RoboCup Soccer - The Team Fundamentals

KTH - Royal Institute of Technology, CSC
DD143X - dkand13, Group 10
Supervisor: Pawel Herman

Langesten, Daniel
langest@kth.se
073-647 54 08
Love Almqvists Väg 4b
112 53 Stockholm

Norberg, Jesper
jenor@kth.se
073-783 16 02
Jenny Linds Gata 7
129 52 Hägersten

2013-05-29

Abstract

The RoboCup 2D Soccer Simulator is an international project for the development of multiagent systems and artificial intelligence in which programmed teams compete in simulated soccer.

This report aims to answer the question "What is a basic team?". In order to do this, it first identifies and analyses the key components and functionalities of basic teams. Further, it presents the work of creating a fully functional team with these fundamentals implemented. In order to evaluate the created team various test matches were set up, with their results analysed. Finally the report discusses possible future work.

The results from the evaluation shows that the created team is of a functional but basic standard, which suggests that the team fundamentals were well defined and that the team is a solid base for further development.

Sammanfattning

RoboCup 2D Soccer Simulator är ett internationellt projekt för utvecklingen av multiagenta system och artificiell intelligens i vilket programmerade lag tävlar i simulerad fotboll.

Den här rapporten syftar till att svara på frågan "Vad är ett grundläggande lag?". För att göra detta identifierar den först nyckelkomponenterna och funktionaliteterna för grundläggande lag. Vidare redogör den arbetet i att skapa ett fullt fungerande lag med dessa fundamentala byggstenar implementerade. För att evaluera det skapade laget anordnades även diverse testmatcher, vars resultat analyserades. Slutligen diskuterar rapporten möjligt framtida arbete.

Resultatet från evalueringen visar att det skapade laget håller en funktionell men grundläggande standard, vilket tyder på att de fundametala byggstenarna var väldefinierade och att laget är en solid grund för fortsatt utveckling.

Contents

1 Introduction

The Robot World Cup Initiative (RoboCup) was made in order to promote artificial intelligence and multiagent systems research. RoboCup is an organisation for multiple robot projects, but the most prominent one is RoboCup Soccer, where the vision is to create a robot soccer team able to compete with human teams by the year 2050. Through providing a problem for which a broad range of technologies can be applied, RoboCup Soccer has created a very useful testing environment for students and researchers alike.[1]

This report will focus on the RoboCup 2D Soccer Simulator (RCSS), which is the oldest branch of RoboCup Soccer, and which focuses on artificial intelligence and team strategy.[2] The progress made using the RCSS has contributed greatly to the fields of real time multiagent planning, communication methods, collaborative sensing and multiagent learning.[3]

When approaching the subject, one will be struck by the vast amount of knowledge contained within the field. There is a multitude of reports and team descriptions, most written in great detail, spanning hundreds of pages. Despite that, there is a topic which is surprisingly lacking: What is a basic team? We believe this question is of importance for those who simply want an overview of the RCSS world.

The aim of this report is therefore to identify and analyse the key components and functionalities of a basic team, and how are they can be applied during the creation of a new team.

1.1 Objectives

The primary objective of this report is to identify and analyse the team fundamentals of the RoboCup Soccer Simulator. We define the team fundamentals as the key components and functionalities required to create a basic RoboCup Soccer team which can move and play at a basic level whilst following the game rules. A basic team should be able to perform tasks such as moving players around, passing the ball between players, and scoring goals.

The secondary objective is to implement the identified features in a team of our own. We do this in order to ascertain the validity of our primary objective. We will then try to determine our team's proficiency through letting it play test matches and analysing the results, followed by a discussion regarding different ways of improving and extending the team's functionality and tactics.

1.2 Approach

In order to reach our objectives there is an initial need to study and understand the mechanics of the RCSS, to create a client that will be compatible with it. Various reports regarding the RCSS as well as the source code for existing teams will also be studied in order to get an idea of how the functionalities are commonly implemented,

what the principles of a team are and which kind of tactics they tend to use. The information obtained will then be analysed and applied in the creation of our own team. To determine how proficient this team is, we will let it play in simulated matches against established teams and then analyse the results. Lastly, for the discussion regarding future improvements, we will bring up more advanced features we did not implement in the current version of our team.

We have decided to use the programming language Java for the programming of the team, mainly because it is the language in which we both are the most proficient. Also, Java has a multitude of classes available for network communication, which is a cornerstone for any functioning team. The downside of course is that Java is not an optimal language for tasks where it is important to do complex calculations quickly, and as such languages like C or C++ would prove more effective.[4] However, since the goal is not to create a high level team, confidence in the code we wrote was deemed more important.

1.3 Scope

We define the fundamentals as the base components and functionalities required to create a basic RoboCup Soccer team which can move and play at a basic level whilst following the game rules. As such, advanced features and systems like coaching, reinforced learning and probabilistic localization is therefore not included in the definition. Also, the secondary goal is to implement the team fundamentals, not to optimise them. We will therefore not spend an excessive amount of time optimising the team. There is also a more complex ruling system than the one described in section 2.4 - just as in real soccer - which is not taken into account as there is no automated way to enforce these rules. In tournament matches there is normally a human referee judging potential rule violations of this sort.[5]

As the goal of this report is to discuss the team fundamentals and to give an overview of a potential program structure, rather than to provide a detailed guide of implementation, we will not bring up algorithm implementations or server data handling. We will also refrain from going into detail regarding how the physics motor in the RCSS works, eg. how the acceleration or deceleration of the ball or players is handled.

1.4 Guide to report

In section 2 we give the reader a brief overview over how the different areas of RCSS works. This includes the server, the monitor, the physics model, the action commands, and the rule system.

In section 3 we discuss what we consider to be the fundamental aspects that needs to be taken into account at the creation of a RCSS agent, as well as discuss how we have implemented these key features in the creation of our team. We also bring up related work that has helped us in the creation process.

In section 4 the results of the test matches will be presented. We will also summarize the matches and discuss the outcomes. For ease of distinction we gave our team a mock name: Limeonds.

In section 5 we will discuss which components of the team were successful, as well as which areas could be improved in order to heighten the standard of the team. We also present an evaluation of our work.

In section 6 we present our conclusions regarding how well we fulfilled the project objectives.

In this report, the terms agent and player will be used interchangeably. Also, though the players are genderless, we will refer to them in masculinum for ease of reading.

2 Background

In this section we give the reader a brief overview over how the different areas of RCSS works. This includes the server, the monitor, the physics model, the action commands, and the rule system.

2.1 The server

The RCSS is a multiagent environment that supports two teams of 11 independent agents interacting in real time in a dynamic environment. Because of RoboCup's vision, to be able to contend against humans by the year 2050, the simulator tries to simulate real world limitations. These include limited vision and hearing, object movement not being perfectly predictable and player stamina being dependent on his actions.[3]

The server is using a specific protocol which each client needs to follow. All communication between the server and client is performed by network using a UDP/IP socket. As such, the client can be coded in any language, as long as it can send and receive UDP/IP signals. One thing which is important to realize is that a client in the RCSS controls no more than one player, and that a team therefore needs 11 separate clients. Each client also needs to have an open connection toward the server. This is accomplished by making the client send a message containing his team affiliation to a specific server port. The server will then respond to the message, providing the player with necessary base information such as the team side, uniform number, as well as a unique port of communication to be used between that player and the server.

During gameplay, the client will be able to receive sensory information regarding its player state, and also gain the ability to respond by sending control commands to the server. These will be discussed in further detail in sections 2.3.2 and 2.3.3 respectively.

More information regarding the server as well as exact server commands can be found in its creator's - Itsuki Noda - report on the matter.[6]

The simulation is run in discrete time, which means that the flow of time is not continuous. Instead you have timed intervals during which you can perform actions. The intervals (cycles) are 100 milliseconds long which in turn gives the client 100 milliseconds to tell the server its control commands. If the server does not receive any command in time, the player will remain stationary for that cycle.

2.2 The monitor

A monitor is the visualisation tool where the simulation is shown. There are multiple implementations of monitors, though the official one is the most commonly used. The information displayed on a monitor includes the current standing, the team names, where the players are and where the ball is. It also offers simple interface options, "kick-off" for example allows one of the spectators to start the game. Depending on the visualisation tool that is being used there are also a multitude of tools which primarily have a debugging role. Examples include zoom, print-outs and manual control of the ball. However, having a monitor is not necessary in order to simulate a game.[5]

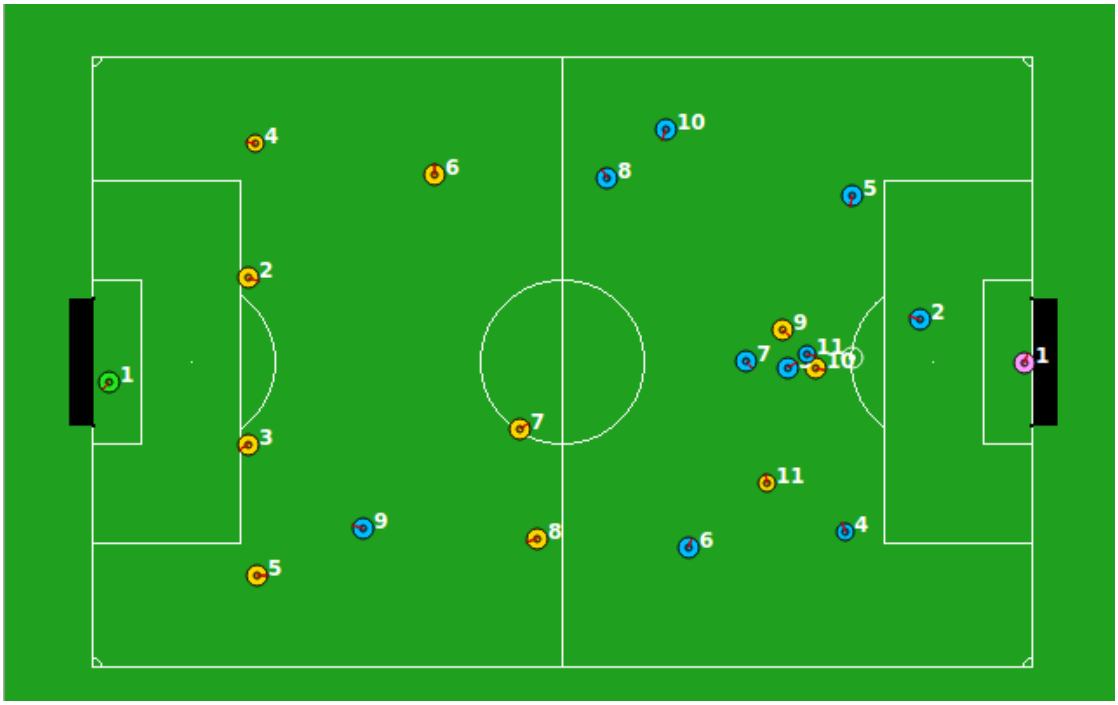

Figure 1: A screenshot of the monitor during a match.

2.3 Physics model

Here we cover two main areas of the RCSS physics model; the field setup and the player's sensory data.

2.3.1 The field

The field is based upon a coordinate system. This system has its origo in the center of the field. The x- and y-axis are two perpendicular axes which increases rightwards horizontally and downwards vertically. Around the field, there are marked positions called flags, representing static positions which are of interest, such as goals and lines. These are then used by the players to navigate the field, which is discussed in further detail in section 3.1.3. A detailed field map displaying the flag coordinates can be found in a master's essay written by C. Rahm.[7]

2.3.2 Sensors

A player needs to have the ability to gather information of its environment in order to be able to decide what action is fitting in the current situation. This information is received by the player as sensory data. The messages received by a player from the server contains information from a body sensor, a visual sensor and an aural sensor. The

only ones necessary for the basic team however is the body sensor and the visual sensor, which are therefore the only ones we will discuss.

The body sensor provides data regarding the player's current body information, such as body angle, current speed, and current stamina. Stamina is consumed when moving and when it runs out, the player is only able to move slowly until he stops and rests.

The visual sensor provides information regarding what the player can currently see. This includes flags, players and the ball. The vision is of varying quality depending on how close to the center of the player's view it is, simulating peripheral vision. As such, the information is unreliable unless the player is looking at something directly. This leads to complications when trying to identify teammates. Another issue regarding the information retrieved is that it doesn't provide you with the coordinates of the objects seen. It only gives the player information regarding the objects location relative the players location. However, the player will not receive any information regarding his own position either. This results in a situation where the player needs to calculate using fixed positions in his vision when he wants to know his exact location. Doing this properly is highly important for proper agent behaviour, as discussed in a report by the Information Sciences Institute and Computer Science Department of the University of Southern California.[8]

2.4 Action commands

2.4.1 Overview

A player has several possible action commands which he can use to communicate his intention to the server, which will then in turn affect the player's surroundings. The ones which we are using are: turn, dash, kick, move, and catch. There are a few more, but the ones mentioned are the only ones necessary for the creation of a basic team. More detail regarding the commands can be found in the manual for the server.[5]

2.4.2 Turn

The turn command, is used to change the direction the player is facing. It takes one argument; the desired moment change in degrees. Depending on the speed the player is currently moving forward, he will have a set maximum for how much he can turn in one cycle. Turning is done at the expense of running, leading to a minor decline in movement speed.

2.4.3 Dash

Next is the dash command, which is used to make the player run. Only one argument is taken, the power which with the player wants to move forward. Depending on the current momentum, the player will then accelerate, maintain speed, or decelerate. Dashing also consumes stamina, as covered in section 2.3.2.

2.4.4 Kick

The kick command is used to kick the ball forward. It takes two arguments; the power and the desired direction in degrees. The current movement speed will affect the kick's power and direction, with a higher movement speed increasing the kick's power but reducing the potential for variation in direction. Kicking is used to move the ball forward, to pass other players, and to shoot at goals. Each task require a different approach to the kicking, which will be covered in 3.2.4.

2.4.5 Move

Next is the move command. Two arguments are taken; an x-coordinate and a y-coordinate. These are then used to instantly move the player to that specific position. This is a conditional command only usable at certain game states, such as right before a kick-off, or after a goal is scored.

2.4.6 Catch

Finally the catch command, which is usable only by the goalkeeper. It takes only one argument - a direction in degrees - in which the goalkeeper will attempt to catch the ball. If the ball is in the immediate area in front of the goalkeeper, he will succeed in catching it. He is then free to kick the ball out into the field again.

2.5 Rules

There are game rules which are automatically enforced and which we therefore need to take into account.[9] Each of these is described below.

2.5.1 Kick-Off

All players in a team must be on their respective side of the playing field at the start of a kick-off. When a kick-off is about to take place, all players are provided with the ability to teleport to a position of their choice using the "move"-command. This should be used in order to preserve stamina which otherwise goes wasted if the player runs using the "dash"-command.

2.5.2 Goal

When a goal is scored the scoreboard will update, and the match will is interrupted for 5 seconds. During this time interval players may prepare for a new kick-off.

2.5.3 Field-Out

When the ball goes outside of the playing field it is moved to a suitable position, i.e. the line marking, the corner, or the goal area - just as in regular soccer. The game state

then changes to kick-in, kick-out or goal-kick respectively. After this game state change it returns to the normal state.

2.5.4 Free Kick

When a free kick occurs the referee moves the other team's players to a set distance from the ball. The team that received the free kick is then free to kick the ball. It is not allowed to do a pass to yourself when making a free kick.

2.5.5 Offside

A player is marked as offside if he is on the opponents side of the field and there are less than two defenders who are closer to the goal than the player. When marked as offside a player cannot kick the ball.

2.5.6 Back-Pass

If a field player passes his goalkeeper then the latter may not catch the ball with his hands.

2.5.7 Half-Time

After 5 minutes (3000 cycles) half-time is called. The teams then switch sides.

2.5.8 Time-Up

The game ends after the second half, provided that a team is in the lead. Otherwise the match goes on until either team scores a goal.

3 Methods

In this section we discuss what we consider to be the fundamental aspects that needs to be taken into account at the creation of a RCSS agent, as well as discuss how we have implemented these key features in the creation of our team. We also bring up related work that has helped us in the creation process.

3.1 The team fundamentals

Here we discuss the fundamental aspects we have identified, and which needs to be taken into account at the creation of a RCSS team.

3.1.1 Basic RCSS interaction

An agent in a basic team needs to possess the ability to interact with the RCSS in a way that takes the areas covered in section 2 into full account. This includes correct interaction with the server through UDP/IP, taking the RCSS physics model into account, implementing the basic action commands, and following the rules.

3.1.2 World model

Each agent is given its own world model, which represents the player's interpretation of the current playing field. The interpretation of static objects will naturally match the actual playing field, but the moving objects (players and ball) will be difficult to localise properly. As such, the world model can only be moderately accurate as the agent is unable to constantly scout his surroundings.

3.1.3 Navigation

The players are not provided with any information regarding their own coordinates and therefore need to calculate their position relative fixed landmarks that they currently see. This also applies to all non-stationary objects in the players current vision, as they only get the objects coordinates relative themselves.

3.1.4 Ball handling

There are several different types of ball handling, each needing its own approach. After locating the ball and reaching it, the player should use its strategic logic to determine which of these is the most suitable. If teammates are within sight, the player can calculate an estimate of where it needs to kick the ball in order to successfully pass. Scoring works in the same manner, except that the player then instead needs to see the opponent goal. Also, in that case, the opponent goalkeeper should be taken into consideration. The standard maneuver however is dribbling, which is simply a weak kick, moving the ball a short distance forward. If properly implemented, this can be of great strategic value. As for intercepting, you simply need to get to the ball before your opponent and kick it before he does. How a goalkeeper uses the catch command is covered in section 2.4.6.

3.1.5 Strategic logic

This represents the need for the agent to decide on a fitting action at any given situation. To do this it is often wise for a player to analyse his surroundings and use logic to decide which action is the most fitting for the current situation. Predefining common situations - such as when to shoot for goal - will facilitate this area.

3.1.6 Player types

A common implementation method is to define different player types. This is done in order to create a more diverse team, able to adapt to different situations. Each role is typically given a certain set of action patterns and responsibilities. The amount of roles can vary, but having a set role for the goalkeeper is fundamental.

3.1.7 Formation

The team formation is central to the team strategy, as it defines positions and therefore playstyles for the team's players. High end teams tend to have multiple formation possibilities which they can switch between depending on the current state of the game.

3.2 Implementation

Here we briefly discuss our implementation of the earlier established fundamental aspects.

3.2.1 Basic RCSS interaction

For the client to easily interact with the server the client needs an easy way to receive and send messages. We have decided to provide each client with a parser object that have the ability to send messages to the server as well as receiving them and parsing the messages received from the server. We have also fine tuned the use of command variables, so that a pass has the right power to be received etc. We've also taken the rules described in section 2.5 into account.

3.2.2 World model

Despite the difficulty in providing an accurate world model, we have through constant updating of all visible movable objects established a functioning implementation. In our design, each client uses its own world model in order to keep track of its surrounding. We therefore equip it with a world model object that is able to keep and process sensory data.

3.2.3 Navigation

To navigate we have chosen to create a component which takes advantage of the player's world model. It asks for the position of known flags and uses these to calculate the

player's current position, from which it can estimate the location of other moving objects. The calculations are primarily done using Pythagoras theorem.

3.2.4 Ball handling

In our implementation, we have a finely tuned passing system, estimating the speed of the receiving player in order to optimise the pass. When shooting, our players shoot toward the closest area of the goal. As for interception, our players simply attempt to steal the ball if it is close enough. The ball handling of our goalkeeper is covered in section 3.2.6.

3.2.5 Strategic logic

Our clients each have their own logic component which tells them what action they should perform in each given situation. These logics differ depending on which type the player has. More on types in section 3.2.6. In order to preserve stamina, we have put high priority on a passing game. Our main strategy is that when a field player comes into possession of a ball, he has a tiered list of priorities. The first priority is to shoot on goal if able. If not, the next priority is to pass a teammate further ahead. If none is in sight, the player will simply dribble forward toward the opponent goal. In practice, as the forwards are the ones furthest ahead, they are the ones being the most aggressive. This is a very basic teamwork strategy that helps bringing the ball towards the opponents goal in a fast and reliable way. The strategy for our goalkeeper is covered in section 3.2.6.

3.2.6 Player types

We decided to split the players into different fixed roles, to get a more specialised team. The established roles have different action patterns and responsibilities. We used four roles for our team: forward, midfielder, back, and goalkeeper.

Forward The role of the forward is to locate the ball, to acquire it, and to shoot for the opponent goal if within range. If the goal is not within range, he will instead try to pass a team member further ahead. If that is not possible either, he will move with the ball toward the goal.

Midfielder The midfielder currently has the same strategy as the forward, except that he stays further back.

Back The back is more stationary than the previous players. When he find the ball he does not instantly move in and try to get it. First he checks if it is close enough for action to be beneficial. We designed them like this in order to reinforce the wall-like back structure we are using, which we cover in section 3.2.7. Other than this they function just like the forward and midfielder.

Goalkeeper The goalkeeper always stays within the goal area, and tries to align himself between the goal and the ball until the ball is close enough. He then runs toward it and tries to catch it. After grabbing the ball, he will immediately pass the first teammate he sees. If he does not see any teammate he will just kick it away from the goal.

3.2.7 Formation

As we are only interested in the basic functionalities, we decided to limit ourselves to one fixed formation, clearly defining each position with a set role. We are using three forwards, three midfielders, four backs and one goalkeeper. The setup is displayed in Figure 2.

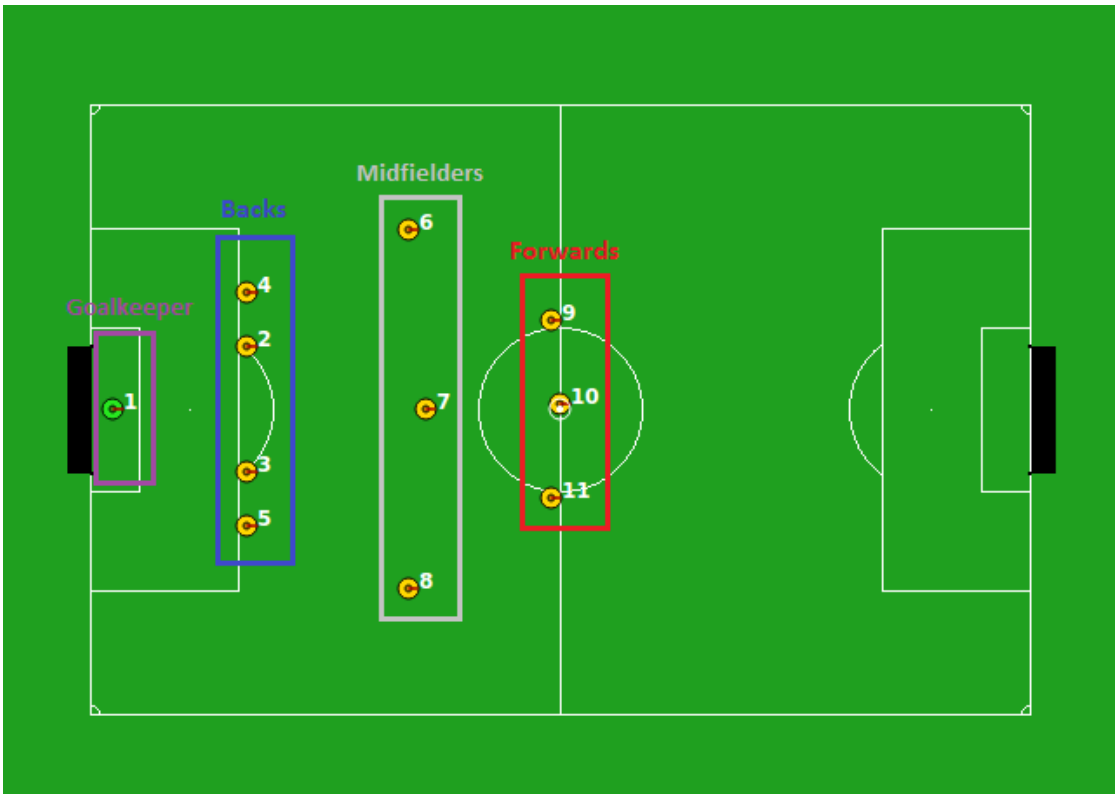

Figure 2: The formation of our team.

3.3 Related work

This section will discuss related reports we have found particularly useful in the task of identifying the team fundamentals and different ways to implement them.

3.3.1 Trilearn

Trilearn is a RCSS client developed by the Intelligent Autonomous Systems Group of the University of Amsterdam. The team has scored 4th place in the world championship, and is well documented. We have found inspiration in various areas, but especially their approach to object localization and velocity estimation has proved helpful.[9]

3.3.2 Sebbot

Sebbot is a RCSS client written by Sébastien Lentz as the project for his Master's thesis. What makes the accompanying report so useful is that it includes much background information regarding the server, which he found through studying the server source code. This proved helpful for the creation of our team.[10]

3.3.3 Biter

Biter is a RCSS client developed by the professors Paul Buhler and José M. Vidal from the University of South Carolina. What makes Biter special is that it is developed in Java. It was therefore very useful in helping us find the right classes and data structures for the creation of our team.[11]

4 Results

In this section the results of the test matches will be presented. We will also summarise the matches and discuss the outcomes. For ease of distinction we gave our team a mock name: Limeonds.

4.1 Limeonds vs. TeamSkynet-2011

TeamSkynet is a Robocup Soccer Team developed in Java by students at the University of Kansas during the year 2011. It took part in a tournament hosted by the university and is the only team we found from that tournament that was open source.[12] We were therefore able to compile the team and challenge it.

The results were 26-0. A hands down victory to our team. TeamSkynet had much difficulty in handling our passes, and its players became quite lost upon losing sight of the ball. Another major problem TeamSkynet had during the game was that its players occasionally ran past our players without a clear purpose. We believe that they have not succeeded in implementing the basic features needed for a team which their dislocation on the game field also suggests. Another reason as to why we believe they struggled is that their team was written in Java just like ours. As mentioned in section 1.2, Java is a friendly language, but it requires more processor power for heavy calculations than a programming language closer to the hardwrre like C.

4.2 Limeonds vs. WEBase-2013

The WEBase Robocup Soccer team is written in C++ and belongs to the Multi-Agent Systems Lab of the University of Science and Technology of China. It is the school's official team, and they have both professors and Ph.D. students working on it. Since it was created in 1999 it has won 3 world championships.[13]

The results were 0-37. We did not stand a chance against them. Compared to our team, their superior formation system became very obvious. They were able to move in a grid formation at all times, and coordinated their passes in a way which outshone ours. Another reason for their superiority was their fast decision making and higher action count, which we believe can be credited to them having better synchronised time intervals with the server, better algorithms, and to their code being written in C++.

4.3 Limeonds vs. Limeonds

We wanted to see if our team was able to play a reasonable soccer game against itself. This in order to give us an indication of how it would perform against a team of the same level.

The result was 3-3. The resulting score was to be expected. Since it was the same team facing itself it is not surprising that the two sides scored an equal amount of goals. However, the game was a strange one if compared to real soccer. Both sides were chasing the ball with pretty much every player that was aware of the ball's location. This is most likely due to that we have not implemented any way for the players to plan ahead. Therefore, each player will, other than passing other players ahead of him, try to play the game by himself.

4.4 Limeonds vs. Liponents

We then wanted to test playing against our own team again, but with a different formation. This was done in order to see how much a slight change in team formation would affect the results. The changes simply consisted of spreading the players out a bit more. The difference is illustrated in figure 3. We mock-named this new team Liponents.

The results were 14-1. As expected we got a different result from when we put two exact copies of the team up against each other. That the change in result would be so dramatic was not expected however. Both teams used the same strategy, but the wall-like formation of the backs in Limeonds proved to be very efficient in preventing Liponents charge from succeeding. It seems that in basic teams a compact defense is more beneficial.

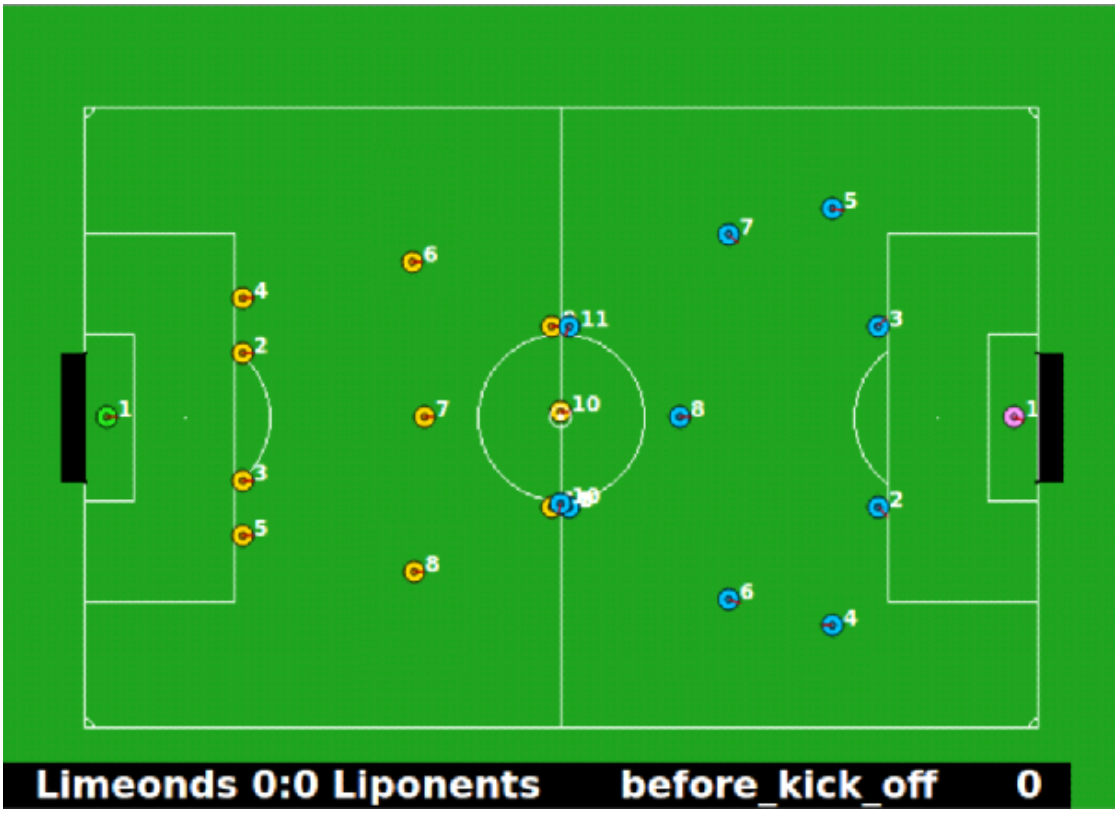

Figure 3: A screenshot of our team's formation, on the left side, with an alternative formation, to the right, which we constructed to evaluate how important the formation was for the outcome of a match.

5 Discussion

In this section we will discuss which components of the team were successful, as well as which areas could be improved in order to heighten the standard of the team. We also present an evaluation of our work.

5.1 Successful components

Here we discuss the features in our team which proved to be useful throughout the test matches.

5.1.1 Basic functionelities

We were successful in implementing the basic functionalities. Our players had no issues in interacting with the server. Further, they were able to adapt to the RCSS physics model completely, using all of the basic action commands from section 2.4 in a satisfiable manner. The players also followed the rules, understanding the different game states.

5.1.2 Passing system

Though our passing system was quite basic, it was still quite successful against Team-Skynet, the only other student team. It is very hard to predict how opponent players are going to pass, and so a strategy focusing on passing is a very effective approach when facing off against basic teams. A major reason for the success of our passing system is that our players were able to correctly identify their teammates and calculate the passing power needed to kick the ball as far as necessary.

5.1.3 Team formation

The importance of formation exceeded our expectations. Our centralised back system proved to be very effective in stopping opponent approaches. As it is difficult to implement a complex interception system, a basic team only focuses on intercepting the ball when it gets close. Also, when creating an attack strategy, it is hard to design it to attack from the sides. As such, placing the backs in the middle means that the opposing team's charge is likely to pass by the backs, increasing the chances of interception.

5.2 Areas to improve

Here we discuss the areas we feel should be addressed if we are to develop the team further.

5.2.1 Logic optimization

Although our players are able to pass each other successfully, there is no joint tactic behind it. We lack strategized attack patterns. This is one of the reasons that makes our team inferior to elite teams, as observed in the match against WEBase. Another simple logic optimization would be the ability to see if an opponent is closing in and to then decide whether or not it is safer for the player to pass the ball than to try and continue on its own.

The team also lacks fast decision making. Perhaps this is much due to the Java limitations, but WEBase made very quick decisions regarding whom to pass and when, with no player ever being inactive. Another contributing factor to faster decisions would be optimised server synchronisation.

5.2.2 Probabilistic localization

The world model information can be used to probabilistically calculate where each object is estimated to be at a certain point in time. This is possible when object information such as velocity and latest viewed position are taken into consideration. Additionally, because of the timestamp that comes bundled with all sensory information, it is possible to provide a confidence value, telling the agent to which degree the information is reliable. Based on this, it is possible to let players make more rational decisions regarding when and where to perform an action. This is a common feature in elite teams, and its

implementation has been covered in a report from the institute for systems and robotics in Lisbon, Portugal. [14]

5.2.3 Reinforced learning

The principle of reinforced learning is to have the team learn by experience even after the programmers are finished writing it. This proves useful as the programmer usually cannot foresee all of the potential situations that can arise. The main idea is to have agents try different things and then reward them if it the outcome is good. This will result in the agents learning what is good to do in different situations. For example, a variety of scenarios of kicking toward goal can be presented. The player will then try different approaches, and register which approaches worked and which did not. He will then be more likely to shoot for goal using the successful approaches. It is important that the reward system is properly implemented so that the players do not learn bad behaviour, which could lead to their gameplay deteriorating.

5.2.4 Communication and flexibility

A key component in any high end team is good communication. In RCSS, communication between players is possible, however it is delayed, has some background noise added to it and is often associated with a cost. When a player wants to communicate it either broadcasts a message to all other players or to one specific agent in the form of a direct message.[5] Communication can be used to spread information regarding the current world model. However, communication can also be used to alter strategies.

Our team has a fixed strategy, but ideally the team roles and positions should change depending on the behaviour of the opponents. This of course requires not only communication, but also coordination. To accomplish this, one could start using the coach. As mentioned in section 1.3 we omitted the coach functionality for our team, but when properly implemented, the coach can watch the game, analysing the opponents strategy. He can then issue commands to friendly team's players, which will then adapt accordingly, possibly changing their formation or player role. This allows for a more flexible team setup, which is beneficial in high end tournament play.[9]

5.3 Evaluation

As TeamSkynet is the only team we found that was not created for a master's degree or by an entire university department, and as it was published as a successful project, we feel it is representative for what a basic student project level is. Since we beat them by quite some margin, we feel that our team is of an acceptable standard. The great loss against WEBase shows however that a fundamental team is far from elite. We were surprised by how big a role the formation played, and the huge difference in scores that resulted from such a minor edit. The results from our test matches suggests that a superior team often wins by very large margins.

The approach we chose proved to be very fitting for our project. This is largely due to RCSS being such a large subject, containing extreme amounts of information. As

such, it helped considerably to research extensively in order to make sure that we did not interpret things incorrectly. The choice of language however hindered our team from being as good as it could have been. As mentioned in section 1.2, Java is not an optimal language for tasks where it is important to do complex calculations quickly, due to its inherently slower calculation speed.[4] As such, if one wants to establish a base team to develop further, it may be wise to write it in C or C++ rather than Java.

6 Conclusion

Here we present our conclusions regarding how well we fulfilled the project objectives.

The primary objective of this report was to identify and analyse the team fundamentals of the RCSS. We defined the team fundamentals as the base components and functionalities required to create a basic RCSS team which could move and play at a basic level whilst following the game rules. We have concluded these fundamentals to be the implementation of: basic RCSS interaction, navigation, ball handling, strategic logic, player types, and formation. As such, we have satisfied the primary objective.

The secondary objective was to implement the identified features in a team of our own, in order to ascertain the validity of the primary objective. We were then to try to determine the team's proficiency as well as discuss various ways of improving and extending the team's functionality and tactics. The team we have created functions properly with all the identified fundamentals implemented. As an evaluation of the test matches showed that our team is at an acceptable level, we conclude that the fundamentals established were reasonable. Further improvements that we have identified that could be implemented to improve on the team's functionality and tactics are: logic optimization, probabilistic localization, reinforced learning, and improved communication and flexibility. Through all this, we have also fulfilled the second objective.

Bibliography

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. *Robocup: The Robot World Cup Initiative*, In proceedings of the IJCAI-95 Workshop on Entertainment and AI/AIife. 1995.

[2] The official RoboCup webpage. Retrieved 2013-04-21 from http://www.robocup.org/

[3] Itsuki Noda, Peter Stone. *The RoboCup Soccer Server and CMUnited Clients: Implemented Infrastructure for MAS Research*, Autonomous Agents and Multi-Agent Systems Volume 7 Issue 1-2, 2003

[4] Andy Georges, Dries Buytaert, Lieven Eeckhout, *Statistically Rigorous Java Performance Evaluation*, In proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. 2007.

[5] J. Murrat, I. Noda, O. Obst, P. Riley, T. Stiffens, Y. Wang, X. Yin, *RoboCup Soccer Server User Manual: for soccer Server 7.07 and later*, 2002.H. Simpson, Dumb Robots, 3rd ed., Springfield: UOS Press, 2004

[6] Itsuki Noda. *Soccer Server: a simulator of RoboCup*, In JSAI AI-Symposium 95: Special Session on RoboCup. 1995.

[7] Christian Rahm, *To create softwareagents for RoboCup*, Naturvetenskapliga fakulteten, Lunds universitet, 2001

[8] S. Marsella, J. Adibi, Y. Al-onaizan, G. A. Kaminka, I. Muslea, M. Tambe, *On being a teammate: Experiences acquired in the design of RoboCup teams*, Proceedings of the Third Annual Conference on Autonomous Agents. 1999.

[9] Remco de Boer, Jelle Kok. *The Incremental Development of a Synthetic Multi - Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*, Master's thesis, University of Amsterdam. 2002.

[10] Sébastien Lentz, *Design of intelligent agents for the soccer game - Sebbot*, University of Liège - Faculty of Applied Sciences, Montefiore Institute, 2010

[11] Biter: A Robocup Client. Retrieved 2013-03-25 from http://jmvidal.cse.sc.edu/biter/

[12] TeamSkynet's repository at github. Retrieved 2013-03-26 from https://github.com/TeamSkynet

[13] WEBase's official webpage. Retrieved 2013-03-26 from http://wrighteagle.org/2D/

[14] J. Hoffmann, M. Spranger, D. Göhring , M. Jüngel, *Exploiting the Unexpected: Negative Evidence Modeling and Proprioceptive Motion Modeling for Improved Markov Localization*, RoboCup 2005: Robot Soccer World Cup IX. 2006.