were relatively few freeloaders. The results produced by the highly skilled groups were often indeed of higher quality than that of the lower skilled groups. It is interesting to note that the average groups would often challenge the most skilled group in the group assignments, by putting in a lot of extra effort.

C.2.2.3 Gantt Charts

The project has a lot of time pressure and tight deadlines. To further help students in dealing with these, every group assignment has an analysis and design phase, where students are instructed to study the assignment, identify the tasks, estimate the duration of these tasks and distribute them among the group members. A Gantt chart (See Figure C.1) [75]

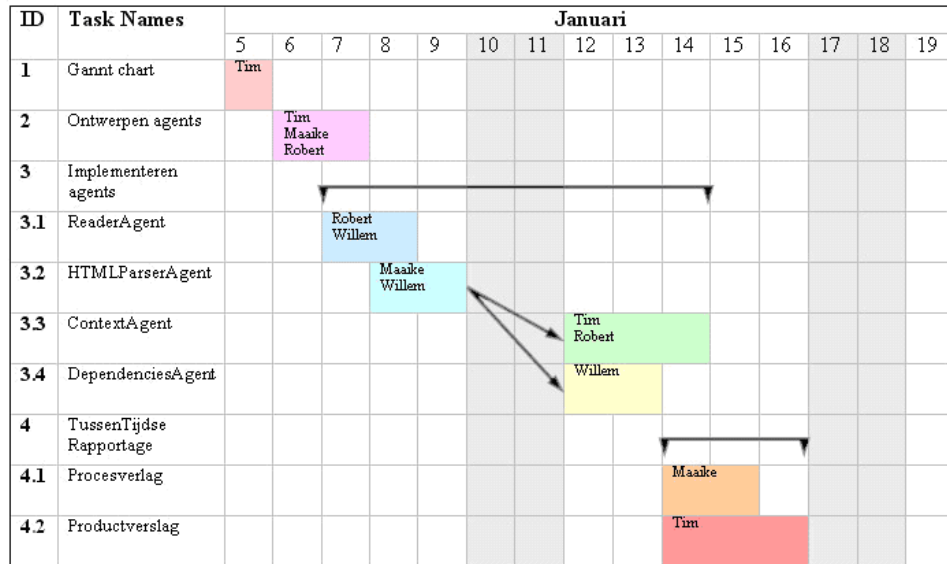


Figure C.1: An Example Gantt Chart

has to be approved by the students before they can start on the implementation. Another Gantt chart has to be made after the assignment is

done, showing the amount of time actually spent on tasks. This is useful for students to evaluate their actual performance to their initial estimations. It is also very useful for the teaching assistants to recognize problems in the group, and to see which members contributed most. It provides a good overview of the assignment, all subtasks, and the distribution in the group. It is quite common to see that the initial Gantt chart has an equal work distribution among the group members, while the final chart shows two people that did nothing more than write the report.

C.2.2.4 Data Flow Diagram

Apart from the Gantt Chart, students will also have to create a Data Flow Diagram (DFD) prior to starting on the implementation phase. The DFD shows all agents, all channels, and all communication from Agents over these channels. The DFD has to be approved by a teaching assistant before students can start working on their implementations. Figure 9.2 illustrates such a DFD.

The Gantt Chart and DFD together ensure that the students know what has to be done for the implementation phase. During the implementation, the teaching assistants remain actively involved in resolving bugs, answering questions and monitoring the group process and progress. As the available time for each assignment is around 2 weeks, there are no intermediate deadlines. Recognizing the importance of timely feedback, grades and comments will generally be given to the groups 1-2 weeks after completion of the assignment.

At the end of each assignments, groups have to write a product report consisting of the problem definition, the design, the implementation, conclusion and answers to some extra questions about the theory of the assignment. This documentation serves several purposes. First, it forces students to keep an overview and document their work. Second, it deals with the students that have not programmed as much. These are often selected to write the reports, to make up for their smaller contribution to the programming. Through writing the reports, they will be aware of the work their peers have done, and understand what decisions had to be made.

C.2.2.5 Oral Exam

After all assignments are done, a group grade is composed as a weighted average over all group assignments. An oral exam, where the entire group and all teaching assistants are present is then held. For this oral exam, the teaching assistants carefully study all classroom observations, Studentrate ratings, Gantt charts, reports and code for all groups and all individual

members. Often one or two members are thought to have played a key role in the group's success through their effort in programming, and these will be given the chance to prove themselves during the oral exam. Students that are on the edge of being freeloaders will have to prove themselves to be worthy of the group grade. During the oral exam, difficult questions regarding the theory or the implementation of the assignments will be posed to these students, and the teaching assistants will come up with individual grades to reward those that put in a lot more effort, and punish those that did not.

Appendix D

Soccer Assignment: Survey

The following survey was handed out to all participating students of the MKT-2 project in 2005-2006.

Multi-Agent Soccer Survey

MKT2-Project
 May-June, 2006
 iweinb@gmail.com

This was the first year the multi-agent soccer assignment was given. The following questions will evaluate your opinion on the assignment. Please bear in mind that as it was the first year, some variables, features and (unknown) bugs will see improvement for the next time the assignment is used.

Filling in this survey should take about 5 minutes. Please complete the survey and return it to one of the project assistants. Thank you!

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Multi-Agent Soccer gave me insight in the problems you encounter in environments without any centralized control (Mobile Ad-Hoc Networks)					
Multi-Agent Soccer made me think about the trade-off between communicating and reasoning autonomously					
I learned more from this assignment than I would have in a 'conventional' practical where you are asked to implement a certain algorithm					
The assignment was at a good level for first year computer science students					
Multi-Agent Soccer increased my programming skills					
The Multi-Agent Soccer software is easy to use					
I liked not having to worry about low-level details such as ball speed, whether someone is from your team or who is shouting					
I like the fact that the assignment focuses on teamwork and cooperation					
The manual was useful					
I would have liked to have a web tutorial (screen cast) for installation and configuration					
I would have liked to have a web tutorial (screen cast) for making a simple player					
I would have liked to have a web tutorial (screen cast) for explaining the assignment					
I like the competitive element (tournament)					

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I would not have been able to finish the assignment without the assistants.					
Multi-Agent Soccer was challenging					
I liked the assignment					
I am pleased with the team my group made					
The assignment stimulated me to be creative					
The assignment took too much time					
I would like to do more assignments with Multi-Agent Soccer					

Describe in 5 keywords the things that you learned from the assignment:

.....

Remarks about manual:

.....

Remarks about software

.....

Remarks about assignment

.....

Additional comments

.....

*Thank you very much for completing the survey!
 We will process your feedback and implement it where possible!*

*If you would be interested in doing more Multi-Agent Soccer assignments,
 are interested in the domain or have more feedback you would like to share,
 then don't hesitate to send an e-mail to iweinb@gmail.com!*

Bibliography

- [1] H. Kitano and M. Asada. *The RoboCup humanoid challenge as the millennium challenge for advanced robotics*, Adv. Robot., vol. 13, no. 8, pp. 723-737, 2000.
- [2] FIFA - Federation Internationale de Football Association, Laws of the Soccer Game, <http://www.fifa.com/en/laws/menu.htm>, Last visited at August 25, 2006.
- [3] J. Schaeffer, A. Plaat. Kasparov versus Deep Blue: The Re-Match. Journal of the International Computer Chess Association, Volume 20, Issue 2, pp. 95-101, 1997.
- [4] RoboCup Federation, Official website, <http://www.robocup.org/>, Last visited at August 25, 2006.
- [5] H. Burkhard, D. Duhaut, M. Fujita, P. Lima, R. Murphey, and R. Rojas. *"The road to RoboCup 2050."*, IEEE Robotics & Automation Magazine, 9(2):3138, 2002.
- [6] Wikipedia - An explanation of the Offside rule, [http://en.wikipedia.org/wiki/Offside_law_\(football\)](http://en.wikipedia.org/wiki/Offside_law_(football)), Last visited at August 25, 2006.
- [7] R.J. Grootjans (advisor: M. Pantic), *"A Simple Agent Framework for Teaching AI Programming to Novices"*, MS thesis, Man-Machine Interaction Group, Delft University of Technology, Delft, 2004.
- [8] M. Pantic, R. Zwitterloot, R.-J. Grootjans, *"Simple agent framework: an educational tool introducing the basics of AI programming"*, Proceedings International Conference on Information Technology in Research and Education, pp. 426-430, August 2003.
- [9] <http://www.fleeble.net/>, Last visited at August 25, 2006.
- [10] Fleeble Documentation and Tutorials, available in Fleeble, from <http://www.fleeble.net>, Last visited at August 25, 2006

- [11] M. Pantic, R.-J. Grootjans, R. Zwitterloot, *Fleebly Agent Framework for teaching an introductory course in AI*, Proc. Intl Conf. Cognition and Exploratory Learning in Digital Age, pp. 525-530, Lisbon, Portugal, 2004.
- [12] M. Pantic, R. Zwitterloot, R.-J. Grootjans, "Teaching Introductory Artificial Intelligence Using a Simple Agent Framework", Education, IEEE Transactions on , vol.48, no.3pp. 382- 390, Aug. 2005
- [13] S. Russell, P. Norvig, *Artificial Intelligence: a Modern Approach.*, Second Edition, Prentice-Hall, 1995.
- [14] K. P. Sycara. "Multiagent Systems", AI Magazine vol. 19(2), pp. 79-92, 1998.
- [15] M.K. Sahota, A.K. Mackworth. *Can situated robots play soccer?* Proceedings of Canadian AI-94 (1994), pp. 249-254.
- [16] M. Jamzad, M. Asadpour, "Introducing Simulation Middle Size: A New Soccer League to RoboCup",
- [17] Sony Aibo, Official website, <http://www.sony.net/Products/aibo/>, Last visited at August 25, 2006.
- [18] Aibo SDE - Development environment for writing software for the Sony Aibo, Official website, <http://openr.aibo.com/>, Last visited at August 25, 2006.
- [19] Humanoid Soccer League, Official website, <http://www.humanoidsoccer.org/>, Last visited at August 25, 2006
- [20] S. Behnke, "Humanoid Soccer Robots", Information Technology, vol. 47, pp. 292-298, May 2005.
- [21] S. Behnke, T. Langner, J. Muller, H. Neub, and M. Schreiber. "NimbleRo RS: A Low-Cost Autonomous Humanoid Robot for Multi-Agent Research", Proc. of Workshop on Methods and Technology for Empirical Evaluation of MAS and Multi-robot Teams (MTEE), 27th German Conf on AI (KI2004), Ulm, Germany, 2004.
- [22] G. F. Wyeth, D. Kee, M. Wagstaff, N. Brewer, J. Stirzaker, T. Cartwright, and B. Bebel. "Design of an autonomous humanoid robot", Proc. of the Australian Conf on Robotics and Automation (ACRA 2001), Melbourne, 2001.
- [23] S. Behnke, J. Muller, and M. Schreiber. "Toni: A soccer playing humanoid robot", Proc. of 2005 RoboCup Intl Symp., July 2005.

- [24] Soccer Simulator, Official website, <http://sserver.sourceforge.net/>, Last visited at August 25, 2006.
- [25] I. Noda, "Soccer server: a Simulator of RoboCup", In Proceedings of AI Symposium '95, Japanese Society for AI, pp. 29-34, December 1995.
- [26] http://www.uni-koblenz.de/~tomomi/rc06/rules_2d.20060615.pdf, Last visited at August 25, 2006
- [27] <http://sserver.sourceforge.net/docs/manual.pdf>, RoboCup Soccer Server Users Manual. Last visited at August 25, 2006.
- [28] RoboCup Simulation Library Archive - An overview of available libraries for developing RoboCup software agents, <http://www.ida.liu.se/frehe/RoboCup/Libs/>, Last visited at August 25, 2006.
- [29] O. Obst, M. Rollmann. "Spark - A Generic Simulator for physical Multi-agent Simulations", Proceedings of the MATES 2004, Lecture Notes in Artificial Intelligence, pp. 243-257, September 2004.
- [30] B. Browning, E. Tryzelaar, "Ubersim: A multi-robot simulator for robot soccer", Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 948-949, 2003
- [31] T. Laue, K. Spiess, and T. Rofer, "SimRobot - a general physical robot simulator and its application in RoboCup," in RoboCup 2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Springer, 2005.
- [32] B. Gerkey, R.T. Vaughan, A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.", Proceedings of the 11th International Conference on Advanced Robotics, Coimbra, Portugal, 317-323, 2003
- [33] O. Michel, "Webots: Professional mobile robot simulation", International Journal of Advanced Robotic Systems, Volume 1, Number 1, pp. 39-42, March 2004.
- [34] J.C. Zagal, J.R. del Solar, "UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League.", RoboCup 2004: Robot Soccer World Cup VIII, Lecture Notes in Artificial Intelligence, 2004.
- [35] H. H. Lund, "Robot Soccer in Education.", Advanced Robotics Journal, Volume 13, Number 8, pp. 737-752, 1999.

- [36] J.K. Archibald, R.W. Beard, "*Goal! robot soccer for undergraduate students*", Robotics & Automation Magazine, IEEE, Volume 11, Issue 1, pp. 70-75, March 2004.
- [37] R. Murphy, "*Competing for robotics education*", IEEE Robotics and Automation Magazine, Volume 8, Issue 2, pp. 4455, June 2001.
- [38] E. Sklar, S. Parsons, P. Stone, "*RoboCup in Higher Education: A Preliminary Report*", Lecture Notes in Computer Science, Volume 3020, pp. 296-307, Jan 2004.
- [39] J.K. Archibald, R.W. Beard, "*Competitive robot soccer: a design experience for undergraduate students*", Frontiers in Education, vol. 2, pp.14-19, 2002.
- [40] F. Heintz, J. Kummeneje, P. Scerri, "*Simulated RoboCup in University Undergraduate Education*", Proceedings of the fourth international workshop on RoboCup at the 4th RoboCup World Championships, 2000.
- [41] T. Huang, F. Swenton, "*Teaching Undergraduate Software Design in a Liberal Arts Environment Using RoboCup*", Proceedings of 8th annual conference on Innovation and technology in computer science education, Volume 35, Issue 3, pp. 114-118, 2003.
- [42] J. Kummeneje, "*RoboCup as a Means to Research, Education, and Dissemination*", Licentiate of Philosophy Thesis, Dept. of Computer and Systems Sciences, Stockholm University, 2001.
- [43] F. Heintz, J. Kummeneje, P. Scerri, "*Using Simulated RoboCup to Teach AI in Undergraduate Education*", Proceedings of the 7th Scandinavian Conference on Artificial Intelligence, pp.13-21, February 2001.
- [44] J. Vidal, P. Buhler, "*Using RoboCup to Teach Multiagent Systems and the Distributed Mindset*", Proceedings of the 33rd ACM Technical Symposium on Computer Science Education, February 2002.
- [45] F.Heintz, "*RoboSoc a System for Developing RoboCup Agents for Educational Use*", Master's thesis, Department of Computer and Information Science, Linkoping university, March 2000.
- [46] Agentlink - An overview of agent software, <http://www.agentlink.org/resources/agent-software.php>, Last visited at August 25, 2006.
- [47] J.P. Bigus, J. Bigus, "*Constructing intelligent agents using Java*", New York: John Wiley & Sons, 2001.

- [48] S. Poslad, P. Buckle and R. Hadingham, *The FIPA-OS Agent Platform: Open Source for Open Standards*, available at <http://fipa-os.sourceforge.net>.
- [49] Pathwalker Agent framework, Fujitsu Laboratories, 2000. Official website: <http://www.labs.fujitsu.com/freesoft/paw/>.
- [50] Java 1.5 <http://java.sun.com/j2se/1.5.0/>, Last visited at August 25, 2006.
- [51] Serialization - saving an object to bytes, <http://en.wikipedia.org/wiki/Serialization>, Last visited at August 25, 2006.
- [52] Java Annotations, <http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html>, Last visited at August 25, 2006.
- [53] G. Bracha, *Generics in the Java Programming Language*, <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>, July 2004.
- [54] P. Boucklee, *Java Packages Tutorial*, http://www.jarticles.com/package/package_eng.html, Last visited at August 25, 2006.
- [55] Context Programmer's Editor, <http://www.context.cx/>, Last visited at August 25, 2006.
- [56] The Eclipse Project, <http://www.eclipse.org>, Last visited at August 25, 2006
- [57] CVS - Concurrent Versions System, <http://www.nongnu.org/cvs/>, Last visited at August 25, 2006.
- [58] Ant - Java-based build management tool, <http://ant.apache.org/>, Last visited at August 25, 2006.
- [59] GNU Make - build management tool, <http://www.gnu.org/software/make/>, Last visited at August 25, 2006.
- [60] SWT - Standard Widgets Toolkit, <http://www.eclipse.org/swt/>, Last visited at August 25, 2006.
- [61] AWT - Advanced Widgets Toolkit, first-generation graphical toolkit for Java, <http://java.sun.com/products/jdk/awt/>, Last visited at August 25, 2006.
- [62] Swing - The Swing Tutorial, second-generation graphical toolkit for Java, <http://java.sun.com/docs/books/tutorial/uiswing/index.html>, Last visited at August 25, 2006.

- [63] Wikipedia - KISS - The *Keep It Simple, Stupid* principle, http://en.wikipedia.org/wiki/KISS_Principle, Last visited at August 25, 2006.
- [64] Wikipedia Soccer, Explanation of the laws of the game of soccer, [http://en.wikipedia.org/wiki/Football_\(soccer\)](http://en.wikipedia.org/wiki/Football_(soccer)), Last visited at August 25, 2006.
- [65] C.E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, July, October, 1948.
- [66] J. Kok, R. de Boer, *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*, Master's Thesis, University of Amsterdam, February 2002.
- [67] Mox - Mapped Object XML, A simple Java 1.5-based XML parser developed by Reinier Zwitserloot, <http://www.zwitserloot.com/java-boilerplate/mox/tutorial.html>, Last visited at August 25, 2006.
- [68] Nullsoft Scriptable Install System, <http://sourceforge.net/projects/nsis/>, Last visited at August 25, 2006.
- [69] T.M. Duffy, J. Lowyck, D.H. Jonassen, "*Constructivism: New Implications for Instructional Technology*", *Constructivism and the technology of instruction: a conversation*, pp. 1-16, 1992.
- [70] J.B. Black, R. McClintock, "*An interpretation construction approach to constructivist design*", *Constructivist Learning Environments: Case Studies in Instructional Design*, B.G. Wilson, Ed. Englewood Cliffs, NJ: Educational Technology, 1995, pp. 25-31.
- [71] R. Lister, J. Leaney, "*Bad theory versus bad teachers: Toward a pragmatic synthesis of constructivism and objectivism*", *Proceedings of the International Conference of Higher Education Research and Development Society of Australasia Inc.*, 2003.
- [72] S. Hadjerrouit, "*A constructivist approach to object-oriented design and programming*", *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, pp. 171-174, 1999.
- [73] J.R. Savery, T.M. Duffy, *Problem based learning: An instructional model and its constructivist framework*", *Educational Technology*, Volume 35, Issue 5, pp. 31-38, 1995.
- [74] C.E. Hmelo-Silver, *Problem-Based Learning: What and How Do Students Learn?*, *Educational Psychology Review*, Volume 16, Issue 3, pp. 235-266, 2004.

- [75] Gantt Charts, http://en.wikipedia.org/wiki/Gantt_chart, Last visited at August 25, 2006.
- [76] Roshambo - Rock Paper Scissors, http://en.wikipedia.org/wiki/Rock,_Paper,_Scissors, Last visited at August 25, 2006.
- [77] Forward Chaining, http://en.wikipedia.org/wiki/Forward_chaining, Last visited at August 25, 2006.
- [78] The Internet Movie Database, <http://www.imdb.com/>, Last visited at August 25, 2006.