FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Multi-robot Coordination using Flexible Setplays: Applications in RoboCup's Simulation and Middle-Size Leagues

Luís Henrique Ramilo Mota

PROVISIONAL VERSION

Programa de Doutoramento em Engenharia Informática

Orientador: Luís Paulo Reis (Professor Doutor)

Co-orientador: Nuno Lau (Professor Doutor)

Dezembro de 2011

Resumo

O RoboCup é uma iniciativa de âmbito mundial, que tem como objetivo incentivar a investigação e desenvolvimento aplicados a problemas complexos, mas bem conhecidos. A iniciativa RoboCup tem no seu seio domínios de aplicação heterogéneos e complementares, sendo que o futebol robótico permanece, desde o início, a área mais desafiante e popular. O futebol está organizado em várias ligas, com diferentes características, onde a cooperação entre robôs é sempre um desafio primordial.

A coordenação multi-agente e o planeamento estratégico são dois dos principais tópicos de investigação no contexto do RoboCup. Porém, inovações nestas áreas são frequentemente desenvolvidas e aplicadas apenas a uma das ligas RoboCup, não exibindo assim um nível de generalização suficientemente alto. Além disso, apesar de o conceito de jogada estudada (Setplay) já ter sido reconhecido, por muitos investigadores, como importante para a organização do comportamento de uma equipa, não foi ainda criada, no âmbito do RoboCup, nenhuma ferramenta para o desenvolvimento e execução de jogadas estudadas genéricas. No sentido usado nesta tese, uma jogada estudada é um plano livremente definível, com número variável de participantes e argumentos, bem como várias fases ou passos, possivelmente executados em alternativa.

Esta tese apresenta uma ferramenta que permite a definição e execução de jogadas estudadas, aplicável a qualquer liga cooperativa do RoboCup, ou em domínios similares. Esta ferramenta é baseada numa linguagem padrão, independente da liga e flexível, que permite a definição de jogadas estudadas, que poderão ser executadas em tempo real, usando comunicação entre os vários robôs participantes. Foi também desenvolvido um algoritmo de seleção em tempo real de jogadas, inspirado pelo Raciocínio Baseado em Casos (CBR - "*Case Based Reasoning*"). Esta ferramenta regista as condições de aplicação de execuções passadas das jogadas estudadas, e usa esta informação para escolher, em cada momento, entre as jogadas estudadas possíveis. Esta estrutura de jogadas estudadas pretende ser uma ferramenta para a rápida prototipagem de planos multi-agente, permitindo uma adaptação expedita aos adversários, podendo explorar as suas fraquezas e obtendo, consequentemente, uma vantagem competitiva.

Nesta tese é descrita a aplicação da ferramenta às ligas de Simulação 2D e 3D, bem como à de robôs médios, com exemplos práticos da definição, gestão e execução de jogadas estudadas. Os resultados atingidos mostram a utilidade da ferramenta. Foram realizadas, na liga de Simulação 2D, experiências em condições de competição. Os resultados mostram uma clara melhoria no desempenho, quando comparado com a situação anterior, sem a utilização de jogadas estudadas. Estes resultados motivam a utilização desta ferramenta como a principal técnica de coordenação de qualquer equipa participante numa das ligas de futebol do RoboCup.

Abstract

RoboCup is a worldwide initiative, that aims at fostering robotic research and development based on complex yet well-known problems. There are different, complimentary application domains inside the initiative, with robotic soccer remaining the most challenging and popular one. This domain is organized around several Leagues, with different characteristics, where cooperation between robots is always a primary issue.

Multi-agent coordination and strategic planning are two of the major research topics in the context of RoboCup. However, innovations in these areas are often developed and applied to only one domain and a single RoboCup league, without proper generalization. Also, although the importance of the concept of Setplay, to structure the team's behavior, has been recognized by many researchers, no general framework for the development and execution of generic Setplays has been presented in the context of RoboCup. In the sense employed in this thesis, a Setplay is a freely-definable, flexible and multi-step plan, which allows alternative execution paths, involving a variable number of robots and having optional arguments.

This thesis introduces such a framework for high-level Setplay definition and execution, applicable to any RoboCup cooperative league and similar domains. The framework is based on a standard, league-independent and flexible language that defines Setplays, which may be interpreted and executed at run-time through the use of inter-robot communication. A real-time selection algorithm for Setplays, inspired by Case-based Reasoning (CBR) techniques, was also developed. This tool stores the past application conditions and success of individual Setplays, and uses this knowledge to select among Setplays whenever they are feasible. The Setplay Framework aims at being used as a tool to rapidly prototype multi-agent plans, that are executed at run-time, allowing the swift adaptation to particular opponents, and thus exploiting weaknesses and gaining a competitive edge.

The application of this framework in the 3D and 2D Simulation Leagues, as well as in the Middle-Size League, is also described with concrete examples of Setplay definition, management and execution, which allow the framework to be properly evaluated. The results achieved show the usefulness of this approach. Experiments were made in competitive settings, in the scope of the 2D Simulation league. The results showed a clear improvement in the performance when Setplays are activated. This motivates the usage of the Setplay Framework as a main coordination technique of any team participating in the Simulation, Small-Size, Middle-Size, Humanoid and Standard Platform leagues of RoboCup.

Agradecimentos

Ao Luís Paulo e ao Nuno.
Aos elementos do Júri.
Aos elementos das equipas FCPortugal e CAMBADA.
À Mónica.
À Olga.
À Teresa.

Luís

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

RoboCup is an international initiative to promote AI, robotics, and related fields. It fosters research by providing a standard problem, where a wide range of technologies can be integrated and examined. RoboCup uses the soccer game as a central topic of research, aiming at innovations to be applied for socially significant problems and industries. Research topics include design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion.

The ultimate vision of the RoboCup initiative is that "by mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."[1] This is certainly an ambitious goal, but research in cooperative robotics has been accumulating results that allow the community to continue believing in this challenge. With this goal in mind, some questions have to be asked: what kind of robotic team will play this decisive game when the time comes?

The RoboCup initiative has always believed that, to reach its intended goal, a wide array of scientific challenges will have to be solved. To better deal with this diversity of challenges, RoboCup has been split in different and heterogeneous leagues: Simulation, Mixed-Reality, Small and Middle-Size, Standard Platform and Humanoid, all in the soccer domain, as well as leagues dealing with other domains, such as the Rescue league, both simulated and with real robots, and RoboCup@Home, that researches the use of robots in domestic environments.

This diversity of applications has certainly allowed research teams to deal with a broad set of scientific issues. Nevertheless, it has also brought some drawbacks: the focussing on

[1] URL: `http://www.robocup.org/about-robocup/objective/`

specific leagues and problems, together with the competitive nature of the competitions, has frequently made teams overspecialize in specific issues, and simultaneously neglect the generalization and wider application of results. This has had as consequence that the solutions developed in one league are not easily transferable to other leagues, lacking therefore general purpose testing and applicability. This question has been discussed in more detail in Mota and Reis (2008).

Robot Soccer needs, as the research in the domain develops, coordination at team scope, which involves planning at many levels. This thesis deals with representing and executing high level, flexible plans for robots playing in different RoboCup leagues. A framework for representing, executing and evaluating such plans is presented, relying on inter-robot communication.

Setplays are commonly used in many team sports such as soccer, rugby, handball, basketball and baseball. There are surely several important differences between robot soccer and human sports, but Setplays are nonetheless a useful tool for high-level coordination and cooperation. From a more practical and applied point of view, one can expect these small and flexible plans to be very useful when applied to specific situations, e.g., when an opponent team leaves an empty area when defending a corner kick. In such a situation, a Setplay could be rapidly developed with the goal of exploiting the weakness detected in this situation, and using the empty area as a shooting point. This example highlights one of the most significative usage scenarios for Setplays: serve as a rapid multi-agent plan prototyping tool, which is used to exploit weaknesses in specific situations.

One additional motivation for high level planning has been raised in the Middle-Size League: the Technical Committee has at one point decided that some of the teams would have to join efforts to pairwise create common teams, in order to decrease the total number of participating teams. This idea was not put forward, but anyway the challenge of building multi-partner teams remains unsolved. One possible way to integrate heterogeneous players would be to present them a list of Setplays, understandable to all, and simply tell them when and how each of these Setplays should be executed. A draft scenario of such a mixed team integration has already been presented in Mota et al. (2006).

From this analysis, one can conclude that there is the need for general purpose tools and frameworks, that can apply to several leagues, thus allowing the easy generalization, testing and transfer of obtained results. Furthermore, Setplays can be used as a tool for the rapid prototyping of behaviors that bring a competitive edge.

1.2 Motivating Scenario

To better ground the need for a Setplay Framework in the RoboCup domain, a motivating scenario, from an actual RoboCup team from the University of Porto is presented.

The FCPortugal[2] 2D Simulation team has previously investigated how to organize its collective behavior around so-called *Setplays*. Such *Setplays* were intended for dynamical use in particular situations during the game. An example is used for the execution of goalie catchs: the objective is to get the ball quickly into the opponent's half, with low probability of interception. To attain this goal, four players are involved. The *goalie* is responsible for starting the *Setplay* by moving to the left edge of the penalty area and subsequently passing the ball to the *left defender*, as soon as the latter has moved to a point located near the touch line. Two additional players, the *left midfielder* and *left forward*, will also move to points near the touch line, in front of the *left defender*. The *Setplay* starts when the *goalie* passes the ball to the *left defender*, which tries to pass the ball on to the *left midfielder*, which, upon ball reception, passes the ball on to the *left forward*. This *Setplay* could be fine-tuned and become a very successful collective move, assuring quick moving of the ball onto the opponent's half.

The concept of Setplay, as described, is used to enhance the team's behavior. In FCPortugal's original application scenario, Setplays were not configurable using parameters, and could only make use of a limited set of actions: positioning and passing. Moreover, the number of players was fixed throughout the Setplay.

But what if this kind of collective move could be described and shared in a standard, league-independent and flexible way, which would be interpreted and executed at runtime? The first benefit would be the possibility of writing arbitrary Setplays, which would dynamically be used during the game, opening horizons to new plays which could, for instance, differ from game to game, to better deal with each opponents' characteristics. In that case, the Setplays could also be used in different leagues. Furthermore, since any player could have access to the definition of Setplays and interpret their content, Setplays could also be a means for the creation of mixed teams, where heterogeneous robots would play together: when the Setplays are being executed, players simply have to follow the steps in the Setplay in order to cooperate.

To fulfill these requirements, one needs a standard language, supported by a framework, where Setplays could be defined and interpreted by any player in any league. The basic concepts of soccer (moves, conditions, skills) need a clear and concrete definition.

[2]URL: `www.ieeta.pt/robocup/`

Also, the transitions between intermediary steps have to be expressed, as well as termination conditions. Such a language and framework are the scientific subject tackled in this thesis.

1.3 Objectives

The main objective of this thesis is to develop a general-purpose framework for the application of Setplays, small multi-player plans in the robotic soccer domain, to any league in RoboCup.

Setplays should be high-level, reusable plans, defined in a league-independent language and with a text-based syntax that allows the quick and easy creation or modification of a Setplay.

This framework should be as modular as possible, offering tools that allows its easy application to any league without any change in the framework. Such tools include:

Setplay language: formal definition of a language to clearly define flexible Setplays;

Graphical editor: an editing tool to configure Setplays in a graphical environment, which should allow general users and field specialists to provide Setplay-definitions to the Framework;

Setplay parser: a parser to read the text syntax of Setplays;

Selection of players: algorithms to select the participating players in a Setplay combining Setplay positionings and actual player positions;

Setplay evaluation and selection: algorithms to evaluate and record the performance of individual Setplays, as a basis for real-time selection of Setplays in actual situations;

Setplay execution engine: an easy-to-use mechanism to manage the execution of Setplays;

Communication management: Setplay execution makes use of the communication between players and the framework must provide tools for its management.

To test the application of the framework and prove the validity of the underlying concepts, the framework was applied to several teams: in simulated soccer, and to the Middle-size League.

1.4 Thesis Structure

This thesis was structured with an increasing level of detail from the beginning to the end. Chapter 2 presents a literature review and evaluates the State-of-the-Art. Chapter 3 presents the Setplay Framework's architecture, with focus on the most important options, while also presenting the newly defined language that will allow the expression of Setplays. Next, chapter 4 describes in detail how the Framework was developed into an autonomous library in C++, which can be applied to arbitrary teams. The application of the Framework to three different teams, from the 3D and 2D Simulation and from the Middle-size leagues is presented in chapter 5. Chapter 6 discusses the results achieved by the application of the Framework to these teams. Finally, chapter 7 sums up the thesis, points to future work and draws the relevant conclusions.

Chapter 2

State of the Art

2.1 RoboCup

In the last years, the RoboCup competitions and symposia (Kitano, 1998; Asada and Kitano, 1999; Veloso et al., 2000; Stone et al., 2001; Birk et al., 2002; Kaminka et al., 2003; Polani et al., 2004; Nardi et al., 2005; Bredenfeld et al., 2006; Lakemeyer et al., 2007; Visser et al., 2008; Iocchi et al., 2009; Baltes et al., 2010; del Solar et al., 2011; Röfer et al., 2011b) have increasingly become a test bed for cooperative robotic approaches. Teams from all around the world compete in nine major leagues including six cooperative soccer main leagues: Simulation 2D; Simulation 3D; Small-Size; Middle-Size; Standard Platform and Humanoid.

2.1.1 Simulation League

2.1.1.1 2D Simulation League

The 2D simulation league is based on the publicly available soccer server system (Noda et al., 1998). It simulates the players and the pitch for a 2D soccer match. The server accepts low-level commands from the players, executes them in an imperfect way and sends (imperfect) perception information back to the players. One of the objectives of the simulation league consists in testing high-level multi-agent research issues, while waiting for the hardware to catch up. Advances in the simulation league (Gabel and Riedmiller, 2011) include the use of learning to optimize individual and team skills (Stone, 2000; Riedmiller et al., 2001; Riedmiller and Merke, 2003; Taylor and Stone, 2009), cooperative techniques like formations and roles (Stone and Veloso, 1999; Kyrylov and Hou, 2010), complex team strategies (Kok et al., 2003; Reis and Lau, 2001), new agent architectures

and agents debugging tools (Stone and Veloso, 1999; Reis and Lau, 2001; Lopes et al., 2010).

2.1.1.2 3D Simulation League

The 3D simulation league was introduced as a new competition in 2004. This competition is based on a completely new simulator with a 3D environment (Kögler and Obst, 2004) and established a new set of research problems that had to be tackled. New problems include the physics dynamics model which is much more precise and realistic than the one of the 2D simulator (Smith, 2004), and new agent actions that are closer to the ones of real robots when compared with the 2D simulator (Noda et al., 1998). The 3D simulation league has drawn a lot of interest in the community.

Since RoboCup 2007, the original 3D server, where players were modeled as spheres, was continuously replaced by versions where players were different kinds of humanoids (Boedecker and Asada, 2008). Presently, players are modeled as NAO robots, identical to the ones used in the Standard Platform League, described below. In this phase of development, research has been mainly focused on the development of robot movements, like gait (Lattarulo and van Dijk, 2011; Shafii et al., 2011).

2.1.1.3 Mixed Reality League

Mixed Reality refers to a combination of real and virtual entities. In this case, the techniques used belong to the Augmented Reality research topic, since virtual objects are inserted into a real world, and also from Augmented Virtuality, since real objects interact with virtual objects.

The Mixed Reality RoboCup sub-league (da Silva Guerra et al., 2008) organizes competitions with a standard micro-robot platform inside an Augmented Virtuality/Reality environment. These competitions consist of a soccer tournament where teams play autonomous robotic soccer games. The real robots act with virtual objects on a screen: ball, poles and field. The number of robots per team has grown from two to five, with a long term goal of playing games with eleven players. A simulator (Simões et al., 2011) has been introduced to assist the teams' development.

This league has been active since RoboCup 2007, and FCPortugal has participated on several occasions, with a second place being the best result obtained (Gimenes et al., 2007), in the initial development phase of this thesis.

2.1.2 Small-size League

Small-size teams consist of five robots that play in a 6x4m green-carpeted field. The rules of this league permit the use of a shared global vision (Zickler et al., 2010) and robots' centralized control. During the game, the off-field software controller receives global vision information and sends out commands to the robots using wireless communication. The need for high speed and precise control has given the small-size league the reputation of the 'engineering league'. Developments in subjects like electro-mechanical design, control theory and digital electronics, have been crucial in this league (Stone et al., 2001; Birk et al., 2002). However, teamwork characteristics like explicit passing abilities and Setplays (Browning et al., 2004) have increasingly become more important due to the advances in hardware and the steadiness of the rules.

2.1.3 Middle-Size League

RoboCup Middle-size League poses a unique combination of research challenges, combining most of the challenges encountered in the simulation and small-size leagues. The game is played by teams including up to five robots, in a 18x12m field, with an official FIFA ball. Robots must be totally autonomous and thus carry all sensors and actuators on board, but communication is permitted. In past RoboCup editions, most teams were mainly concerned with hardware design, and complex solutions have been developed: omni-directional cameras (Iocchi and Nardi, 2000; Marques and Lima, 2001), laser range-finders (Weigel et al., 2001) or omni-directional drive designs, self-calibrating vision systems (Lange and Riedmiller, 2005; Heinemann et al., 2007b; Neves et al., 2009), ball handling mechanisms (de Best et al., 2011) and robot self-localization (Lauer et al., 2006; Heinemann et al., 2007a) have concentrated a lot of attention. Only a few teams introduced high-level approaches derived from previous simulation league approaches (Weigel et al., 2001; Hafner and Riedmiller, 2003; Lau et al., 2008, 2010; Conceição et al., 2006). Past directions in this league included the participation of mixed teams (Iocchi et al., 2007), composed by the joint efforts of initially distinct teams, introducing several research challenges, directly concerned with the objectives of this project.

2.1.4 Standard Platform league

The Standard Platform league had its debut on the 2008 edition of RoboCup. It is a replacement for the four-legged league, described in the next paragraph of this section. All teams must use the same robot, NAO[1], a humanoid model which is available since

[1]URL: `http://www.aldebaran-robotics.com/eng/Nao.php`

early 2008. The field is 6x4m and the teams can have a maximum of four robots. B-Human (Röfer et al., 2011a) is one of the best performing teams, having won this league's last three editions of RoboCup.

The four-legged league, which has ceased to be maintained and was last present in RoboCup 2007, was a predecessor to the Standard Platform League, since the robotic platform was common to all team and locomotion was made using legs. Teams consisted of four quadruped AIBO robot platforms from Sony. Objects had different colors, and special markers existed in the field to enable self-localization. Research in this league was mainly focused on vision, self-localization and robot locomotion. However, some teams like the previous champions UNSW and the German Team, have implemented teamwork algorithms like formations and the use of roles. The case of the German Team is particularly interesting because they were voluntarily organized as a mixed team. Several teams competed separately inside Germany but joined efforts when at the World Championship, mixing the software modules through the usage of a common architecture, developed by the four participant universities (Lötzsch et al., 2004). This approach, though successful, requires that all partners commit to a common architecture, leaving much place for innovation on cooperation of teams with heterogeneous architectures.

2.1.5 Humanoid league

The Humanoid League is played by autonomous robots with a structure similar to a human body and human-like senses. Major research issues in this league are stable strategies for walking, running, and kicking the ball, and visual analysis of the environment. The league has three sub-leagues, distinguished by robot sizes: Teen Size, Kid Size and Adult Size. Field size ranges from 6x4m in Kid Size to 9x6m in Teen Size. The number of players per team also depends on the sub-league, and ranges from two to three. Robots are designed and built independently by each team. Team Nimbro (Behnke and Missura, 2011) and Darmstadt Dribblers (Friedmann et al., 2011), both from Germany, have consistently obtained competitive results.

2.2 Agent Architectures in RoboCup

2.2.1 Internal Agent Architecture

Traditionally, architectures employed in RoboCup tend to be hybrid, mixing reactive components with deliberative reasoning. This choice is made because players must be able to

rapidly change their behavior when there is a sudden change in the environment (e.g. ball possession is lost), while there is certainly the need for mid to long-term planning.

Berger et al. (2004, 2006) describe the Double-Pass Architecture, that aims at decoupling the sense-think-act cycle, by using a long-term planner that organizes intentions and an executor that chooses short-term actions to execute. This architecture presents a balance between deliberative long-term planning and reactive response to unexpected events.

An already cited successful example is the German Team (Röfer et al., 2006), which was a cooperative project between several German teams. Each team developed an independent four-legged team, that joined efforts to build up a national team. In order to integrate the best components developed by every team, a general control architecture was defined (Röfer, 2003), where behaviors are defined using an XML-based language (Lötzsch et al., 2004). The system provides development tools like a debugger and a simulator.

Ramos et al. (2007) proposed a middle-ware for soccer robots, with three components: State-of-the-World, decision and low-level skills. The decision component can be set up using either Petri Nets, State Machines or Fuzzy decision. The interface to the middle-ware is not fixed and can therefore change between different applications.

Kleiner and Buchheim (2004) introduce a plug-in based architecture, where each component is developed independently and then connected to the system. This allows a certain degree of hardware independence, since components can be easily replaced. To allow this flexibility, the system's interface is fixed, and all components must comply to it.

Farinelli et al. (2005) present a development and control architecture for robotic platforms. This architecture has a hardware-abstraction layer that allows high-level control components to be applied to different platforms. To help the development, components are built modularly and there are tools for remote inspection. A perceptual model is used for communication between robots.

One less common approach is presented in Ferrein et al. (2006). This framework, based on situation calculus, uses Golog as the implementation language. This way, the framework allows the declarative description of behaviors.

2.2.2 Multi-agent System Architecture

The cooperation between the different players in a team has been the subject of research in many teams, specially in the simulated leagues. This is certainly a broad field of development, and many different approaches have been proposed. This section looks at some relevant examples.

Yokota et al. (1999) present a cooperation mechanism based on communication, using concepts common in the Multi-agent system domain, like communication protocols and speech acts. Shiota et al. (2006), also a Japanese team, similarly aim at using communication, while keeping the bandwidth as low as possible. Agents exchange simple, limited messages with clear semantics.

Lafrenz et al. (2006) use a graphical language to define distributed behaviors. These behaviors are executed by the robots, namely by following synchronization and role-distribution phases. The behavior's progress is decided using State-of-the-World information in each robot, thus needing little communication resources.

In Beaudry et al. (2006), heterogeneous robots are used on the same team, with ball and own positions being communicated to other players. Decision making is done independently by each robot. These decisions are, though, communicated to a supervisor, which has the power to change them in case it considers they are not in the best interest of the whole team.

In Castelpietra et al. (2001), formations are chosen through a voting process. Subsequently, roles in the chosen formation are distributed according to each robot's evaluation of its capacity to fulfill the role. This is possible because all robots are able to play any role, whose characteristics are public, in order to allow adequacy evaluation.

Utz et al. (2005) argue that a team of heterogeneous robots can cooperate simply by the exchange of messages about the State-of-the-World. The communication protocol is based on CORBA. The messages' content is the estimated position and velocity of the player and the observed objects. The communication architecture could also be used to convey information other than the perceived State-of-the-World.

Stulp et al. (2006, 2010) defend the usage of temporal prediction models of teammates in the Middle-size league to help agents coordinating for recovering ball possession. The learning is done off-line through model trees and Artificial Neural Networks. Estimations of the state of teammates and of the utilities of their intentions serve to adapt to their predicted actions.

While planning the present thesis, a proposal was made for an ad-hoc framework to structure the communication and cooperation of heterogeneous agents. This proposal Mota and Reis (2007b) was not further developed, but it will be considered as future work.

2.3 Modeling and Communication Languages

Several authors research on hardware-independent languages to carry high-level information. These languages can have different purposes. Lovell and Estivill-Castro (2005)

define a potentially hardware independent, XML based language to describe physical objects like the ball. Stanton and Williams (2004) go in the same direction, also covering the most basic concepts. One can see these languages as ontologies.

The soccer server (Noda et al., 1998) used in the robotic soccer Simulation competitions also introduces perception and action languages. These languages convey information about the different objects on the field, originally modeled in 2D. The abstraction level is, though, quite low, only allowing primitive actions (dash, turn, kick, etc.) and basic State-of-the-World concepts (distances and heading to objects).

Coach Unilang (Reis and Lau, 2002) is a coaching language that defines concepts essential to coach robot soccer teams, including high-level topics like tactics and formations, and low level concepts like time periods and field positions. There are several options to express the recommended playing options: team mentality, team pressure, field use, formations, etc. Real-time communication of game statistics is also possible. The language allows expressing observed situations like players and ball position, enabling it to be used as a communication language. Based on this language, RoboCup community developed Clang (Chen et al., 2003).

Davin et al. (2005) present a language for players to communicate in the Simulation 2D league. Since there are severe bandwidth restrictions, the main challenge is to produce concise content. It deals mainly with State-of-the-World information. The only exception is a message type that can be used to ask for a pass if the player is in an advantageous position.

Buttinger et al. (2002) introduced a Strategy Formalization Language (SFL), extending CLang by the ability to represent team behavior in a human-readable, easily modifiable format.

2.4 Team level Coordination in RoboCup

Coordination among players is one of the most important research topics in RoboCup, particularly in the Simulation leagues. This subject is certainly very complex, since the environment is not predictable and agents perceive it differently. Also, communication, which could be a tool for synchronization and coordination, is either unreliable or restricted. There have thus been numerous and varied approaches to coordination, which will be looked into in the next sections.

2.4.1 Role allocation

Teams in RoboCup's different leagues do make extensive usage of the concept of role. Roles can have values such as defender, corner kick taker or right forward, and do somehow condition the player's behavior. Since a player might act in different roles, and these can change along the game, there is a need for a strategy to allocate roles.

A general overview of role allocation in RoboCup is presented in Gerkey and Mataric (2004). The authors analyze this question from an Operational Research point of view, and argue that some of the currently employed strategies can be enhanced.

A classical approach for coordination in multi-robot teams ('Alliance') is presented by Parker (1998). This architecture aims at efficient role distribution and relies on communication for coordination. It is not well suited for dependent tasks, since their outcome is not predictable with certainty.

A simple role distribution mechanism is described in Moreira et al. (2006): a central supervisor allocates roles to robots and communicates this to the other players. This kind of choice requires no negotiation, thus being very simple and straightforward.

Another role distribution architecture is presented in Kose et al. (2005), where roles are allocated through auctions among the team members. Costs for fulfilling a role are estimated through reinforcement learning.

Kyrylov and Hou (2007) describe joint defensive positioning, resulting from multi-criteria assignment with constraints on the number of defenders and attackers, and on not allowing a defender to mark more than one attacker. On a later effort, Kyrylov and Hou (2010) elaborate on the Pareto Optimality principle to improve the adequacy of assignments, minimizing the time required to execute an action and considering the risk prevented by marking one attacker. The same kind of principles is applied to offensive situations (Kyrylov and Razykov, 2008), with restrictions on space occupation and availability for pass reception.

TechUnited (Aangenent et al., 2009) team has recently switched from static to dynamic assignment of roles, which is done centrally by a dedicated module, based on the State-of-the-World, namely player and ball positions.

McMillen and Veloso (2006) introduced a strategy for role assignment in the four-legged league. This strategy implies the communication of the currently chosen *Play*, which provides a set of *Roles* to be assigned to all the available players in the team. The strategy assures coordination by the existence of a leader that selects the best momentary *Play* and instructs the other robots on what *Roles* to take. Each *Role* fully determines the player's behavior. The strategy does not, however, define a concept of *Setplay* with intermediary states, and *Plays* do not have *Parameters*. *Plays* are, in this context, a more

limited concept than Setplays, since they merely aim at distributing roles among all the teams players. It is therefore more of a coordination methodology than cooperation with actual plans, as is the case with Setplays.

For role assignment in the middle-size CAMBADA team, a dynamic algorithm that adapts the formation to a possible varying number of active robots is used, which will assign each role/robot to the strategic positionings according to priorities and number of active robots (Lau et al., 2009).

2.4.2 Positional Coordination

A method to achieve coordination based on repulsions and attractions called *Strategic Positioning by Attraction and Repulsion* (SPAR) was introduced by Stone (2000). When an agent is positioning itself using SPAR, the agent maximizes the distance to other players and minimizes the distance to ball and to the goal. This is achieved evaluating several forces: repulsion from opponents and team-mates, attraction to the active team-mate, ball and opponents' goal. It also uses other constraints that have influence in agent's positioning: stay in an area near home position, stay within the field boundaries, avoid being at an offside position and stay in a position where it is possible to receive a pass.

Later, the *Situation Based Strategic Positioning* (SBSP) was introduced in Reis et al. (2001), Lau and Reis (2002) and Lau and Reis (2007). If an agent is not involved, and will not be soon, in an active situation, it will try to occupy its strategic position relative to the actual situation of the game. Through the analysis of the tactic, formation, self positioning in the formation and player type, a player is able to define its base strategic positioning. This position is then adjusted accordingly to ball's pose and game situation (i.e. attack, defense, etc...). The player type defines strategic characteristics like ball attraction, admissible regions in the field, specific positional characteristics for some regions in the field, tendency to stay behind the ball, alignment in the offside line, and attraction by specific points in the field in particular situations. Using a strategic positioning like SBSP, the players will be more well distributed over the field than using an active one like SPAR, which is the reason why several teams adopted SBSP as the standard positioning method.

The CAMBADA coordination layer (Lau et al., 2008) is based on SBSP strategies (see above) used in RoboCup 2D Simulation League, adapted to the MSL specifications.

Dynamic Positioning based on Voronoi Cells (DPVC) (Dashti et al., 2006) positions players along the pitch, based on attraction vectors to reflect players' attraction towards objects, depending on the match's current situation and players roles. Some limitations in SBSP are enhanced, such as the need for home positions and fixed number of players for each role.

Delaunay Triangulation (Akiyama and Noda, 2008) shares principles with SBSP. The soccer pitch is divided into triangles based on training data and a map is built from a focal point (e.g. ball position) to the positioning of players. Constraints are used to solve topological relations between different sets of training data, in order to attain more flexible formations. Though simple, this positioning method manages to obtain reasonable approximation accuracy, and is fast running, adjustable, and scalable.

Work et al. (2009) present a positioning for the four-legged league, based on potential fields, which can be considered in line with the previous work by Stone (2000), described earlier. Besides defining positioning, the algorithm also determines the participants' roles.

Atkinson and Rojas (2008) present a formation choice strategy, in the four-legged league, based on *Game Conditions*. Several aspects of the players action are evaluated, through a trained Neural Network. This evaluation influences the choice of Formations, with experimental results supporting evidence on an improvement of performance.

2.4.3 Setplays

Setplays can be understood, in a broad sense, as multi-agent plans that need the commitment of several players in order to reach a common goal. Setplays are very common in most sports, e.g., soccer, rugby and handball, which can make one believe that such constructs can also play a useful role in robotic soccer.

The concept of *Setplay* is present in a teamwork and communication strategy for the 2D Simulation league, presented in Stone and Veloso (1999). These *Setplays*, however, lack some of the most relevant features now presented in this thesis. Namely, they are meant to be used only in very specific situations, like corner kicks and throw-ins, which are decided by the referee, and are unique for each of these situations. Thus, the question of Setplay activation and choice is not considered. Further, there is no mention to Parameters, though Player Roles are proposed. Most importantly, a *Setplay* is limited to a sequence of Steps, without alternatives, which minimizes the need of choice announcing, and therefore the use of communication with this purpose.

The RFC Stuttgart/CoPS team uses Special Interaction Nets (Zweigle et al., 2006), a simplified version of Interaction Nets adapted to cooperation in multi-agent environments. These diagrams include states representing actions, transitions modeling conditions, eventually global, and sub-nets, with the former components. Certain conditions can model time-dependent issues, and can be used to synchronize multi-agent behavior. Messages can also be used to synchronize multiple networks. The model does not present a standard set of concepts, thus not enforcing generalization, and may lead to the development of very specific cooperative strategies. This team uses dynamic role assignment

(Zweigle et al., 2008), with sub-roles, e.g., role Defender and sub-role Left or Right. The role allocation is done locally by every robot, based on the shared world model, that integrates information from all robots. This being the case, inconsistencies are minimized, since all robots decide based on the same centralized information. Role allocation will be potentially wrong only when the shared world model is inconsistent.

XABSL is a language to describe behaviors for autonomous agents based on hierarchical finite state machines, and has been used by different teams in RoboCup, namely the German Team (Röfer et al., 2006). Recent developments (Risler and von Stryk, 2008) have allowed the use of the language to develop cooperative multi-agent behavior, through synchronization elements, that allow the specification of minimum or maximum number of robots is a given state.

An interesting approach is presented in Castelpietra et al. (2002), where Setplays are represented as transition graphs. These plans, which are formally defined, have a high level of abstraction, and can be applied to different robotic platforms, as it has been the case with Middle-size and four-legged robots. The actual execution of plans and how the robots deal with synchronization issues are unclear topics. In a related research effort (Iocchi et al., 2007), Petri Nets have been used to structure the development of a joint team with robots from two distinct institutions.

A formal proposal for the usage of UML State-charts to specify single and multi-agent behavior was introduced in Murray (2004). This approach was developed and refined in Furbach et al. (2008), but lacks practical implementation and is not sufficiently clear about synchronization mechanisms. Further, the underlying state charts are considerably dense, and therefore difficult to read. This approach could not easily be used by non-IT specialists.

Rad et al. (2004) use a tree of plan sequences to choose the best suited plan in each situation. This tree must permanently be re-evaluated. The mechanism for synchronization, a vital issue, is not clearly described.

Kok et al. (2003, 2005) and Kok and Vlassis (2006) introduce Coordination Graphs (CG), exploiting the dependencies between agents and decomposing a global payoff function into a sum of local terms that are communicated among agents. Nodes in the CG represent agents and its edges define dependencies between them, which have to be coordinated. The continuous aspect of state spaces in robotic soccer discourages the direct application of CGs. To solve this question, roles are allocated to agents to discretize this space and then the CG methods are applied to the derived roles. To simplify the algorithm, it is assumed that only a limited number of near players need to coordinate their actions.

The Brainstormers Tribots (Lange et al., 2008) have their roles, and the team formation, decided centrally by a dedicated, high-level module. The closest robot to the ball

acts as a master and decides which play to use.

Team Agent Behavior Architecture (TABA) (Ruiz and Uresti, 2008) uses hierarchical task decompositions to coordinate the behavior of players in the old four-legged league. Collaboration between players is managed through formations including roles, which describe players' positions. Formations and role choices depend on the team attitude and game state. A CBR system stores a strategy base. A strategy case is a plan designed to achieve a particular goal and includes applicability and termination conditions and a list of formations with roles. Further work by Vega et al. (2006) includes a Soccer Strategy Description Symbols (SSDS) graphical notation, an eXtensible Markup Language (XML) behavior language and a control simulator based on Finite State Machines.

The Carpe Noctem team (Skubch et al., 2011) has introduced a XML and XMI-based language to help structure responsive teamwork. The underlying model is similar to Petri-Nets. The resulting plans can be dynamically instantiated, but they primarily serve the purpose of distributing roles and behaviors among the team members. There is an associated plan editor, which is more appropriate for use by robotic soccer experts, and unsuitable for users only acquainted with regular soccer.

Cooperative behaviors through Petri Net Plans, based on the Joint Intentions theory (Cohen and Levesque, 1991), were developed by Palamara et al. (2009) for the management of passes in the four-legged-league. Ziparo et al. (2008) introduced a framework based on Petri-Nets for representing multi-agent plans. This framework allows the description of multi-agent interactions, with primitives for synchronization between players.

Stoye and Elfers (2007) present a framework introducing plans in the Small-size league, including a graphical editor where positions depend on a discrete partitioning of the field. Plans are evaluated through several dedicated *experts* which consider qualitatively the success rates of dribbles, passes and shots. Plans typically lead to a shot at goal, have usually only two or three steps, and can be interrupted if a good shooting opportunity occurs. Steps are considered to be defined in a fixed order, and there is no alternative to transition from a step to another. No experimental results are presented, and player coordination is not considered, since this is not an issue on this league, as all agents are controlled centrally.

2.4.3.1 Setplay selection and activation

One relevant task for the successful application of Setplays is their selection during the game, and therefore possible tools with this purpose must be considered.

Case-based reasoning (CBR) is a technique to solve new problems based on the solution to known, similar situations in the past (Aamodt and Plaza, 1994). Typically, a CBR

system has a large case base, with knowledge about past situations. When a new problem is presented, the case base is searched for a similar case (case retrieval). The solution to the retrieved case is then adapted to the new problem at hand, and applied. This application can at this point be evaluated and also inserted in the case base, for later usage, generating new cases for future selection and adaptation. Case-based Reasoning has had several applications in RoboCup, but is not yet a very popular research topic. The most significant results will be looked at in the next paragraphs.

Karol et al. (2004) present a formal approach to classification and evaluation of plans. Unfortunately, this approach remains purely theoretical, since it has not been implemented in a real system.

An application of Case-based Reasoning techniques to the four-legged league has been presented in Ros et al. (2006, 2007) and Ros and Veloso (2007). A single agent is responsible for retrieving the most suited case and advertising its choice to the teammates, who will perform the sequence of actions in the plan. This approach lacks testing in real game situations.

A different usage of CBR is presented in Wendler and Bach (2004), where behaviors when manipulating the ball are analyzed and predicted. In this case, the results can be used to better predict the opponent's behavior. Lattner et al. (2006) aim at extracting small interaction between players from game logs. The extracted interactions are, though, not evaluated nor used in real game situations.

Berger and Lämmel (2007) try to use CBR to activate cooperative behaviors, in this case involving two players. The experimental results showed that this application was not clearly successful.

In the approach presented in Bowling et al. (2004), the success of *Plays* in the scope of the Small-size league is recorded, in order to help the choice of these *Plays* in future execution opportunities. This evaluation can rapidly change, even during one game, to cope with *Plays* that ceased to be effective due to practical reasons like not being adapted to a specific team, or opponent adaptation to the *Play*.

Ma and Cameron (2009) aim at applying Reinforcement Learning techniques to multi-agent planning. The presented results, of experiences in the 2D Simulation league, show some improvement only after several hours of learning (up to 20), which means that this technique should be used only for off-line learning. Moreover, experiences were done in a laboratory environment, only with teams with very limited performance. Hence, the results of the application of this approach to real game situations are unknown.

2.5 Summary

Although many approaches to cooperative robotics exist in the context of RoboCup, only a few teams include high-level concepts general enough to be useful for more than one league.

There has been no general, abstract proposal for Setplay-based cooperation that would be applied to the domain as a whole, which is the main scientific contribution of the present work. This thesis' project approach differs from all the known approaches, since it has the long term intention of being used in all RoboCup leagues with the same agent-based software. Furthermore, it also aims at integrating robots with different origins in its teams.

Chapter 3

Setplay Framework

This chapter makes a thorough description of the core components of the thesis. Section 3.1 describes the information architecture of the framework, while the communication protocol is described in section 3.2.

This Setplay Framework was designed with the aim of being flexible and applicable to different teams in different leagues. To make the integration of the Framework in already existing teams easier, several alternative usage profiles were envisaged, as described in section 3.3. This kind of integration will demand the application of some abstract concepts in the Setplay Framework to actual concepts or skills in the underlying team code. To support this task, some auxiliary classes were defined, as described in section 3.4. Additionally, some operations created to manage Setplays are described in section 3.5. The language underlying the framework is presented in section 3.6.

To wrap up this chapter, section 3.7 presents a complete, practical example of the usage of the Setplay Framework.

3.1 Framework Description

The Setplay Framework (Mota and Reis, 2007a), inspired by the motivating scenario and the challenges it creates, was designed with the goal of being general, flexible, configurable and applicable to any robotic soccer league. Its intended integration model with an arbitrary soccer player can be seen in Fig. 3.1. The picture, an UML component diagram, shows how the player's components interact with the Setplay-specific tools and components. Basically, the player's top-level decision maker will use the Framework's *Evaluation and Selection* tool to evaluate, at all appropriate moments in the game, if it should run one of the available Setplays. If it chooses to do so, then it will interact with

the *Execution Engine* component, which will guide the execution of the Setplay, depending on the State-of-the-World and on some choices done by the player, as seen further on section 3.3.2. During this execution, the Framework will need both to check if some conditions are satisfied, and to order the execution of some actions. To do this, it will use the native capabilities of the player, which will be available through the implementation of abstract Actions and Conditions, as described in section 3.4.

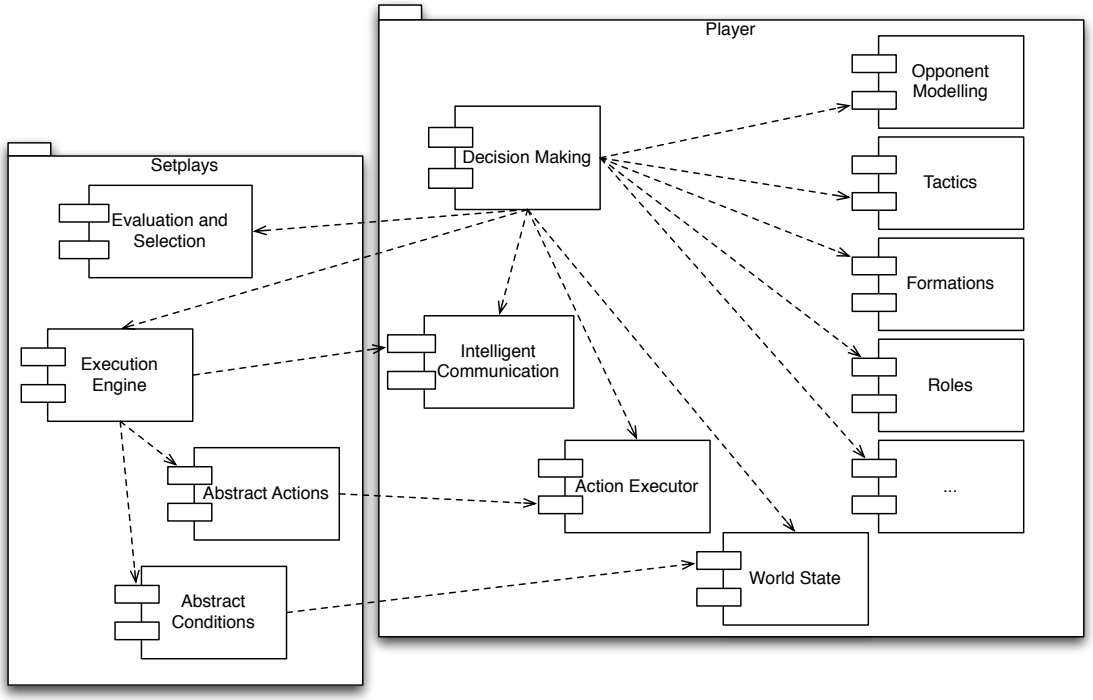


Figure 3.1: Setplay integration with an abstract soccer player

The general structure of the Framework is shown schematically in Fig. 3.2. At the top level, a *Setplay* is identified by a name and a numeric identification (*id*), and has *parameters*, which can be of simple data types like integers and decimals, or more sophisticated concepts as *points* and *regions*. An additional argument allows the *Setplay* to be labeled as *invertible*, which means that this *Setplay* can be inverted, and will be valid on the opposite longitudinal side of the pitch. *Setplays* also have *Player References*, which identify players taking part in the Setplay. The *Player References* can point to specific players, or be *Player Roles*, i.e., abstract representations of a particular role in the *Setplay*, identified by a name (e.g., attacker, supporter). *Parameters* and *Player Roles* will be instantiated at run-time, allowing a flexible use of the *Setplay*. There is also an *Abort Condition* which, when satisfied at any moment during execution, implies the Setplay should be aborted. *Conditions* will be described in further detail later in this section.

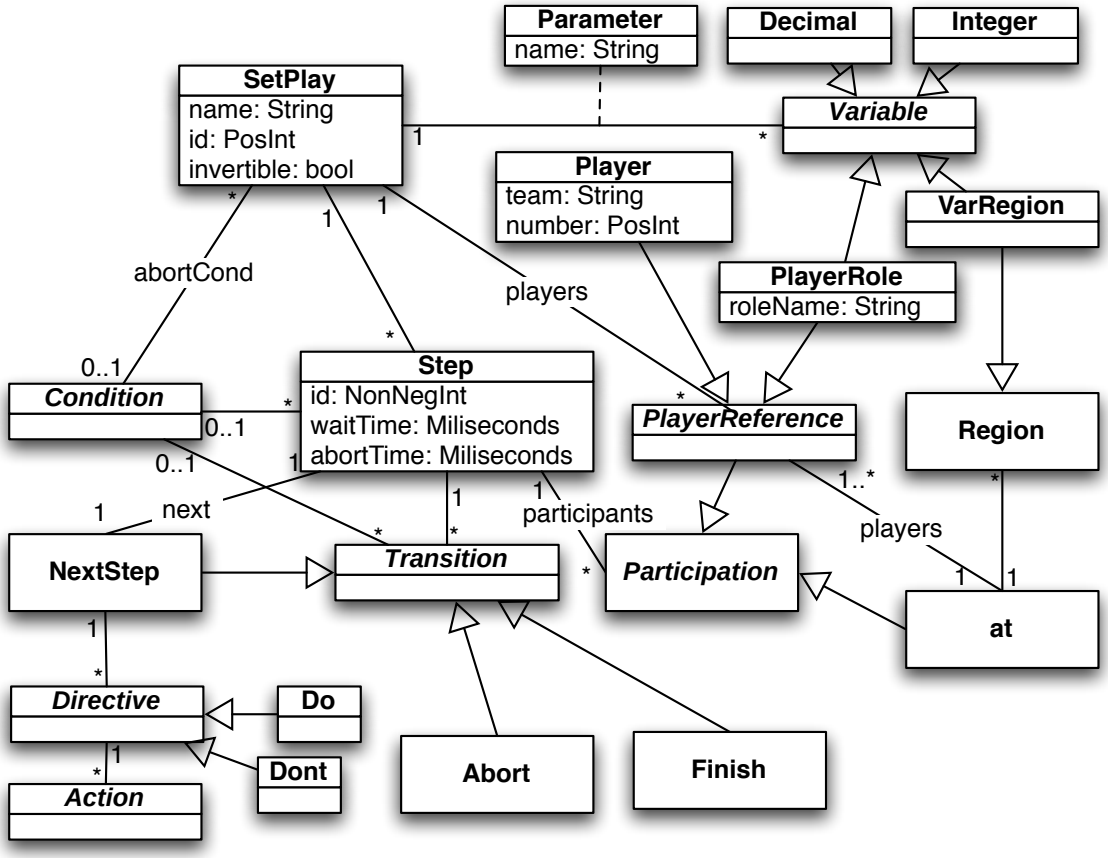


Figure 3.2: Setplay definition

Steps are the main building blocks of a *Setplay*, which will contain an arbitrary number of these, gathered in a list. A *Step* can be seen as a state in the execution of a *Setplay*. By convention, the first *Step* in a *Setplay* is always labelled with 0 as its *id*. Some players participating in a *Setplay* will follow all of these *Steps* in order to accomplish the successful execution of the *Setplay*. Other players might only participate in some of the *Steps* and then abandon it in subsequent *Steps*.

A *Step* has an *id*, which is a non-negative integer. In order to control the *Step*'s execution, the concepts of *'waitTime'* and *'abortTime'* are introduced. Wait time is the amount of time the player should wait, after entering the *Step*, before starting the transition to another *Step*, or simply finishing the *Setplay*. The abort time is the threshold after which the players will abandon the *Setplay*, if it was not possible to progress from this *Step* to another one, or finishing the Setplay. A *Step* can also have a *Condition*, which should be satisfied before entering the *Step*. A list of *Participations* identifies the players taking part in this *Step* in particular, with an optional location (*at*), indicating the desired positioning of the player in this *Step*.

There are several possible ways out of a *Step*, which are defined as *Transitions*. All *Transitions* can have a *Condition*, which must be satisfied for the *Transition* to be followed. An *Abort Transition* represents a situation where the *Setplay* must be abandoned, either because it is no longer judged useful, it is thought not to reach its goal, or for any other reason. The *Finish Transition* represents that the *Setplay* has reached its intended goal and should end at that point. The most common *Transition*, that is used to link between the different *Steps* is defined as *NextStep*. It includes the *id* of the next *Step* to be reached, and contains a list of *Directives* that will be applied in order to accomplish the *Transition*.

3.1.1 Directives and Actions

Directives include *Actions* and can be of two kinds: *'Do'* and *'Dont'*, meaning, respectively, that the contained *Actions* should, or should not, be executed. In this context, *Actions*, depicted in Fig. 3.3, are high-level concepts that represent skills and moves, both simple and complex, that can be executed by a player. Examples of such *Actions* are passing the ball to a player or region, shooting at goal, intercepting the ball, or dribbling. In the *Setplay* framework, the *Action* concepts were initially inspired by the ones defined by Clang (Chen et al., 2003), the coaching language used in the Simulation league. The original action names were kept, but the reference to players (and player sets) in arguments was changed: the original definition allowed only integers (representing player numbers) as identification. Since Setplays are commonly defined using player roles, the arguments

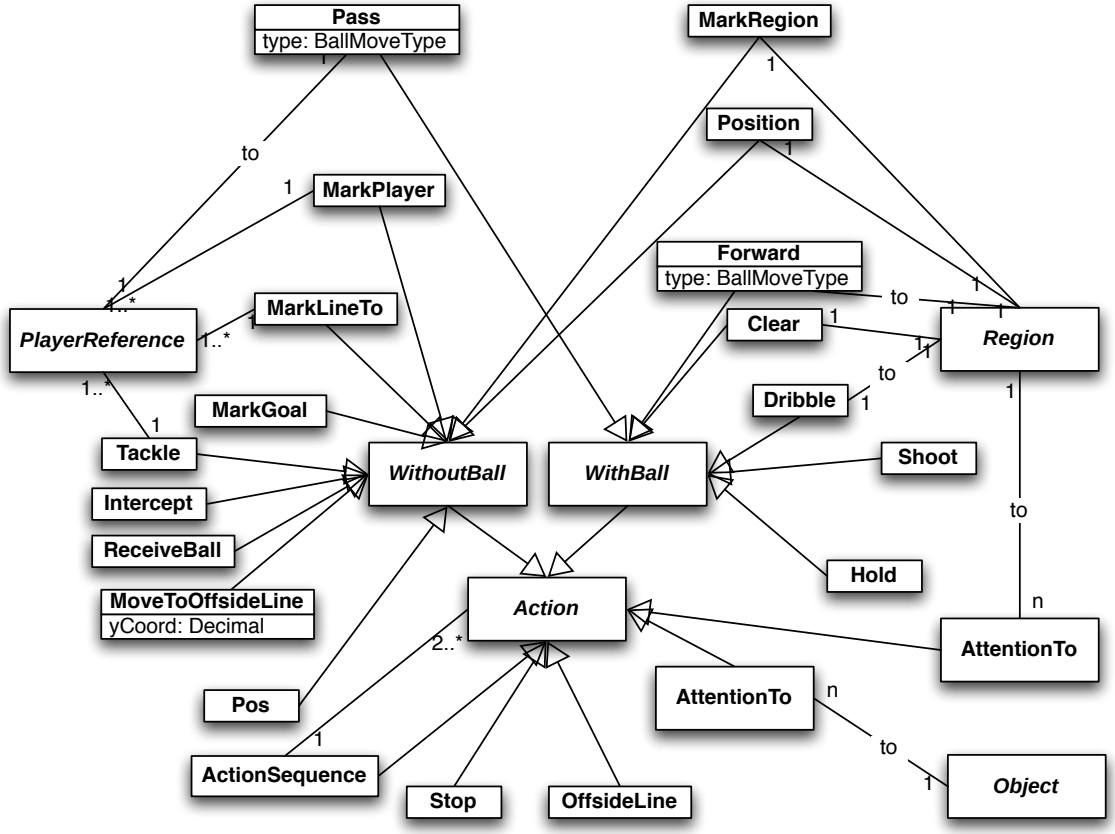


Figure 3.3: Action definition

of actions involving players were defined as *PlayerReferences*, which can be, as mentioned before, both *PlayerIDs* and *PlayerRoles*. These actions are the most common and fundamental to play soccer, and will be described here:

Position: takes a region as argument, the executor should position itself in that region (short name: *pos*).

Forward to region: takes a region as argument, the ball should be forwarded to that region (short name: *bto*). There is an optional argument, *type*, with *normal*, *fast* and *slow* as possible values, describing the type of kick that should be employed. In future applications of the Framework to other leagues and teams, the possible types might be expanded to include concepts like lob-kick, or even heel kick.

Pass to player: takes a player reference as argument, instructs the executor to pass the ball to that player (short name: *bto*). Similarly to the forward action, there is an optional argument, *type*, with values similar to the ones described in the previous action.

Mark player: takes a player reference as argument, which should be marked by the executor (short name: *mark*).

Mark pass line to player: takes a player reference as argument, instructs the executor to mark the pass line to that player (short name: *markl*).

Mark pass line to region: takes a region as argument, the executor should mark the pass line to that region (short name: *markl*).

Set Offside line: takes a region as argument, instructs the executor to set the offside line trap at that region (short name: *oline*).

Dribble: takes a region as argument, instructs the executor to dribble the ball to that region (short name: *dribble*).

Clear: takes an optional region as argument, instructs the executor to clear the ball from that region (short name: *clear*).

Shoot: without any argument, instructs the executor to shoot at goal (short name: *shoot*).

Hold: without any argument, instructs the executor to hold the ball at the present location (short name: *hold*).

Intercept: without any argument, instructs the executor to actively intercept the ball (short name: *intercept*).

Tackle: takes an (opponent) player reference as argument, which presumably holds the ball and should be tackled by the executor (short name: *tackle*).

There was, however, the need to introduce several new actions, to cope with needs in the different leagues to which the Framework was applied:

Action Sequence: composite action, with unspecified number of arguments, where several actions are to be executed following a particular order.

Stop: action with no arguments, to force a robot to stop, especially important for robots moving fast and/or with high inertia (short name: *stop*);

Attention to Region or Object: two different actions, with either a *Region* or *Object* as argument, to instruct a robot to keep attention on the argument (short names: *attentionToReg* and *attentionToObj*). In the Framework, an *Object* represents any physical object present on the field, such as the goals, the ball or players, as pictured in Fig. 3.4 .

MarkGoal: action with no arguments, orders the robot to position itself close to the own goal, in order to avoid opponent goals, may be especially important when Goalie is not present or is malfunctioning (short name: *markGoal*).

ReceiveBall: action with no arguments, informs the robot that the ball will be passed to it, in order for it to be able to position itself properly, and possibly perform an active reception (short name: *receiveBall*).

Move To Offside Line: action with an optional decimal as argument, instructs the robot to move rapidly towards the opponent offside line to the y-coordinate matching the argument, to prepare the interception of the ball when it is passed to the back of the opponent defense. This action is important to allow the player performing this action to find itself isolated behind the opponent defense (short name: *moveToOffSideLine*).

Both Action Sequence and the actions to focus attention on an object or region and to mark the goal were introduced with no specific league in mind: such abstract actions might be necessary in particular Setplays, e.g. when a robot does not have omnidirectional vision available and needs to know the positioning of an object or the vacancy of an area. Actions to receive the ball and to stop were deemed necessary when applying the Framework to the middle-size league team CAMBADA: on the one hand, the reception of passes is a considerably precision-demanding task, that needs active positioning and controlling of the ball handler, while, on the other hand, when a robot is actively moving,

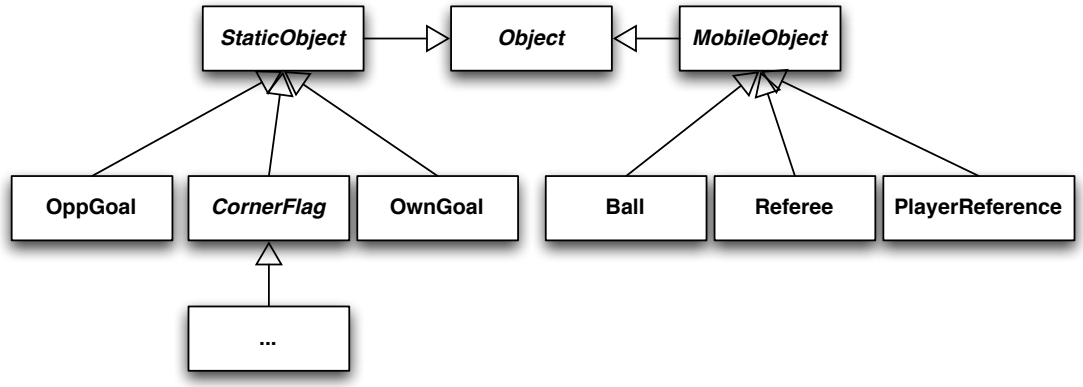


Figure 3.4: Object definition

it needs to be instructed to stop, or else the robot will continue moving due to inertia. The action to move to the offside line only makes sense in leagues where there is a Offside rule. This is the case in the 2D Simulation league, and such an action might be profitably used in Setplays.

As argued, some actions are relevant only to particular leagues, and might be ignored in leagues where they do not apply or are not meaningful. Since the Framework was designed to apply to different leagues, this design option was taken, as it is a way to allow the Framework's application to all leagues without any change. In cases when an action should not be used, an option that should be taken by the developer in charge of the application, the action can simply be ignored and not implemented, as long as it is checked that the action is not used in actually executed Setplays.

3.1.2 Conditions

The concept of *Condition*, already mentioned before, also plays a central role, and is depicted as an UML class diagram in Fig. 3.5. Such *Conditions* have a wide field of application, and deal with the whole domain of robotic soccer. Similarly to the *Actions*, the basic *Conditions* in this framework were inspired in Clang. Examples of such *Conditions* are players and ball being in particular positions or regions, ball ownership and play mode. These elementary conditions, and their arguments, can be described as follows:

Player Position: takes a list of player references, a region and a *minimum* and *maximum* as arguments, and succeeds if, from the player list, there are at least *minimum* and at most *maximum* players located in that region (short name *ppos*).

Ball Position: takes a region as argument, and succeeds if the ball is located in that region (short name *bpos*).

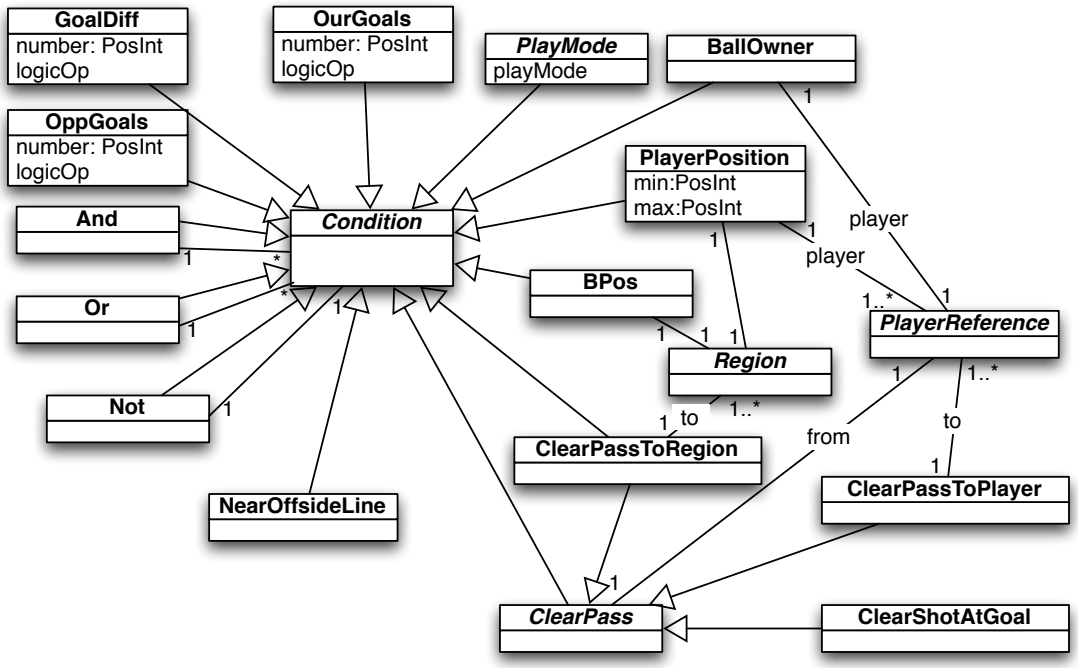


Figure 3.5: Condition definition

Ball Owner: takes a list of player references as argument, and succeeds if one of these players has ball possession (short name *bowner*).

Play Mode: takes a token representing a play mode as argument, and succeeds if that is the current play mode (short name *playm*).

Time: takes a comparison operator and a number as arguments, and succeeds if the comparison of current time with the numeric argument complies to that logical operator (short name *time*).

Opponent Goals: takes a comparison operator and a number as arguments, and succeeds if the comparison of the opponent's score with the numeric argument complies to that logical operator (short name *opp_goals*).

Our Goals: takes a comparison operator and a number as arguments, and succeeds if the comparison of the own score with the numeric argument complies to that logical operator (short name *our_goals*).

Goal Difference: takes a comparison operator and a number as arguments, and succeeds if the comparison of the score difference with the numeric argument complies to that logical operator (short name *goal_diff*).

And, Or and Not: take other conditions as arguments (singleton condition in the case of *not*) and have the usual meaning for these logical operators (short names *and*, *or* and *not*).

In the Setplay application domain, however, several new *Conditions* had to be introduced, in order to model complementary situations. Particularly, some *Conditions* refer to the possibility of accomplishing passes and shots, i.e., modeling the success of passes to players and regions, and shots at goal. One should pay special attention to this kind of *Conditions*: they are not based on a verifiable State-of-the World, but instead are an estimation of a success probability. This could be considered as intrinsically different from *Conditions* like player position, which are tangible and verifiable. Even these *Conditions* are, though, in the scope of robotic soccer, also somehow an estimation: the players do not know the exact State-of-the-World, they simply have their own estimation, built from own observation and, possibly, from information shared by other team-mates. Therefore, for the sake of simplicity and expressiveness, all these concepts are indistinguishably considered *Conditions*. The newly introduced *Conditions*, also visible in Fig. 3.5, are as follows:

Clear Shot at Goal: takes a list of player references as argument, and succeeds if one of these players is in a good position to shoot at goal with a high success probability (short name *canShoot*).

Clear Pass to Player: takes two lists of player references as argument (*from* and *to*), and succeeds if one of the players in *from* is in a good position to pass the ball to one, or more, of the players in *to* (short name *canPassPl*).

Clear Pass to Region: takes a list of player references and a region as arguments, and succeeds if one of these players is in a good position to forward the ball to that region (short name *canPassReg*).

Near Offside Line: takes a list of player references as argument, and succeeds if one, or more, of these players is near the opponent's offside line (short name *nearOffsideLine*).

Some of the defined *Conditions* are very straightforward to check, e.g. those regarding the score and time, while others are league-specific and somehow more fuzzy, like ball possession and pass possibility. As such, some of these *Conditions* will have to be implemented specifically for each team and league, while others can be checked by simply evaluating basic information in the State-of-the-World. The *Conditions* that actually need to be implemented when applying the Framework to a team are cited in section 3.4.

3.1.3 Regions

Regions are another concept in the core of the definition of *Setplay*, and are depicted in the diagram in Fig. 3.6. The definition of these concepts originates from Clang, including spatial entities like *Points*, *Triangles*, *Arcs* and *Rectangles*. Similarly, the concept of *Dynamic Point*, referring to the location of a player or of the ball, is also introduced. Named regions were added to the Framework, in order to model intuitive locations like 'our mid-field' or 'their penalty box', with names inspired by the ones defined in Reis and Lau (2002).

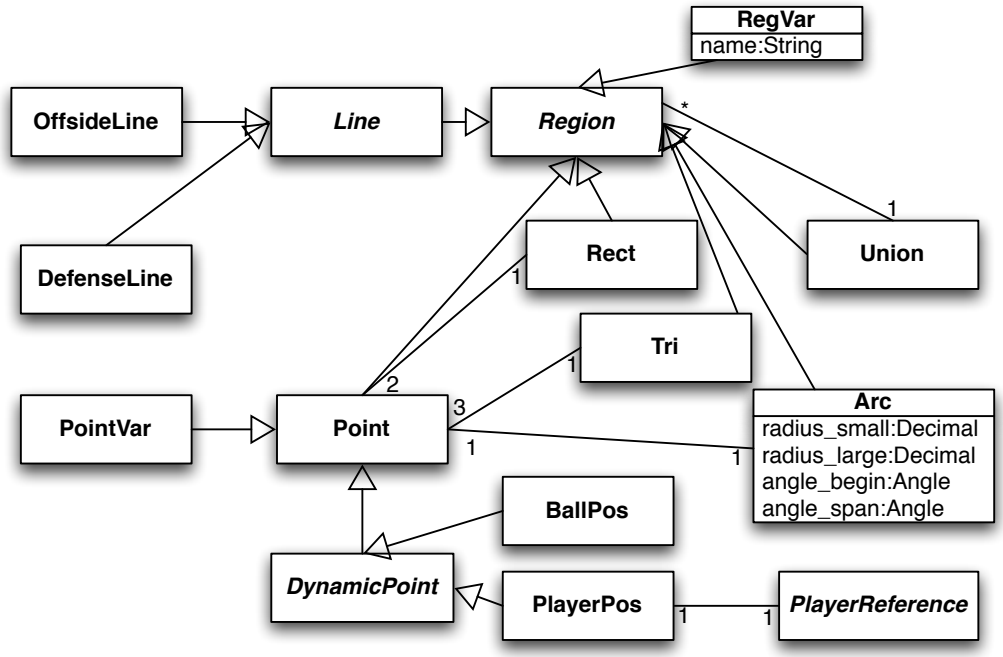


Figure 3.6: Region definition

3.2 Inter-robot Communication

The major issue in this framework was how to achieve coordination between the robots when executing a *Setplay*. The Setplay Framework application scenario considers that all players in the team know the available Setplays. In practice, though, a complex *Setplay* must follow several steps, and all participating players must be minimally synchronized in order to achieve fruitful cooperation. The first step towards this objective was to define a communication and synchronization policy, which should be as straightforward as possible, and can be seen in Fig. 3.7. This figure depicts the protocol that guides the interaction between the participants in a Setplay.

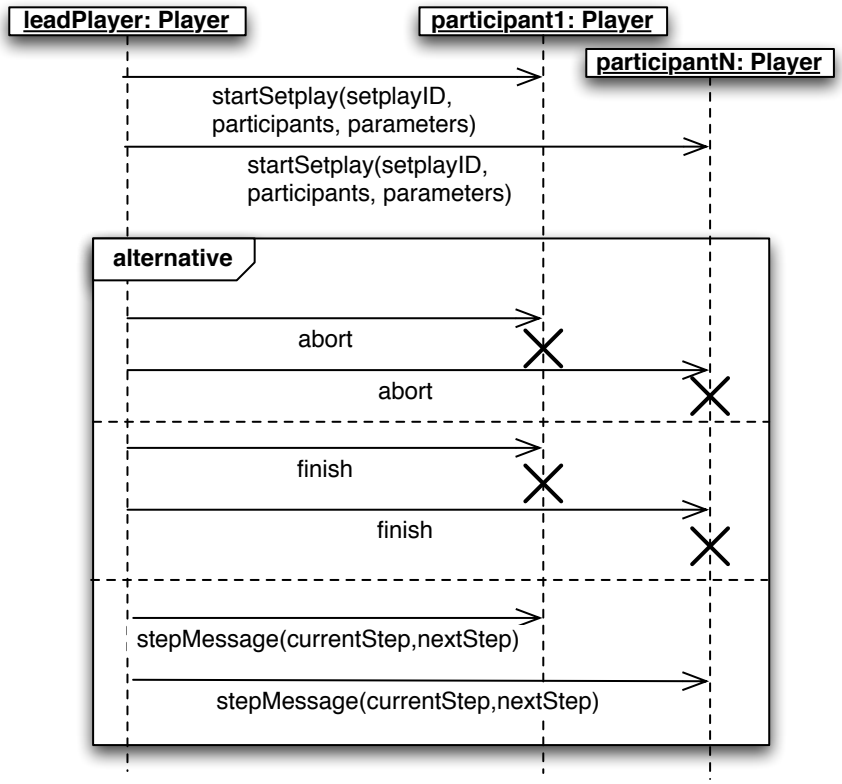


Figure 3.7: Setplay interaction scheme

Setplay start is decided by one of the participating agents (a player, or, e.g. the coach, depending from the team's application policy), and communicated to the other participants through an *InitMessage*, which has three different parts:

1. the identification of the Setplay being started, through its numeric *id*;
2. the specification of the participating players, which will instantiate the Setplay's roles, through their player numbers;
3. the value of all the other parameters present in the Setplay's definition.

To illustrate this kind of message, one can look at an example of a Setplay, whose identification number was 4, in which three players participated and which had an integer as sole argument. In order to instantiate such a Setplay, one possible *InitMessage* would be:

```
(startSetplay :setplayID 4 :participants 2 3 4 :parameters 8)
```

This message indicates players 2, 3 and 4 as the participants, and 8 the value of the single parameter. In settings where the bandwidth is limited, it is possible to write this kind of message in a more concise format, as follows: `S4 2 3 4 8`.

After Setplay start, each step will be managed by the so-called *lead player*, which takes the most important decisions. This player, which is possibly not fixed throughout the *Setplay*, and can thus change from step to step, will monitor the execution of the *Setplay*, instructing the other players on step entry and transition choice, through *StepMessages*, which must contain two integers. The entry into a new step, which is decided by the lead player in charge of the previous step, possibly implies the change of the lead player, depending on Setplay definition. Using some more examples, a message to indicate that Setplay execution's *current Step* is identified by 3, and that the *nextStep* has already been chosen, and is number 4, would be as follows:

```
(stepMessage :currentStep 3 :nextStep 4)
```

In a shorter format, the message could also be expressed as simply as "`3 4`". If the *nextStep* was not chosen yet, the second element of the message would be -1. When the *Setplay* reaches a successful finish, or is aborted, the *current Step* in the message will be number -1, while the *nextStep* will be 1 if the *Setplay* was successful, and -1 if it was aborted. These messages allow all players to be informed of the Setplay's outcome, which is important information for *Setplay* evaluation.

Details on the application of this communication protocol can be seen in the example described in section 3.7. The format and encoding of the messages is not dictated

by the Framework, as different application settings and the teams' options may influence the actual communication procedure. The Framework does, though, supply a class that models the different messages, and provides methods both to access the messages' elements and to output it as text. It is up to the user/developer in charge of the Framework's application to choose the best way of transmitting the messages. The applications of the Framework described in chapter 5 show that the messages' transmission can be done in different ways: as server-managed communication in the 2D Simulation league, and using the team's black-board in the Middle-size league, in the case of the CAMBADA team.

3.3 Practical Application and Usage profiles

When applying the Framework to a team, there are three different tasks that must be executed to accomplish Setplay execution: Setplay selection, Action selection and execution, and Communication management.

As this Framework was designed to be used in different environments, teams and leagues, it may happen that the integration in the existing code is done in different ways, to ease adaptation and avoid any type of redesign of the existent code. Thus, for two of the previously mentioned tasks, there will be alternate usage profiles. The ultimate goal was to keep the Setplay Framework as modular and encapsulated as possible, which would bring two major advantages to the person doing the code integration, which will from now on be called user/developer: abstraction from the Framework's implementation details and flexibility while integrating the Framework in any previously existent code.

Two particular aspects of the code integration are subject to flexible usage: Setplay selection and activation on one side, action selection and execution on the other. These, as well as Communication management, will be the subject of the next sections.

3.3.1 Setplay selection

The way Setplays are chosen depends heavily on the user/developer's options: the Framework supplies a tool to help with this task, but it always remains the possibility that the user/developer prefers to make his own choice. This CBR-based tool, described fully in section 4.2, is available as a method of the Framework's class '*SetplayManagerWithCBR*, and can be seen pictured in Fig. 3.8.

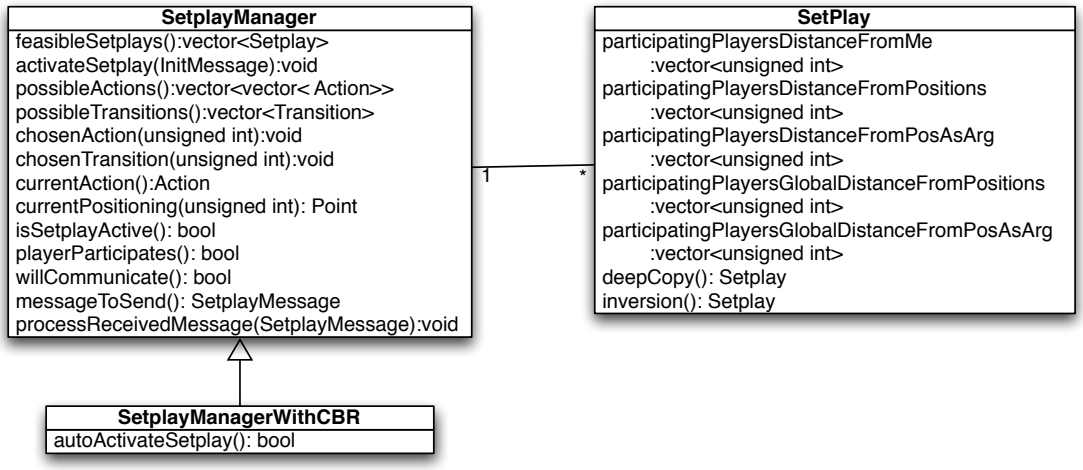


Figure 3.8: Setplay Manager

3.3.1.1 External selection

If the user/developer wishes to perform this choice autonomously, outside the scope of the Framework, he may call the method to check which Setplays are feasible (*feasibleSetplays*), and subsequently instantiate one of these through the corresponding method (*activateSetplay*), which accepts an *InitMessage* (see section 3.2) as the sole argument.

Among these arguments there will certainly be the Setplay's participants. These have to be chosen, by the user/developer, in order to instantiate the Setplay. To assist this choice, the Framework supplies several tools, which are described in detail in section 4.1.2.

Besides the selection of players, any other parameter in the Setplay must be externally selected, and also sent as a argument. The Framework does not provide any tools for aiding the choice of these parameters.

3.3.1.2 Automatic selection

The user/developer may also choose to let the Framework take care of all the tasks necessary for Setplay selection, instantiation and start. With these functionalities in mind, a system inspired by Case-based Reasoning (CBR) was developed, which stores information about Setplay execution and the corresponding success or failure. This system is described in detail in section 4.2. If at a given moment in the game one or more Setplays are feasible, this system will select the Setplay with the best evaluation. The evaluation depends on prior success of the Setplay, both in this game and previous ones. This can be done through the invocation of method *autoActivateSetplay*.

In this case, participating players are automatically chosen according to the distance from their positions to the Setplay's initial positions. Since there is no way of selecting other Setplay parameters, these are not permitted to exist in this scope. Thus, for automatic selection of Setplays, one can only consider Setplays with no other parameters besides participants.

3.3.2 Action selection and execution

When a Setplay is running, a player executing it will want to have the necessary actions executed. Depending on the user/developer's options, the player might also want to select what action to perform, when there is more than one available. Is this scope there are two different usage profiles: one is automatic, the other relies on external selection of alternatives. This option is fully independent from the one taken for Setplay selection and activation.

3.3.2.1 Automatic action selection and execution

From the point of view of the user/developer, this option is like a black box: execution of actions, any inherent choices and Setplay management are done automatically, without any external intervention. In this profile, the Setplay Framework does all the choices and executes the actions without any intervention from the user/developer. This profile is thus probably the easiest to adapt to a new team, since the user/developer only needs to make sure that, at some point where an action should be executed, he calls a general method (*executeSetplay*). This method invocation instructs the Setplay to update it's internal state and execute any necessary action.

3.3.2.2 External selection and execution of actions

In this profile, one can keep more control over the execution of Setplays, since the user/developer has to choose among alternative actions or transitions; he must also explicitly order the execution of any necessary action; and, finally, he can execute complementary tasks, such as positioning itself, when there are no explicit actions to execute.

The price of this rise in control level is that the management of internal state update, alternative choice and action execution have to be explicitly invoked. Specifically, some tasks must be taken care of, to ensure proper Setplay execution:

1. **Update internal state:** each time an action is to be executed, the Setplay's internal state must be updated, because changes in the State-of-the-World might imply changes in the Setplay, like *Step* changes or Setplay ending. Therefore, before the

execution of each *Step*, the corresponding method (*updateInternalState*) should be invoked.

2. **Choice among alternatives:** the Framework allows a *Step* to have alternative transitions, either to other *Steps* or to Setplay ending (*abort* or *finish*). When more than one *Transition* is available, a choice between the existing alternatives must be made. The possible actions or transitions can be accessed through dedicated, alternate methods (*possibleActions*, which returns all possible *Actions* in a vector, and *possibleTransitions*, whose value is a vector of *Transitions*). These alternative methods are complementary, and can be used freely according to the user/developer's choice: the vectors returned will always have the same size, and they only differ in the content of the vector, as the first option only contains *Actions*, while the second contains *Transitions*. This option is offered to the user/developer because in some cases (as was the application of the Framework to the middle-size team CAMBADA) it is a special agent (the team's coach) that makes this kind of decisions, and it never executes any actions, since it is an intangible agent with no physical existence. After having evaluated these alternatives, and consequently having chosen one of them, a method must be invoked to inform the Framework of this choice (*chosenAction* or *chosenTransition*, depending on which of the previous methods was used).

3. **Execute the necessary action:** when an *Action* is to be executed in order for the Setplay to progress (which can be checked by invoking the method *currentAction*), then the user/developer must make sure that it is effectively executed.

4. **Optional tasks:** besides the execution of actions, the user/developer he may choose to perform other complementary tasks to enhance the execution of Setplays. He may, for instance, ensure that the player, when possible, moves to the current Setplay position (accessible through method *currentPositioning*), or control the looking direction, pointing it e.g. to a relevant player (which is tentatively determined by the method *relevantPlayers* of class *Action*).

3.3.3 Communication management

The joint execution of Setplays by a set of agents demands that the participants are minimally synchronized, which is a complex task in an open environment such as robotic soccer. Even though all players know all Setplays definitions, it has somehow to be assured that agents know at which *Step* the Setplay is at any given moment, as this will dictate which actions must be executed, as well as the players positions.

Since perception is very unreliable in such a dynamic world, communication must be used to ensure the necessary amount of shared knowledge and synchronization, as described in section 3.2. The agents must, as seen in that section, be able to exchange Setplay-related messages. To help accomplishing this task, the Framework provides three simple tools, available as methods, and depicted in Fig. 3.8:

Need to communicate: to check if the agent should send a message to its teammates at the present moment, it should use the corresponding boolean method (short name *willCommunicate*). In this situation, the user/developer should ensure the sending of the necessary message, as returned by the next method.

Message needing to be sent: if there is the need to communicate, the relevant message is returned by the method *messageToSend*.

Process received Message: on the side of the message receivers, each time a new message is received, its content should be made known to the Framework, through the invocation of method *processReceivedMessage*, which is available both with a string and a message as argument.

In this case of Communication management, the need to offer alternative usage profiles was not deemed necessary, and thus a single set of tools is offered to the user/developer.

3.3.4 Summary

The Framework was developed to be used with possibly different profiles, in order to make the integration in previously existent code easier. There are two subjects where the usage of the Framework can be done in different ways: Setplay selection and activation, and *Action* selection and execution. In both cases, the Framework provides tools for a automatic execution, or for a more detailed and interactive deployment. It is up to the user/developer to choose the option that best fits his needs. The choice in Setplay selection is independent from the one in action selection and execution.

3.4 Wrapper classes

The Setplay framework aims at being applied to different teams and leagues. The framework was thus designed in a modular and easy to integrate way.

One important issue is that, while being used, the Framework, depending on the way it is used and according to the usage profiles described previously, will need to gain access to some of the league-specific features: it will need to be able to evaluate *Conditions*, e.g.,

to check if a Setplay is feasible by evaluating its pre-conditions, and it might also need to order *Actions* to be executed, when actually running a Setplay. This was seen in more detail in the previous section.

The Framework was thus designed with two wrapper classes, upon which the execution of Setplays depends. These are two abstract classes, which must be implemented by the user/developer with league and team specific code, in order for the Setplay Framework to be able to properly perform in this environment.

These two abstract classes are called *Context* and *Action::Executor*. The first one should encapsulate code that models the State-of-the-World, and must be able to evaluate the existing *Conditions*. This abstract class has thus a method, *lookup*, with signatures accepting as arguments all the possible *Conditions*. This way, when the Framework is for instance evaluating a Setplay's pre-conditions, it will be able to ground these abstract concepts and, e.g., be able to lookup if some player has ball possession, or if the team has scored more than two goals.

The *Context* class also has some methods to check elementary data about the game, such as positions of ball and players, field dimensions and game time. This information is frequently used in Setplay management, and must therefore be available at any time.

As for *Action::Executor*, it is also an abstract class and, as the name suggests, it should execute actions, thus encapsulating the code that is able to execute all the available abstract *Actions*.

To better characterize these two classes, the UML diagram in Fig. 3.9 was drawn.

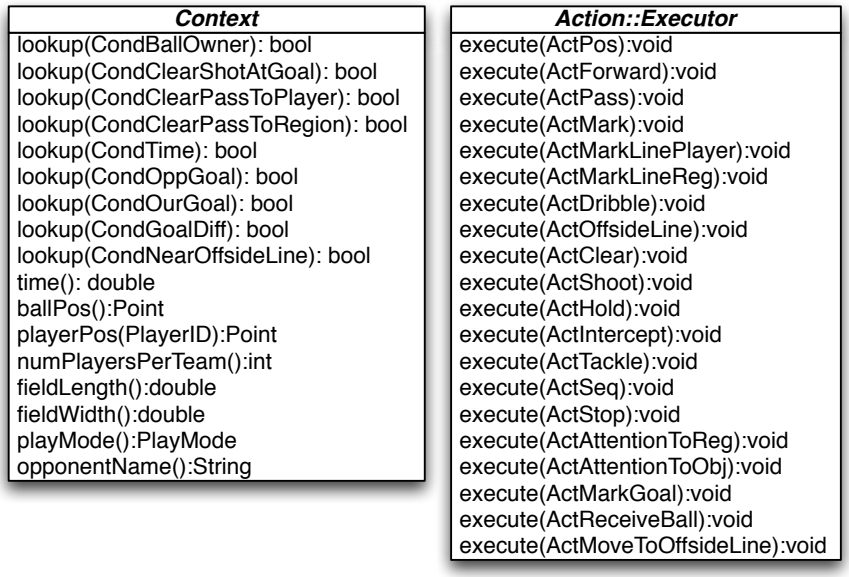


Figure 3.9: Context and Executor Wrapper classes

3.5 Operations on Setplays

In the previous section, the concepts pertaining to the domain of Setplays were introduced. With these, it is possible to describe a Setplay as a small plan involving several players, that follow steps according to the Setplay definition.

In order to use these Setplays in a real soccer playing environment, some operations were defined to ease their application. The present section describes in detail the main operations available, whose integration in classes can be seen in Fig. 3.8. Some of these operations were mentioned before, but will be looked upon here once more, to better characterize their integration in the available classes.

3.5.1 Setplay management

Typically, a team using this framework will have a set of Setplays active in a game. These Setplays need to be managed in some way. To cope with this need, the class *SetplayManager* was created. This class will store the library of Setplays, and offer some helpful operations, provided as methods of this class:

activateSetplay: when a player receives a Setplay activation message (*InitMessage*, see details in section 3.2), it must call this method, with the message as argument. The method will deal with the message and, if the player is in the Participant list, will activate the selected Setplay accordingly.

feasibleSetplays: at any point during the game, this method, with the *Context* as argument, can be used to check for Setplays that are feasible at that moment.

isSetplayActive: checks if there is any Setplay being executed.

playerParticipates: checks if the (self) player participates in the Setplay being executed.

Several other methods were created to ease the management of Setplays, and also to allow debugging procedures, but they are too numerous to be described here, and thus the interested reader is recommended to look at the documentation available within the source code.

3.5.2 Setplay duplication

The class *SetplayManager* keeps a library of available Setplays, which can be consulted and tested for feasibility, as described in the previous section. After the decision of starting a Setplay, it does still have to be instantiated, i.e., its arguments (namely the participants)

must be given some value. Since this instantiation is only valid while the Setplay is being executed, it was considered wiser to create a copy of the stored Setplay and only instantiate that copy. This way, after that Setplay is finished or aborted, it can simply be discarded.

Setplays can have varying number of components, like *Steps*, *Participants* or *Conditions*, which led these components to be designed making use of dynamic data types (pointers), thus needing special care when being duplicated. This operation is done by the method *deepCopy* of class *Setplay*. Similar methods will be called on all Setplay components, assuring that a clean copy, with no shared memory, is created. Such a copy can safely be instantiated, without compromising the original definition.

3.5.3 Setplay inversion

Some Setplays are designed in a way that makes them valid only on one side of the field. As an example, a Setplay for playing a kick-in on the left side of the pitch may require a second player some distance to the right of the ball. Such a Setplay could theoretically also be used for kick-ins on the right side of the pitch, but that same second player would have to position itself to the left side of the ball instead. The whole Setplay could thus be labelled as *invertible*, which would mean it could be subject of a operation of inversion. Such operation will create a new, independent Setplay, and is performed by the method *inversion* (with no arguments), that will mirror all spatial references and offsets with respect to the x-axis. This operation will also change both the Setplay name, adding a '*_inv*' suffix, and its *id*, which will be inverted to a negative integer.

This operation will also have to be operated on all Setplay components, since *Steps* and *Conditions* like ball position may also contain spatial references. The operation will thus be performed in cascade, from the upper level, through all *Steps*, down to the deepest components.

3.6 Setplay Language

Setplays are meant to be freely definable through text files that can be read upon agent launching. To accomplish this goal, Setplays must have a defined syntax on which to base both the writing of Setplays and the parser which will load them. In order to keep this syntax familiar to the RoboCup community, it was based on s-expressions (Rivest, 1997), which are already the building blocks of the communication language with the server and with the coaching language, both from the Soccer Simulation 2D league.

The syntax of this Setplay definition language, written as a set of BNF statements (Naur et al., 1960), is included bellow. In terms of BNF syntax, terminal symbols are in regular type, non terminals in italics between the '<' and '>' symbols, and BNF specific symbols in bold, with the following definitions being used:

::= : definition;

| : alternative;

? : 0 or 1 occurrences of the previous term;

***** : any amount of occurrences of the previous term;

+ : 1 or more occurrences of the previous term.

Some comments will be inserted amidst the text to clarify some options. Generally speaking, the language tried to use the concepts already present in CLang (Chen et al., 2003), the standard coach language in the 2D Simulation league. In some cases, though, practical experience showed that additional concepts were necessary, which were added to the language.

In general terms, one should emphasize the following options:

- All references to players are done through the type PLAYER_REFERENCE;
- All operators (+, -, *, /, ==, !=, <, <=, >, >=) are prefix;
- Arguments of objects and functions have added labels, prefixed by a colon (e.g. *:label*);
- All lists are built through the s-expression with the functor *list* , like, e.g., *(list <args>)*.

At general level, one will consider a Setplay a conjunction of a *Parameter* list, a *Player Reference* list, an *Abort Condition* and a *Step* list, along with the Setplay's name and id, and indication if it is invertible.

```
<SETPLAY>::= (setplay :name <CLANG_STR> :id <INTEGER>
   :invertible <BOOL> <PLAYER_REFERENCE_DEFINITION_LIST>
   <PARAMETER_DEFINITION_LIST>?
   :abortCond <CONDITION> <STEP_LIST>)
```

Parameters will be characterized by their name and type. The available types are only numbers and spatial entities (points and regions).

```
<PARAMETER_DEFINITION_LIST>::=
   :parameters (list <PARAMETER_DEFINITION>+)
   <PARAMETER_DEFINITION>::= (parameter :name <VAR> :type <TYPE>)
   <TYPE>::= integer | decimal | region | point
```

Players can be referred to in two different ways: either through a full identification by team and jersey number, or through the name of the role they will play in the Setplay. When in the scope of the parameter definition, the full syntax of player roles and player identification must be employed, as defined in *PLAYER_REFERENCE_DEFINITION*. In other situations, inside the Setplay definition, player roles can simply be referred to by their names, as per *PLAYER_REFERENCE* definition.

```
<PLAYER_REFERENCE_DEFINITION_LIST>::=
   :players (list <PLAYER_REFERENCE_DEFINITION>+)
<PLAYER_REFERENCE_DEFINITION>::= (playerRole :name <VAR>)
   | (player :team <TEAM> :number <PLAYER_NUM>)
<TEAM>::= our | opp
<PLAYER_NUM>::= [0-9] | 10 | 11
<PLAYER_REFERENCE_LIST>::= (list <PLAYER_REFERENCE>+ )
<PLAYER_REFERENCE>::= <VAR>
   | (player :team <TEAM> :number <PLAYER_NUM>)
```

Steps are, as said before, the main building blocks of Setplays, where they represent intermediary stages in Setplay execution. As such, *Steps* must be precisely characterized. Their *id* allows *Steps* to be referred in *Transitions* from other *Steps*. *WaitTime* represents the time that needs to elapse before trying to transition to another *Step*, while *abortTime* is the time after which the Setplay is to be aborted if no transition to another *Step* is possible. Both time intervals must be non-negative integers. The *condition* must be satisfied before entering the *Step*. The *Step* also contains a list of *Participants* and *Transitions*, described hereafter.

```
<STEP_LIST>::= (list <STEP>+ )
<STEP>::= (step :id <NON_NEG_INTEGER>
   :waitTime <NON_NEG_INTEGER> :abortTime <NON_NEG_INTEGER>
   :participants <PARTICIPATION_LIST> :condition <CONDITION>
   :transitions <TRANSITION_LIST>)
```

A *Participation* identifies a player that takes part in the *Step*, and may optionally also determine the player's desired location for this *Step*, which shall be done through the usage of the '*at*' functor.

```
<PARTICIPATION_LIST>::= (list <PARTICIPATION>+)
<PARTICIPATION>::= <PLAYER_REFERENCE>
   | (at <PLAYER_REFERENCE> <REGION>)
```

Transitions are the options to move from one Setplay *Step* to another (through *nextStep*), or to finish or abandon Setplays. In the case of transitions to other *Steps*, a list of *Directives* can be used to determine the actions the player should execute, in the case of the *Do* directive, or avoid to execute, in the case of *Dont*.

```
<TRANSITION_LIST>::= (list <TRANSITION>+ ) | <TRANSITION>
<TRANSITION>::= <NEXT_STEP> | <FINISH> | <ABORT>
<NEXT_STEP>::= (nextStep :nextStepNumber <NON_NEG_INTEGER>
   :condition <CONDITION> :directives <DIRECTIVE_LIST>)
<FINISH>::= (finish :condition <CONDITION>)
<ABORT>::= (abort :condition <CONDITION>)
<DIRECTIVE>::= (<DIRECTIVE_NAME> :players <PLAYER_REFERENCE_LIST>
   :actions <ACTION_LIST>)
<DIRECTIVE_NAME>::= do | dont
<DIRECTIVE_LIST>::= (list <DIRECTIVE>+)
```

Actions, in the scope of the Setplay Framework, represent abstract concepts that model the different high-level behaviors a player might need to execute. To keep the maximum compatibility with previously existing concepts, all the actions defined in CLang (Chen et al., 2003) were imported, with similar meanings and syntax. Some new actions had, though, to be introduced, in order to deal with necessities not covered by the original set of actions, as seen in section 3.1.1. These were the following:

receiveBall : receive a pass from another player, which is, implicitly, the ball owner. Takes no arguments.

***attentionTo*(object or region)** : direct visual and/or auditive attention to a particular object or region, taken as argument.

seq : action sequence, is simply an aggregator of actions, that are given as arguments, and should be executed sequentially.

markGoal : position in a location suitable for avoiding opponent shots to enter the goal.

moveToOffSideLine : move near the offside line, avoiding at the same time to cross it. The argument indicates the *y*-coordinate of the desired location.

stop : stop as quick as possible, making sure that any remaining inertia is adequately counter-balanced.

```
<ACTION>::= (pos :region <REGION>)
   | (bto :region <REGION>)
   | (bto :players <PLAYER_REFERENCE_LIST>)
   | (mark :players <PLAYER_REFERENCE_LIST>)
   | (markl :players <PLAYER_REFERENCE_LIST>)
   | (markl :region <REGION>)
   | (markGoal) | (oline :region <REGION>)
   | (clear :region <REGION>) | (hold)
   | (dribble :region <REGION>) | (shoot)
   | (tackle :players <PLAYER_REFERENCE_LIST>) | (intercept)
   | (receiveBall) | (stop) | (attentionTo :region <REGION>)
   | (attentionTo :object <OBJECT_LIST>)
   | (moveToOffSideLine :y <DECIMAL>) | (seq <ACTION>+)
<ACTION_LIST>::= (list <ACTION>+ ) | <ACTION>
```

Objects are further characterized as being mobile or static, and can include all the agents (human and robotic) as well as passive objects that can be possibly located on the pitch. Static objects are not yet exhaustively defined.

```
<OBJECT_LIST>::= (list <OBJECT>+) | <OBJECT>
<OBJECT>::= <STATIC_OBJECT> | <MOBILE_OBJECT>
<STATIC_OBJECT>::= <POST> | <FLAG> | ...
<MOBILE_OBJECT>::= (ball) | <PLAYER_REFERENCE> | (referee)
```

Conditions were originally based on CLang, similarly to *Actions*, and more details about these can be found in section 3.1.2. In this case, though, to better tune the some specific situations, several new *Conditions* had to be created, as follows:

***canPassPl*:** check if some player in the *from* list can execute a pass to some player in the *to* list.

***canPassReg*:** check if some player in the *from* list can execute a pass to the region given as the *to* argument.

***canShoot*:** check if some player in the *players* list can shoot at goal.

***nearOffsideLine*:** check if some player in the *players* list is near the offside line, for situations where a forward behind this line is to be done.

Conditions regarding score and time are done through dedicated keywords and the usual comparison operators.

```
<CONDITION>::= (true) | (false)
   | (ppos :players <PLAYER_REFERENCE_LIST> :min <INTEGER>
   :max <INTEGER> :region <REGION>)
   | (bpos :region <REGION>)
   | (bowner :players <PLAYER_REFERENCE_LIST>)
   | (playm <PLAY_MODE>)
   | (canShoot :players <PLAYER_REFERENCE_LIST>)
   | (canPassPl :from <PLAYER_REFERENCE_LIST>
   :to <PLAYER_REFERENCE_LIST>)
   | (canPassReg :from <PLAYER_REFERENCE_LIST> :to <REGION>)
   | (nearOffsideLine :players <PLAYER_REFERENCE_LIST>)
   | (and <CONDITION_LIST>)
   | (or <CONDITION_LIST>)
   | (not <CONDITION>)
   | (<COND_COMP>)
<COND_COMP>::= <TIME_COMP>
   | <OPP_GOAL_COMP>
   | <OUR_GOAL_COMP>
   | <GOAL_DIFF_COMP>
<TIME_COMP>::= (time <COMP> <INTEGER>)
<OPP_GOAL_COMP>::= (opp_goals <COMP> <INTEGER>)
<OUR_GOAL_COMP>::= (our_goals <COMP> <INTEGER>)
<GOAL_DIFF_COMP>::= (goal_diff <COMP> <INTEGER>)
<COMP>::= < | <= | == | != | >= | >
<PLAY_MODE>::= bko | time_over | play_on | ko_our | ko_opp
   | ki_our | ki_opp | fk_our | fk_opp | ifk_our
   | ifk_opp | ck_our | ck_opp | gk_opp | gk_our
   | gc_our | gc_opp | ag_opp | ag_our
<CONDITION_LIST>::= (list <CONDITION>+ )
```

Regions model sections of the pitch, and are organized around the usual concepts of arcs, triangles and rectangles. In order to use names commonly used for well known regions, a new region type was added (*Named_Region*), which may have a wide range of pre-defined values, inspired in part by the ones defined by Coach-Unilang (Reis and Lau,

2002). Several other areas, partitioning the whole field, were created to be used under the scope of the Setplay selection tool, and more details on these are given in section 4.2.1.

As for points, there is a wide number of different operators to refer to static and dynamic points, as well as to perform some operations, namely translations, on these.

```
<REGION>::= <VAR> | (regVar :name <CLANG_STR> )
   | (regVar :name <CLANG_STR> :value <REGION>)
   | <POINT> | (regNamed :name <REGION_NAME>)
   | (arc :center <POINT> :radius_small <REAL>
   :radius_large <REAL> :angle_begin <REAL> :angle_span <REAL>)
   | (triang :points (list <POINT> <POINT> <POINT>))
   | (rec :points (list <POINT> <POINT>))
<REGION_NAME>::= field | outside | our_middle_field |
   their_middle_field | left | right | far_left |
   mid_left | centre_left | centre_right |
   mid_right | far_right | our_back |
   our_middle | our_front | their_back |
   their_middle | their_front |
   sl_1 | sl_2 | sl_3 | sl_4 | sl_5 | sl_6 |
   sl_7 | sl_8 | sl_9 | sr_1 | sr_2 | sr_3 |
   sr_4 | sr_5 | sr_6 | sr_7 | sr_8 | sr_9 |
   our_penalty_box | our_goalie_area |
   their_penalty_box | their_goalie_area
<POINT>::= <VAR> | (ptVar :name <CLANG_STR> )
   | (ptVar :name <CLANG_STR> :value <POINT> )
   | (pt :x <REAL> :y <REAL>) | (pt ball)
   | (pt :player <PLAYER_REFERENCE>)
   | (ptRel :x <REAL> :y <REAL> :pt <POINT>) | (<POINT_ARITH>)
<POINT_ARITH>::= (<OP> <POINT> <POINT>)
<OP>::= + | - | * | /
<REGION_LIST>::= (list <REGION>+)
<CLANG_STR>::= "[0-9A-Za-z().+-*/?<>_]+"
<VAR>::= [A-Z][a-zA-Z0-9]*
```

3.7 Setplay Example

In this section, a simple example is presented, to illustrate the actual definition of *Setplays*, how the players (in this example, in the 2D Simulation league) deal with it, and how inter-robot communication is deployed. The described Setplay was defined in a configuration

file, read upon player start-up, as seen in Fig. 3.10. The communication protocol and the syntax of the deployed messages can be seen in section 3.3.3.

A situation where a Setplay can be properly used is the corner-kick: it is an offensive situation close to the opponent goal and holes in the defense can be exploited. To keep this example clear, a simple situation, with only three participants, will be described, and can be seen in Fig. 3.11. This Setplay's script is simple: there are three participating players, with the corner kick taker placing itself near the ball, the shooter in front of the goal, and another player, the receiver, between them. In Step 0, the ball is passed by the corner kick taker to the receiver. Next, in Step 1, the receiver makes a second pass towards the shooter. When this player gets ball possession, Step 3 is reached, and the ball is shot at goal. Step 4 is reached if a goal is actually scored and the corresponding play mode is reached.

One will now look at the Setplays execution with more detail, and elaborate on the exchanged messages. The Setplay initiator triggers execution, by sending an instantiation message, where `S0` identifies the start of the chosen Setplay with number 0, representing its identification number (*id*). In this case, the lead player is the corner kick taker nr. 10 (taking role *cornerP*), and the two other participants nrs. 7 and 11 (roles *receiver* and *shooter*). This message can be constructed by creating a new object of the *InitMessage* type. When converted to plain text, the message would be as follows:

```
(startSetplay :setplayID 0 :participants 10 7 11 :parameters )
```

In step 0 of the Setplay, the participating players reach their positions, after which the *cornerP* tries to reach step 1, passing the ball to the *receiver*, as depicted in Fig. 3.11(a). The figure shows some execution monitoring messages printed on top of the actual game, which is possible with help of the FCPortugal debugger. One can thus see who the lead player is trying to pass the ball to, and where it is looking at, which in this case typically coincides. This phase of the Setplay's execution would be signaled by a *StepMessage*, which, in a textual representation, would look as follows:

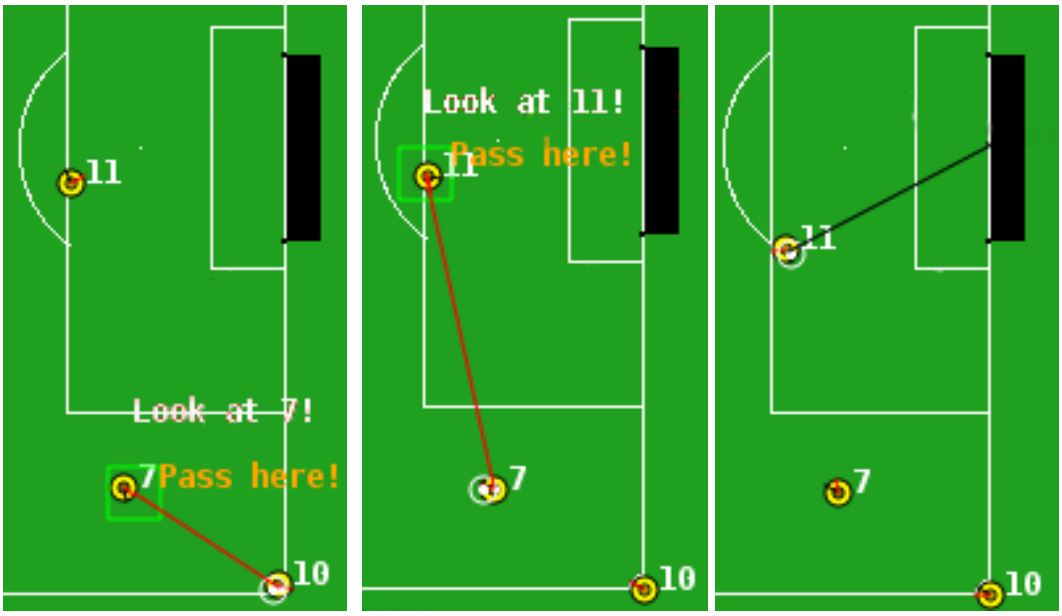
```
(stepMessage :currentStep 0 :nextStep 1)
```

Upon gaining possession of the ball, *receiver* starts being the new lead player and immediately sends a message to the other players, informing them that the Setplay is currently in step 1, and that the *receiver* will try to reach step 2, as follows:

```
(stepMessage :currentStep 1 :nextStep 2)
```

```
(setplay :name simpleCorner :id 3 :invertible true
  :players (list (playerRole :roleName cornerP)
       (playerRole :roleName receiver)
       (playerRole :roleName shooter))
   :steps (seq
     (step :id 0 :waitTime 15 :abortTime 70
       :participants (list (at cornerP (pt :x 52 :y 34))
            (at receiver (pt :x 40 :y 25))
            (at shooter (pt :x 36 :y 2)))
       :condition (playm ck_our) :leadPlayer cornerP
       :transitions (list
            (nextStep :id 1 :condition (canPassPl
              :from cornerP :to receiver)
              :directives (list
                (do :players cornerP
                     :actions (bto :players receiver))
                (do :players receiver
                     :actions (receiveBall))))))
     (step :id 1 :waitTime 5 :abortTime 70
       :participants (list (at receiver (pt :x 40 :y 25))
            (at shooter (pt :x 36 :y 2)))
       :condition (and (bowner :players receiver)
            (playm play_on))
       :leadPlayer receiver
       :transitions (list
            (nextStep :id 2
              :condition (canPassPl :from receiver
                :to shooter)
              :directives (list
                (do :players receiver
                     :actions (bto :players shooter))
                (do :players shooter
                     :actions (receiveBall))))))
     (step :id 2 :abortTime 70
       :participants (list (at shooter (pt :x 36 :y 2)))
       :condition (and (bowner :players shooter)
            (playm play_on) )
       :leadPlayer shooter
       :transitions (list
            (nextStep :id 3
              :condition (canShoot :players shooter)
              :directives (list (do :players shooter
                :actions (shoot))))))
     (step :id 3
       :participants (list shooter) :condition (playm ag_our)
       :leadPlayer shooter :transitions (list (finish)))))
```

Figure 3.10: Corner Setplay definition



(a) (b) (c)

Figure 3.11: Corner Setplay execution steps.

When the *receiver* verifies that it can make a pass to the *shooter*, it will do so, as depicted in Fig.3.11(b). In this figure, the *receiver* is looking at the *shooter* in order to accomplish a good pass, as it was the case in the precedent image. At this point of execution, the *receiver* will try to pass the ball to the *shooter*, which will be signaled through the following message:

```
(stepMessage :currentStep 2 :nextStep 3)
```

Finally, as soon as it considers that shooting at goal is possible, the *shooter* executes the shot (see Fig. 3.11(c)) and moves to step 3, which simply finishes the Setplay. At this moment, the *shooter* will send a message stating that it reached step 3, and that there is no further step in the Setplay:

```
(stepMessage :currentStep 3 :nextStep -1)
```

3.8 Summary

This chapter presented the general design of The Setplay Framework, as well as its organization through concepts and related tools. Details were given on the underlying language and the related communication model. As the Framework was designed to be used in

different teams, several usage profiles were envisaged. An example of a simple Setplay was also presented to aid the understanding of the Framework's functioning.

In the next chapter, the implementation of the Framework and its related tools is described, while on the following, chapter 5, the application of the Framework to different teams is looked upon in detail. The results of these application processes are considered critically in chapter 6.

Chapter 4

Framework Implementation

The Framework's outline and objectives, described in chapter 3, demand a modular component to be developed, comprising all the data structures, algorithms and tools that will support the definition and execution of Setplays. This module is meant to be applied to arbitrary robotic soccer teams through an as simple as possible process.

This chapter will describe the development of the Framework module, and the related tools. The next chapter, in turn, will describe the processes that led the Framework to be applied to different teams, in the Simulation League, as well as in the Middle-size league.

4.1 Implementation of a C++ library

The Setplay Framework was, from the beginning on, intended to be applicable to different teams and leagues: the Framework should be applied in different leagues and, further, it should be possible to mix players from different originating teams in one single, mixed team, through the execution of Setplays.

Since the initial implementation and testing were planned to be conducted on top of the FCPortugal (Simulation) and CAMBADA (Middle-size) teams, both implemented using C++, it was decided to develop a C++ library, with two goals: ease the application to these teams, or any other implemented in C++; and maintain a common, stable framework that would be applied to all teams.

The implementation intends to provide as much tools as possible, and therefore the following features were developed:

Setplay definition parser Since Setplays can be freely defined according to the model described in the previous chapter, or edited using a separate graphical tool (see section 4.3), it was obviously necessary to develop a parser to load files and read them into C++ objects. Details in section 4.1.1.

Setplay participants selection Alternative algorithms to select the Setplay's participants, according to their positions, have been developed. Details in section 4.1.2.

Setplay execution engine A Setplay execution is determined by its definition and real-time instantiation. Therefore, the framework provides an execution engine to be easily used, out-of-the-box. Details in section 4.1.3.

Setplay evaluation and selection At different moments in the game, one will need to decide whether to start the execution of a Setplay, and choose the most appropriate, when more than one is valid at the moment. With this purpose, a Case Based Reasoning inspired tool was developed, and is described in section 4.2.

Setplay graphical editor To aid the definition of Setplays, and allow users to edit these in a graphical environment, a stand-alone editor (*SPlanner*) was developed, which assists the definition of Setplays through the positioning of players on the field and the design of their moves and actions on the pitch. This tool is described in section 4.3.

With these tools, namely the Setplay evaluation and selection mechanism described in section 4.2, a team using the Framework does not have to worry about the execution details of a Setplay: it simply has to implement the abstract actions and conditions (see next chapter), and create the necessary Setplays, eventually using the graphical editor described in section 4.3.

The rest of this section will describe in more detail, but still from a high level point of view, how the Framework was developed as a library and some of the more important design options.

4.1.1 Setplay definition parser

Setplays were designed to be freely configurable and easily changeable. It was therefore out of the question to let them be, somehow, hard-coded inside the agent, since that would compromise these objectives. To accomplish these requirements, it was chosen to let Setplays be defined in an external configuration file, using a well established syntax, already presented in section 3.6. This way, Setplays can be manually edited, or created through the tool *SPlanner*, which saves them as text files.

This design option requires the Framework to be equipped with a parser that efficiently reads the (possibly extensive) configuration file, that will typically include several Setplay definitions.

This parser was, in a first effort, developed from scratch, using C++'s native methods to handle strings. This proved to be a difficult, time-consuming and error-prone developing process. In spite of this, the parser was fully developed and used in the initial versions of the Framework. However, when later there were changes to the Framework's model (e.g., new *Actions* and *Conditions*), one verified that this parser was hard to change and easy to corrupt. At this point, it was considered that this kind of parser was generally a too rudimentary tool.

This experience led to the redesign of the whole parser, this time built upon a well tested parser tool. From the different options considered, the Spirit library (de Guzman and Nuffer, 2003)[1], available as part of the latest Boost (Walker, 2003)[2] distributions, was chosen. It is a quite simple tool which allowed the rapid prototyping, and later full development, of the necessary parser, used in the final version. The same tool was used to parse the messages exchanged by the agents during Setplay execution.

4.1.2 Selection of players participating in a Setplay

After selecting a Setplay for execution, one subsequent crucial choice is the selection of the players participating in the Setplay. Since Setplays can, at each *Step*, include the intended position for each participating player, the positions in *Step 0* can be seen as the Setplay's initial positions. The participants' choice is important, since a bad or aleatory choice could result in selecting distant players, which would take long, and spend a lot of energy, to reach the initial Setplay positions. Such a delay could jeopardize the Setplay's success, and compromise the players' energy reserves.

4.1.2.1 Selection from Distance to lead player

To cope with this question, several different selection algorithms were developed. Initially, a simple algorithm chose the players according to their distance to the player that starts the Setplay, i.e., the lead player (method name: *participatingPlayersDistanceFromMe*). This algorithm, naturally, may only be used if the lead player is one of the field players: it makes no sense when the lead player is an intangible agent like the coach. This algorithm pays no attention to the Setplay's initial positions: it simply iterates the participants list and allocates available players according to their distance to the lead player. All possible players are considered as candidates, excluding the lead player, which is already participating, and the Goalie, which is left out, when not the lead player, since it should

[1] URL: `http://boost-spirit.com/`
[2] URL: `http://www.boost.org/`

always keep full attention to its particular tasks. The algorithm of this selection method can be seen in Alg. 1.

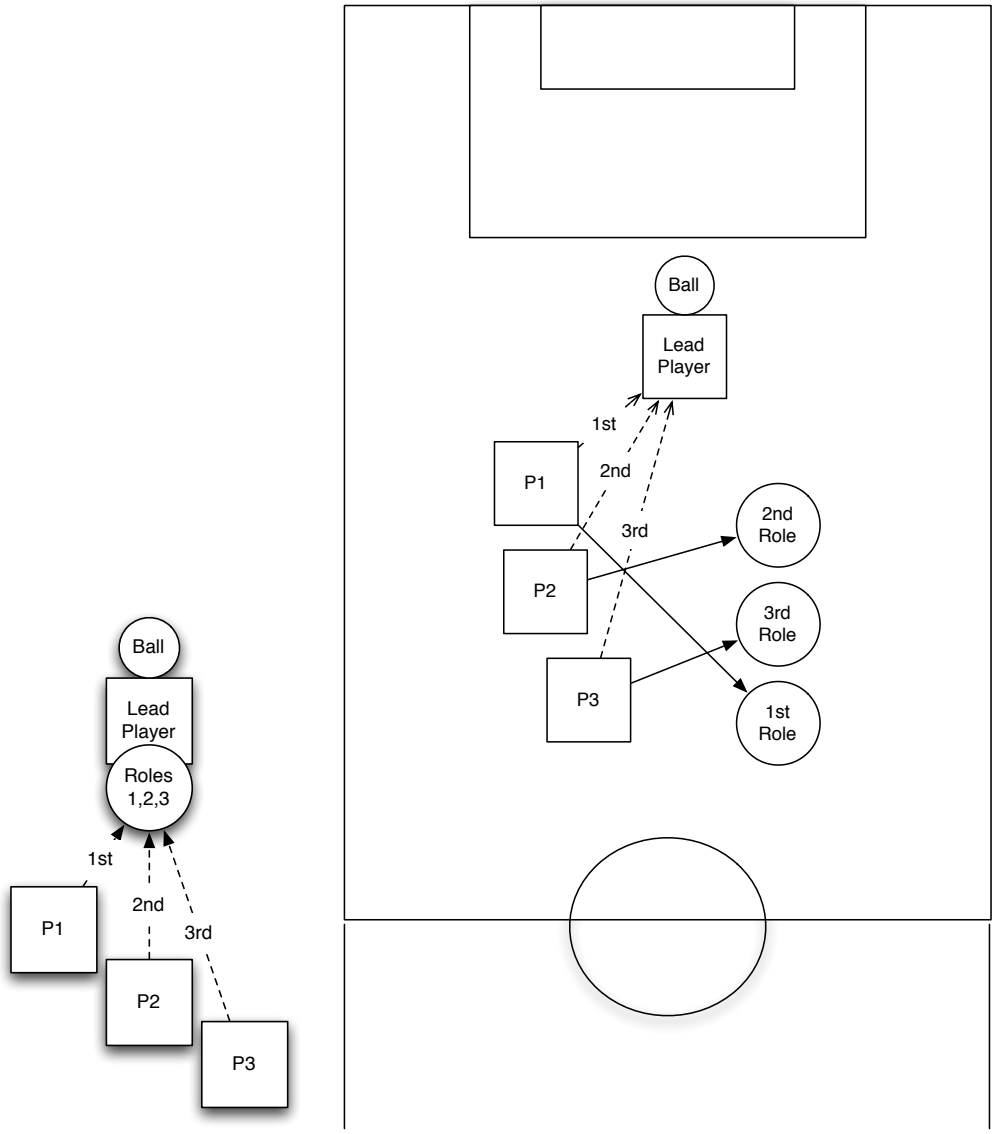
Algorithm 1 Select $numRoles(s)$ participants for Setplay s by distance to lead player

Require: $(lead(goalie) \wedge numRoles(s) \leq numPlayers) \vee (\neg lead(goalie) \wedge numRoles(s) \leq numPlayers - 1)$
  if $lead(goalie)$ **then**
    $availablePlayersList \leftarrow$ all players except goalie
  else
    $availablePlayersList \leftarrow$ all players except goalie and lead player
  end if
  $chosenPlayersList \leftarrow \{leadPlayer\}$
  $roles \leftarrow roles(s)$ except $leadRole(s)$
  for all role r in $roles$ **do**
    $chosenPlayer \leftarrow$ player in $availablePlayersList$ closest to $leadPlayer$
    remove $chosenPlayer$ from $availablePlayersList$
    add $chosenPlayer$ to the end of $chosenPlayersList$
  end for
  return $chosenPlayersList$

Two scenario of this algorithm's usage can be seen in Fig. 4.1. In this case, as mentioned before, the position of roles is not considered in player selection, and thus their position in the diagram is meaningless. In Fig. 4.1(a) one sees in which order players are considered for selection: by distance from the lead player. Fig. 4.1(b) shows how this kind of selection might prove inefficient in particular situations, e.g. when roles are ordered in the Setplay participant's list inversely to their distance from the lead player. In this figure, distances to the lead player are shown in dashed arrows, and allocation of players to roles in regular arrows.

4.1.2.2 Selection from Player's individual distance to Setplay positions

Since the initial selection algorithm was far from optimal, a new, greedy strategy was implemented, considering the distance to the Setplay's initial positions (method name: *participatingPlayersDistanceFromPositions*). In this case, the participant's list is iterated, and the available player closest to each position is chosen immediately. This means that the algorithm is greedy, which makes it sub-optimal: the distances are considered, in general a good allocation is chosen, but there is no guarantee that this is the case. In the example shown in Fig. 4.2, the selection is not optimal, and the paths the players have to follow to reach the Setplay's positions intersect, which may cause collisions and consequent delays in positioning. This can be particularly important in leagues using real robots, like the Middle-size league.



(a) (b)

Figure 4.1: Player Selection from distance to Lead Player

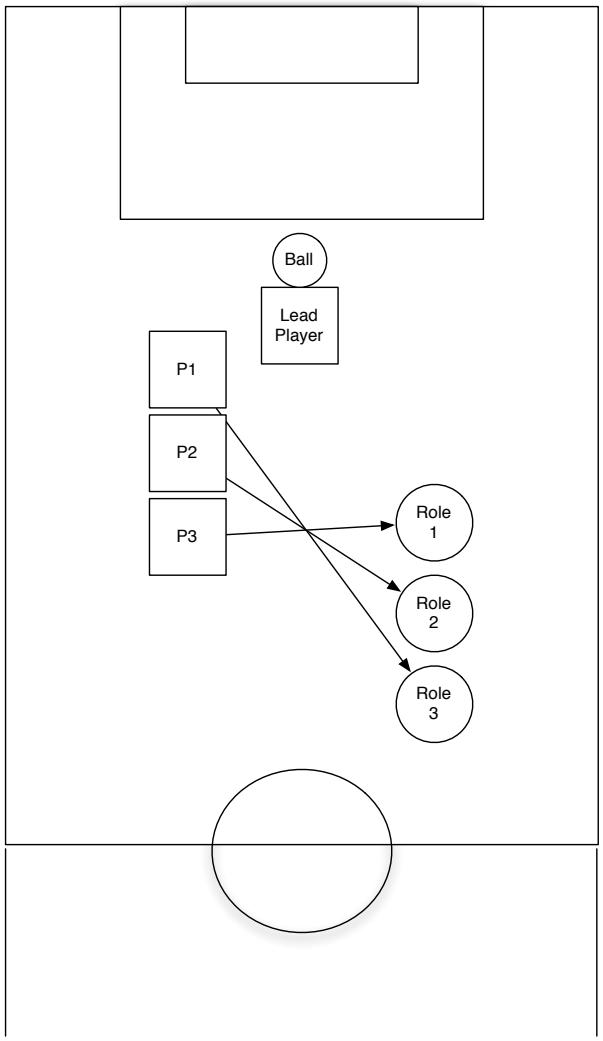


Figure 4.2: Player Selection from distance to Setplay Positions

The algorithm for this selection strategy is as follows:

Algorithm 2 Select $numRoles(s)$ participants for Setplay s by distance to each role's position

Require: $(lead(goalie) \wedge numRoles(s) \leq numPlayers) \vee (\neg lead(goalie) \wedge numRoles(s) \leq numPlayers - 1)$
  if $lead(goalie)$ **then**
    availablePlayersList $\leftarrow$ all players except goalie
  else
    availablePlayersList $\leftarrow$ all players except goalie and lead player
  end if
  chosenPlayersList $\leftarrow \{leadPlayer\}$
  $roles \leftarrow roles(s)$ except $leadRole(s)$
  for all role r in $roles(s)$ **do**
    chosenPlayer $\leftarrow$ player in *availablePlayersList* closest to $position(r)$
    remove *chosenPlayer* from *availablePlayersList*
    add *chosenPlayer* to the end of *chosenPlayersList*
  end for
  return *chosenPlayersList*

4.1.2.3 Player global distance to Setplay positions

In order to overcome the weaknesses identified with the second approach, a third alternative algorithm was developed. Each possible player arrangement's summed distances to the Setplay's initial positions are computed. The set of players with the minimum distance is kept as the solution. With this algorithm, the case presented previously would have a different selection, minimizing the global path distance (method name: *participatingPlayersGlobalDistanceFromPositions*). This strategy is described in Alg. 3

Naturally, this algorithm might prove to be computationally too demanding for Setplays with many participating players: a Setplay with 6 participants will possibly have 15 120 player arrangements in teams with 11 players. Therefore, this algorithm should not be applied in this kind of situations, since it might hinder timely response from the player choosing the participants.

In the two algorithms that involve positions in the selection process, the actual player's location is consulted through a method of the class that represents the State-of-the-World (*Context*). This means that the positions considered correspond to the present location of the players. In some cases, though, it can be useful to consider other positions, like strategic positions, during the selection process (see section 3.3.1.1). To cater to this kind of needs, two other versions of these algorithms were created: the rationale behind them is exactly the same, but the positions considered for calculation are sent as argument to

Algorithm 3 Select $numRoles(s)$ participants for Setplay s by global distance to all roles' positions

Require: $(lead(goalie) \wedge numRoles(s) \leq numPlayers) \vee (\neg lead(goalie) \wedge numRoles(s) \leq numPlayers - 1)$
  if $lead(goalie)$ **then**
    $availablePlayersList \leftarrow$ all players except goalie
  else
    $availablePlayersList \leftarrow$ all players except goalie and lead player
  end if
  $chosenPlayersList \leftarrow \{leadPlayer\}$
  $currentMinDistance \leftarrow \infty$
  $roles \leftarrow roles(s)$ except $leadPlayer$
  for all sequence seq of distinct $numRoles(s)$ players in $availablePlayersList$ **do**
    $tempDistance \leftarrow 0$
    for $i \leftarrow 0$ **to** size of $roles$ **do**
      $r_i \leftarrow i^{th}$ element in $roles$
      $seq_i \leftarrow i^{th}$ element in seq
      $tempDistance \leftarrow tempDistance + distance(position(r_i), position(seq_i))$
    end for
    if $tempDistance < currentMinDistance$ **then**
      $currentMinDistance \leftarrow tempDistance$
      $chosenPlayersList \leftarrow seq$
    end if
  end for
  return $chosenPlayersList$

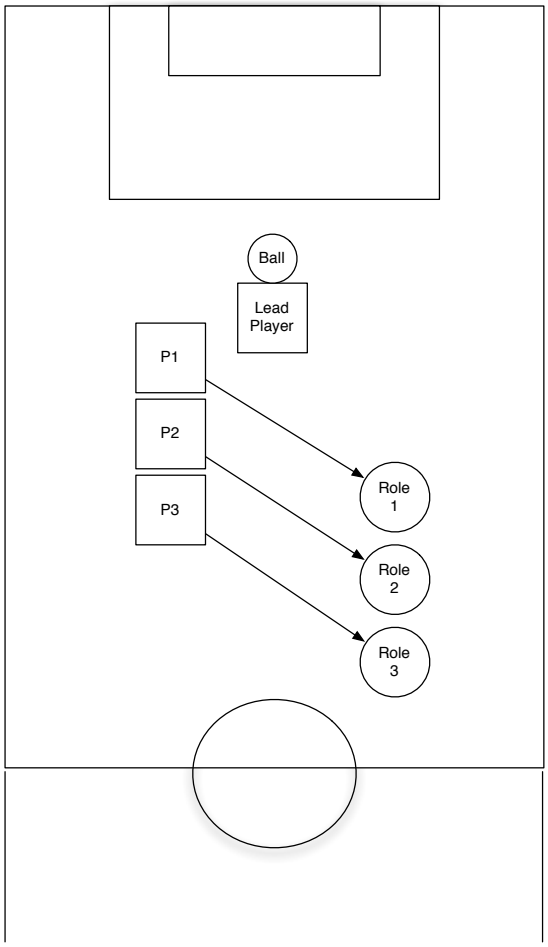


Figure 4.3: Player Selection from global distance to Setplay Positions

the method (method names: *participatingPlayersDistanceFromPosAsArg* and *participatingPlayersGlobalDistanceFromPosAsArg*).

To better describe the different algorithms behavior, Fig. 4.4 shows an example situation, and the different players' choice: the selection from distance to the lead player is shown in dashed arrows, while the greedy selection from players' distances is shown in doted arrows, and the selection according to the global distances is represented through thick arrows. From the arrows' lengths, one can verify that the latter algorithm is the most efficient.

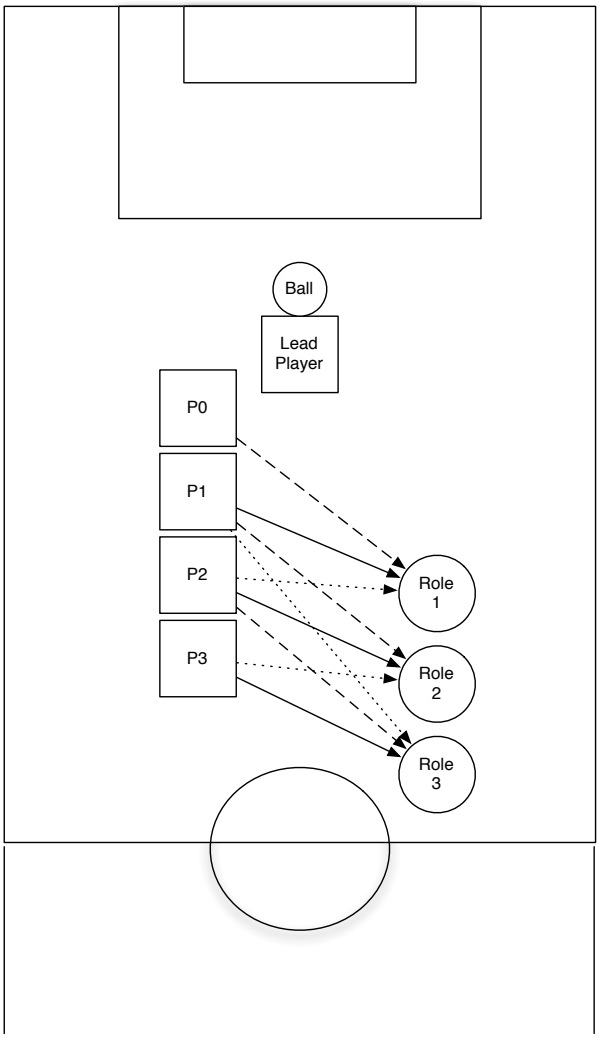


Figure 4.4: Player Selection by all algorithms: distance to lead player (dashed arrows), distance to individual roles (dotted), global distance to positions (thick).

4.1.3 Setplay execution engine

After choosing a Setplay to be executed, and the corresponding participating players (as well as, eventually, other parameters in the Setplay), the Framework will be ready to actually execute the Setplay, according to its definition. This execution will be primarily managed by the Framework itself, but some necessary tasks have to be triggered, as described in section 3.3. The Framework needs thus to support these tasks, supplying the corresponding methods.

The first necessary task is to activate the chosen Setplay, which can be done in different ways, as described in the aforementioned section 3.3. In any case, this process of activation will be done through the execution of two sub-tasks:

1. creation of a copy of the chosen Setplay, in order to keep the original definition unchanged and to allow the unrestrained manipulation of the copy;

2. instantiation of the copy's participants and other parameters, in order to ground the Setplay's arguments to actual values.

These two tasks are easily and efficiently accomplished, since the Framework's structure was designed with them in mind: the copy of the Setplay is simply done through the invocation of the corresponding dedicated method (*deepCopy*); the instantiation of parameters is done by a simple attribution of the new values, through the corresponding methods (*set*). Since all references to each parameter were linked to the same instance during the parsing process, the attribution of new values will have global repercussion.

After the activation, the Setplay is ready to be executed. This execution consists mainly of three tasks:

Monitoring of the State-of-the-World: by checking the *Conditions* existent in the Setplay's definition: *finish*, *abort* and *Step* conditions. When some of these conditions are satisfied, the internal state of the Setplay will change, by, respectively, abandoning the Setplay and reaching a new *Step*. This reaction to changes in the State-of-the-World is performed through the method *updateInternalState*.

Transition choice: Setplays execution might progress from a *Step* to several other, through alternative *Transitions*. These can be followed only when their *Conditions* are satisfied, which might happen simultaneously to several *Transitions*. When this happens, there are several valid *Transitions*, and thus the user/developer must choose among these. When there is only one (valid) *Transition*, then the choice is obviously trivial.

Execution of actions: every time the Setplay's definition determines it, the participating players will need to execute the corresponding actions. This must be done in one of the alternative ways described in section 3.3.

To sum-up the description of the Setplay execution process, one can observe that the progress through the Setplay steps is conducted internally by the Framework, which keeps track of the updates in the State-of-the-World which imply changes in the Setplays internal state. This management must, though, be triggered from outside the Framework at certain moments, by the player invoking the corresponding method. When there are alternative *Transitions* between *Steps*, the user/developer must choose among the valid ones.

4.2 Setplay Evaluation and Selection: Case-based Reasoning

For the Setplay Framework to be as self-sufficient as possible, it must provide a way to automatically evaluate and select Setplays at run-time. Such a mechanism can be very useful for the Framework's deployment in teams with no or little expertise in using Setplays. Such a mechanism should function autonomously, by choosing the best Setplay to perform in each moment. Naturally, this choice is a very difficult one, since it depends on an enormous set of factors, namely: opponent style and game quality, positioning on the field, momentary positioning of opponents, player's types and stamina, etc.

Several options could be considered to tackle this problem. Initially, when first considering this challenge, it was thought that, at each moment, one could consider the available Setplays and evaluate two different issues: on the one hand, the probability of Setplay's success (P_{S_n}), estimated, e.g, by comparing the predicted execution with the opponent players positions; and, on the other hand, the Setplay outcome's utility (Out_{S_n}), such that a goal would get a very high score, a simple progression on the field a low positive score, and a kick back to our goalie a negative score, etc. The combined evaluation of these two issues would sum up to be the Setplay's evaluation, and could be used to choose among alternative, possible Setplays. The evaluation formula for Setplay n (S_n) would look as follows:

$Eval(S_n) = P_{S_n} \cdot Out_{S_n}$

This kind of evaluation is not based on previous experience: it relies on estimations of the success of passes, shots and other moves, that depend on the topological relations of players and other objects, as well as on the prediction of players' behavior. Such an analytical algorithm would, thus, be very difficult to develop and implement, and would

have limited applicability: even if it proved to be appropriate to a particular team or league, it might be useless in other settings with different characteristics. For instance, the probability of ball interception depends highly on the league's and team's characteristics, and would be very difficult to abstract to the whole RoboCup domain.

As this initial idea seemed very difficult to develop, a more empirical algorithm was deemed necessary. This algorithm should perform an evaluation of Setplays, based on its previous performance, both in past and the current games. Setplays with a higher success rate would be chosen with higher priority, while unsuccessful ones would only be chosen rarely, to assure that they would still have a chance to be executed, in order to check if they have had their performance enhanced, for some circumstantial reason, like changes in opponent's game strategy. This mechanism would, thus, base the choice on past experiments' success.

A mechanism with such characteristics can be inspired by Case-Based Reasoning (CBR) principles (Aamodt and Plaza, 1994). CBR systems record past experiments or reasoning processes as cases, in a so-called Case Base, the common name for the Knowledge Bases underlying this kind of systems. When a new situation (commonly referred to as problem) arises, the CBR reasoner searches the Case Base for a case similar to the new problem. If it manages to find a case similar enough to this new problem, the solution attached to the recorded case is retrieved. Depending on the particular CBR implementation's option, this solution can be applied immediately, or revised to better fit the problem at hand. After applying this solution, it's success can be monitored. If the system considers that the new problem, together with the applied solution, shows enough novelty with regard to the existing cases, it may choose to insert it in the Case Base as a new case. If this is done, the new case will be available for choice to solve future problems.

From this empirical point of view, a CBR system will perform four different tasks, as depicted on Fig. 4.5, when considering a new problem:

1. **Retrieve:** search the Case Base for the most similar case to the new problem at hand;

2. **Reuse:** combine the retrieved case with the problem at hand into a proposed solution to the problem;

3. **Revise:** test or evaluate the proposed solution, and possibly change it, if it is unsuccessful;

4. **Retain:** if the new problem and solution are considered interesting, store them in the Case Base.

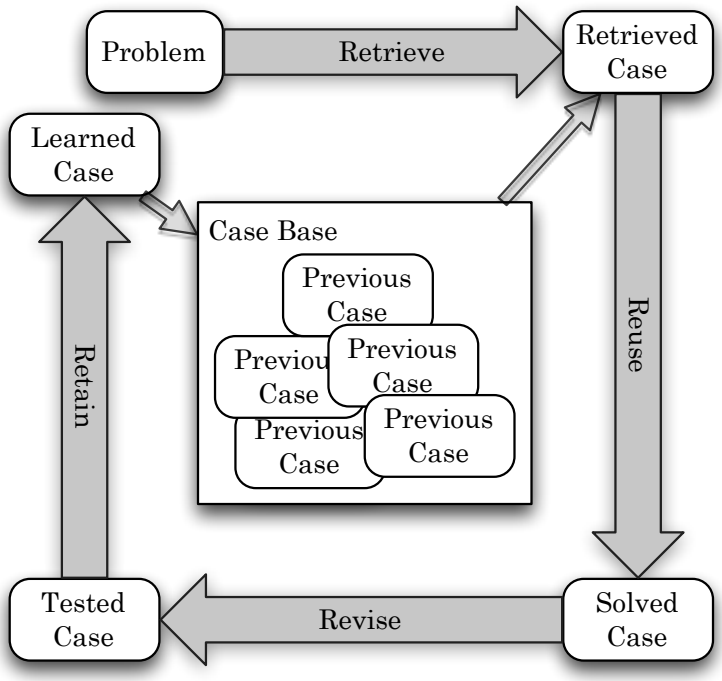


Figure 4.5: CBR cycle, adapted from Aamodt and Plaza (1994)

This mechanism must, thus, record the performance and application conditions of each execution of every Setplay, in order to collect data to evaluate their performance and, consequently, the usefulness of each Setplay in similar conditions. This data should be examined and consolidated (e.g. by abstraction or generalization) in order to better estimate the performance of a Setplay in common situations.

This general CBR architecture has to be adapted to the RoboCup domain in order to be fully functional and competitive. Two of the previously cited tasks must be adapted: case retrieval is based on resemblance between recorded cases and the new problem, and thus needs a measure of similarity, while case revision would need some kind of adaptation algorithm for Setplays to be reused outside their original scope. These issues will be looked into in the following sections.

4.2.1 Case characterization

Case similarity can be estimated upon comparison of the different characteristics that describe each case. In the RoboCup domain, a CBR system for Setplay choice will consider cases as particular executions of a Setplay. These executions must be properly characterized and described, as these characteristics will be the information on which case comparison and selection will be based. This choice of characteristics must be a well balanced

one: it must include the relevant information, but should not be too thorough, since this would increase the amount of information to be analyzed, making the comparison process more time consuming.

In the present approach, the choice was directed towards simplicity: the Setplay details stored in each case are simply the spatial localization of the case, the opponent and the Setplay's success. The most delicate part in this characterization is probably the localization: on the one hand, a Setplay takes place in an extended area, which is difficult to describe, and, on the other hand, minor spatial differences should be disregarded, in order to attain an acceptable level of generalization. With the goal of keeping the representation simple, case localization is made through the Setplay's starting point: since Setplay execution does not typically differ much between executions, the point where the Setplay is started can be seen as a very simple, yet significant, spatial reference. Concerning spatial abstraction, one opted for grouping nearby Setplays: instead of storing the Setplay's starting-point, the representation just records this point's inclusion in three different partitions of the field. This was done by simply marking five intermediary, equally spaced points on the field's big edge, and doing the same on the small edge. By linking these points transversally and longitudinally, two partitions were made, seen in figures 4.6 and 4.7.

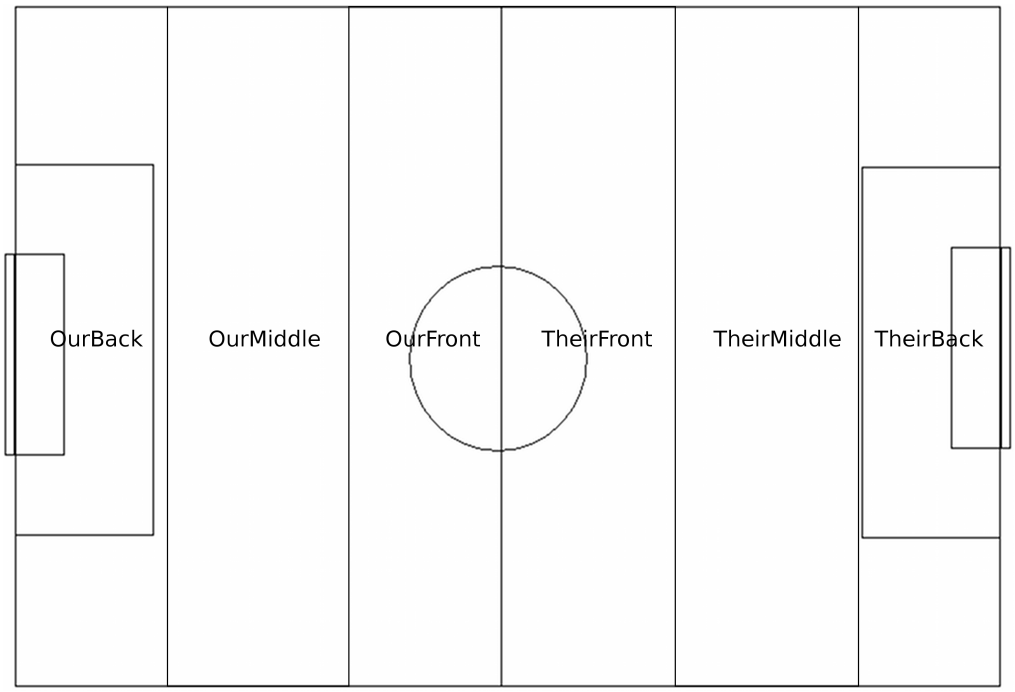


Figure 4.6: Transversal partition of the pitch

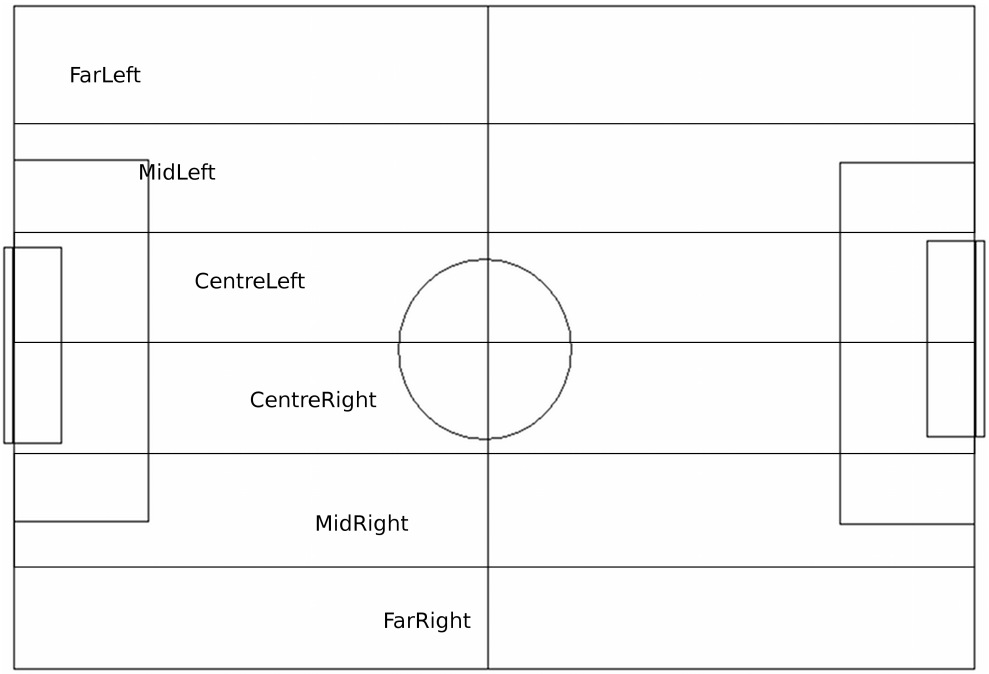


Figure 4.7: Longitudinal partition of the pitch

To better characterize the location with respect to the opponent's goal, another partition was designed, by connecting the points on each of the long edges and the own short edge with the opponent goal centre. This partition, referred to as *radial*, can be seen in Fig. 4.8. Since every point will be considered as belonging to an area in each of these partitions, one can consider that the field will be divided in quite small areas, as depicted on Fig. 4.9. In all these figures, the penalty areas are depicted only to aid the reader's understanding: they play no role in the partitions. One should also note that the localization of Setplays is only done when this is relevant: when the game mode is a kick-off or a goalie catch, the location is fixed, in the former case, and freely choosable in the latter, in the case of the Simulation 2D league. Thus, in these cases, the location of the Setplay is not decisive.

This option for Setplay localization results in case aggregation: all executions of a Setplay initiated in (usually distinct) points of one of the areas in Fig. 4.9 will be aggregated in the same case. These executions will, though, probably have different outcomes: some are successful while others fail. Therefore, each case aggregation will contain information about the total number of Setplay executions, and the fraction, among these, when the Setplay was successful.

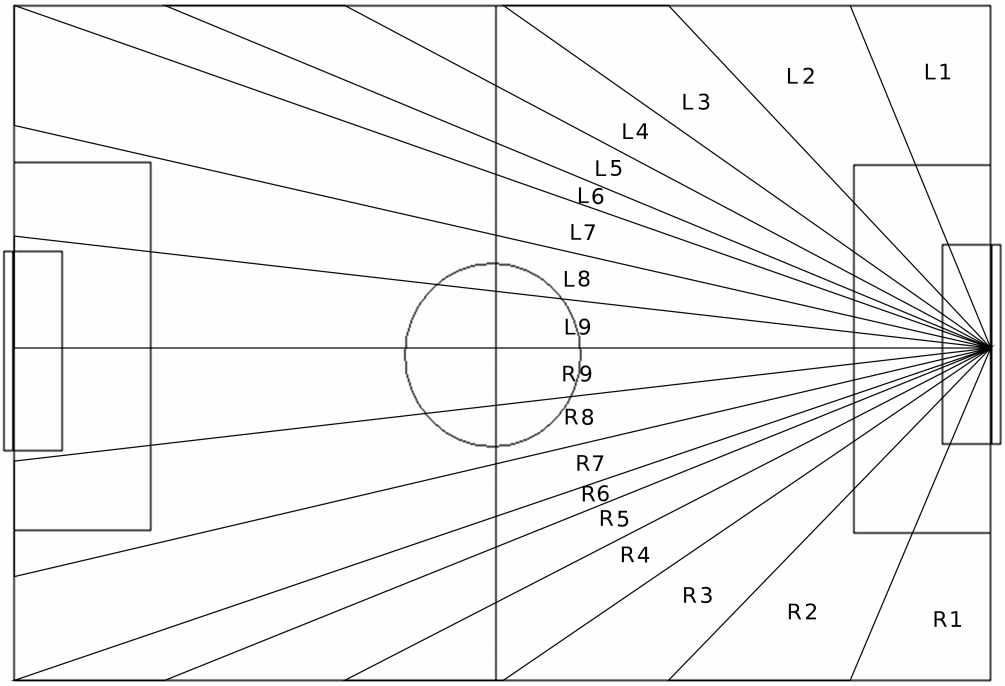


Figure 4.8: Radial partition of the pitch

4.2.2 Case spatial similarity

When analyzing a new problem, the CBR system needs to compute its similarity with each of the cases in the Case Base. This similarity will determine each case's relevance to the choice.

The first issue in this computation is spatial proximity: the closer a past case is to the current problem, the more it should be considered. As seen in the previous section, case localization is stored, and thus aggregated, as membership of areas that partition the field. With such a discrete representation, it is not viable and does not make sense to use a metric distance.

Since, in this situation, one is dealing with adjacent areas, resulting from partitions, it was chosen to use a discrete distance between the areas: one area will of course have a zero distance from itself, while adjacent areas, like '*OurBack*' and '*OurMiddle*' in Fig. 4.6, will have a distance of 1. Since this kind of area distance is empirically expected to grow rapidly as separation increases, '*OurBack*' and '*OurFront*' will have a distance of 4, which corresponds to the square of the simple separation between the areas, i.e., 2^2, and so on for the other areas. The formula for transversal distances between areas a_1 and $_2$ is thus $dist_{trans}(a_1,a_2) = numberOfFrontiersBetween(a_1,a_2)^2$. For the longitudinal partition, a similar calculation is used and thus, for instance, the distance between '*MidLeft*' and '*MidRight*' will amount to 9, i.e. 3^2. The radial partition of the field

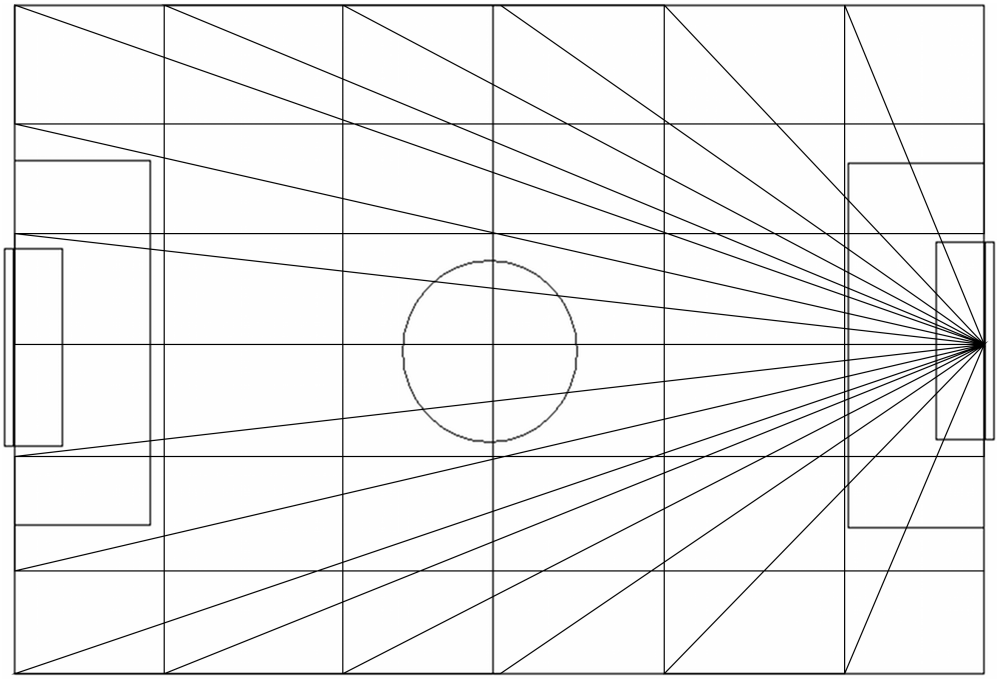


Figure 4.9: Global partition of the pitch

contains much more areas, which led the distance to be calculated with a more subtler increase with separation. In this case, the separation is only raised to 1.5, according to the formula $dist_{rad}(a_1, a_2) = numberOfFrontiersBetween(a_1, a_2)^{1.5}$. As such, one will for instance have a distance of 5.2 between '*L7*' and '*R9*', i.e. $3^{1.5}$. With this kind of calculation, distances according to the longitudinal and transversal partitions are limited to a maximum value of 25 (5^2), while the radial partition never exceeds 70.1 ($17^{1.5}$).

Naturally, these three topological distances must be combined. If this was to be done through a sum, the result would be inversely proportional to the similarity between locations: the bigger the distance, the less the locations are similar. One must, thus, consider a way to rework this result such that the result ranges between zero and one, where zero represents maximum difference, and one a high similarity. The formula employed is as follows, with abbreviations such as $a_{trans}(p_{problem})$ meaning the transversal area where the problem's location is located, and so on: $simil(p_{problem}, p_{case}) = 0.5^{dist_{trans}(a_{trans}(p_{problem}),a_{trans}(p_{case}))+dist_{long}(a_{long}(p_{problem}),a_{long}(p_{case}))+dist_{rad}(a_{rad}(p_{problem}),a_{rad}(p_{case})))}$

Departing from the ball's location, which describes the current problem, the algorithm to combine the three distance values and to compute the similarity to one particular case, will work as follows, using the same kind of abbreviations:

With this algorithm, a case located in the same partitions as the problem will have a spatial similarity of $0.5^{0^2+0^2+0^{1.5}} = 1$. When, on the other hand, one tries to compare a case located in '*OurBack*', '*FarRight*' and '*R5*' with a very distant problem, lo-

Algorithm 4 Compute the spatial similarity between a point p and a case c

$dist_{trans} \leftarrow dist_{trans}(a_{trans}(p), a_{trans}(c))$
$dist_{long} \leftarrow dist_{long}(a_{long}(p), a_{long}(c))$
$dist_{rad} \leftarrow dist_{rad}(a_{rad}(p), a_{rad}(c))$
return $0.5^{dist_{trans}+dist_{long}+dist_{rad}}$

cated in '*TheirBack*', '*FarLeft*' and '*L1*', the similarity would amount to $0.5^{5^2+5^2+13^{1.5}} \cong 0.5^{96.87} \cong 6.91 * 10^{-30} \cong 0$. Such spatial similarity values are in line with what was desired.

4.2.3 Case retrieval

In this implementation's context, case selection is not a trivial task: since the localization of a case plays a very important role in case storage, it is very common for the same Setplay to have been executed in different locations. As an example, the same kick-in might have been executed at different points along the touch-line, with different levels of success. It would be too simplistic to use a nearest neighbor approach and select only the nearest case: this would neglect information in other nearby cases. Thus, the different cases pertaining to the same Setplay can be used to estimate that Setplay's success, with different weights corresponding to their characteristics in common with the current problem. These weights depend on the spatial similarity level described in the previous section, but also on some other configurable factors, which will be described hereafter.

The case base contains, as seen in section 4.2.1, the registry of different Setplays' executions, against the current and other teams, both in the present and past games. Since these execution details differ considerably, one might want them to consider different evaluation weights in the CBR choice: typically, executions against this same team will have a higher weight than executions against other opponents, while executions done previously in this same game will have a still higher weight. The relative importance given to each of these situations might be different between teams or leagues: one team might choose to give a very high weight to the cases in the current game, while other might want to have the same weight to all situations. Thus, these three weights (of executions in this game, in previous games against the same opponent, and in previous games against other opponents) are configurable as arguments to the CBR algorithm.

In still different situations, a Setplay might have no record of previous executions in the Case Base. In such a situation, that Setplay would never be chosen in a pure CBR setting, which relies solely on past experience. To ensure that such a Setplay might nevertheless be chosen, the system is also prepared to receive, by convention, a default evaluation value for Setplays with no previous record of execution. This value should

be low (e.g. corresponding to a Setplay with almost all executions failed), to avoid the Setplay to be chosen with high probability. Naturally, this default value is only used in the Setplay's first execution.

This retrieval method is better described through a practical example: one will consider a new problem in a game against a team called *opp1*, which consists of a kick-in in a point placed in the three following areas: '*OurFront*', '*FarLeft*' and '*L4*'. When analyzing this problem, the system will only consider Setplays that are possible at that moment, i.e., that have their pre-condition satisfied. Therefore, only Setplays that are possible in the kick-in game mode would be possible. Considering only three particular Setplays, called *kick-in1*, *kick-in2* and *kick-in3*, one might have the following cases:

1. *kick-in1* placed in '*OurFront*', '*FarLeft*' and '*L4*' against *opp1* in the current game, one trial with one success;

2. *kick-in1* placed in '*TheirFront*', '*FarLeft*' and '*L3*' against *opp1* in a past game, two trials with one success;

3. *kick-in2* placed in '*OurBack*', '*FarLeft*' and '*L6*' against *opp2* in a past game, three trials with one success;

4. no record for the execution of *kick-in3*.

The first step for the evaluation of these cases will be the computation of the positions' similarity: each case's positioning has to be compared with the current problem's position. As an example, let us look at case 2: the longitudinal area is the same, but there are differences of 1 in the transversal and radial partitions. As a reminder, the formula for calculating the spatial similarity, presented is section 4.2.2, is $simil(p_{problem}, p_{case}) = 0.5^{dist_{trans}(a_{trans}(p_{problem}),a_{trans}(p_{case}))+dist_{long}(a_{long}(p_{problem}),a_{long}(p_{case}))+dist_{rad}(a_{rad}(p_{problem}),a_{rad}(p_{case}))}$. Therefore, the topological similarity will be given by the expression $s_2 = 0.5^{0^2+1^2+1^{1.5}} = 0.5^2 = 0.25$. Using the same algorithm, one will have $s_1 = 0.5^{0^2+0^2+0^{1.5}} = 0.5^0 = 1$ and $s_3 = 0.5^{0^2+2^2+2^{1.5}} = 0.5^{6.828} \cong 0.009$.

To put forth these cases' evaluation, one has to consider the weights for executions in this same game, in past games against this team and in past games against other teams. These will be, in this example, respectively, $w_g = 4$, $w_t = 2$ and $w_o = 1$. With these values, one can proceed to the final evaluation of the cases, using the formula $e = s * w * (\frac{num_{tries}}{num_{fails}} - 0.8)$, where s stands for the topological similarity, w for the opponent specific weight, num_{tries} for the number of times the Setplay has been executed, and num_{fails} for the number of executions that were unsuccessful. The subtraction of 0.8 was done to privilege more successful Setplays: a Setplay with no successful executions will have a

rating of 0.2 to be multiplied by the spatial and opponent specific weights, while a Setplay with only one failed execution in three will multiply the weights by 2.2. If one applied this formula to the data in case 1, it would yield the following value: $e_1 = s_1 * w_g * (\frac{num_{tries}}{num_{fails}} - 0.8) = 1 * 4 * (\frac{1}{0} - 0.8)$, which is impossible to calculate, due to the division by zero. In these cases, one has to apply one exception and consider a fake value for the number of fails. This value was set to 0.5, with the goal of having an increasing evaluation as the number of successful executions grows. In this case, the corrected value will thus be $e_1 = s_1 * w_g * (\frac{num_{tries}}{num_{fails}} - 0.8) = 1 * 4 * (\frac{1}{0.5} - 0.8) = 4.8$. For the remaining cases, the general formula will be used: $e_2 = s_2 * w_t * (\frac{num_{tries}}{num_{fails}} - 0.8) = 0.25 * 2 * (\frac{2}{1} - 0.8) = 0.6$ and $e_3 = s_3 * w_o * (\frac{num_{tries}}{num_{fails}} - 0.8) = 0.009 * 1 * (\frac{3}{2} - 0.8) \cong 0.006$. As for the *kick-in3* Setplay, there are no recorded cases, and it will be awarded a default value (0.4, by convention, corresponding to the double of a Setplay with all fails), which will be multiplied by the weight relative to this game, and will be later referred to as $e_4 = 0.4 * w_g = 1.6$, in order to make its choice possible.

4.2.4 Case selection and reuse

With all active cases evaluated, the moment to select the Setplay to be executed is reached. A straightforward strategy would be to simply choose the case with the highest evaluation. Such a strategy would, though, neglect other cases, which might have a low evaluation due to past executions with other opponents, but that might be competitive due to, for instance, good adaptation to the present opponent. To overcome this issue, another strategy was setup, which privileges cases with higher evaluation, while still leaving a chance for other cases to be selected.

Returning to the example from the previous section, one had come to four different case evaluations, with individual evaluations $e_1 = 4.8$, $e_2 = 0.6$, $e_3 = 0.006$ and $e_4 = 1.6$. The sum of these values is 7.006. In this step of the CBR algorithm, a random value will be generated, with possible values between zero and the sum of evaluations. To select a Setplay, the individual evaluations will be accumulated, in order of case retrieval. When the accumulator equals or exceeds the random value, the Setplay under consideration is chosen. In the present example, a random value of 4 would output the Setplay related to case 1, while 5 would select the Setplay of case 2, 5.403 the one related to case 3 and, finally, a random value of 6 would select the Setplay of case 4. This algorithm will allow any Setplay with a positive evaluation to be selected, while keeping the selection probability proportional to the evaluation. This was the behavior desired for the system. To better illustrate the logic behind this choice, one can refer to Alg. 5.

Algorithm 5 Select a Setplay from a list *listEval* of their evaluation values

Require: *listEval* is not empty
  sumEval ← sum of all values in *listEval*
  rand ← random value in [0, *sumEval*[
  runningSumEval ← 0
  for $i \leftarrow 0$ **to** size of *listEval* **do**
    $runningSumEval \leftarrow runningSumEval + listEval[i]$
    if $runningSumEval \geq rand$ **then**
      return i
    end if
  end for

Setplays can primarily use two different kinds of spatial coordinates: absolute, and relative to the ball or a player. If the second kind is used in a Setplay definition, then the Setplay can be displaced freely around the field, accompanying the ball's position. The Setplay definition should always define the field areas where the Setplay is usable, both when the CBR system is used and when not, to avoid invalid positions, e.g with players outside the field.

When using Setplay selection by CBR, the user/developer should therefore pay attention to this issue: as Setplays might be executed in different locations, all spatial references should be relative. Such Setplays need no adaptation to new settings. Using this kind of Setplays only, one avoids any kind of adaptation in case reuse.

4.2.5 Case revision

Revision is in this domain practical: the selected Setplay is tested by execution in game situations. Its success is determined by wether the execution reached the desired finish, or, on the contrary, if it was aborted. Success has thus a boolean value.

Since Setplays are very complicated combinations of players positions and actions, and related timings, if was deemed difficult to design a system to automatically fine-tune or evolve a Setplay. Therefore, when the Setplay is unsuccessful, it is not altered.

4.2.6 Case retention

Since the cases are stored in an aggregated manner, the storage of an individual case does not typically change the case base considerably: if the region it was executed was already present in the case base, it simply demands a change in the number of executions and successes; if that area is new, one entry is added to the case base, which can later store information about other executions of the same Setplay.

Under these circumstances, it was decided to always retain new cases, since the results of execution in the present game are the ones that present the most valuable information for decision. Thus, every execution's result is retained in the case base.

If the user/developer decides to do so, he may configure the CBR system to write the case base to file, in order for the results of the game to be considered in future Setplay selections by the CBR system.

4.3 Graphical Design of Setplays: SPlanner

In an effort parallel to this thesis and based on the Setplay Framework, a graphical tool (*SPlanner*) was developed to design Setplays. A tool for the graphical design of Setplays can act as a catalyst for Setplay creation and dissemination. Such a tool is essential for usage by non-specialists, e.g. real soccer coaches, who will not be able to edit Setplays in their textual format. The tool as a whole is described in Cravo (2011), but the Setplay design procedure will be looked into hereafter.

4.3.1 Other strategy tools

Some tools have been developed with the goal of assisting the definition of strategy in human soccer, like Coach-Helper[3], ForCoach Tactics Viewer and ForCoach of Soccer[4]. These tools are commercial and their functionality is focused on soccer tactical panels for the definition of team formations. These tools do not provide any support for the definition of Setplays.

The video-game industry has grown considerably over the last decades. There are many soccer games, like FIFA Soccer 2011 [5] or Championship Manager 2010[6], available in different platforms,which allow the definition of some strategies.

Online games are quite simple, offering few tools for strategy management. Soccer simulators offer a larger set of tools in order to allow the adjustment of different playing styles. FIFA Soccer 2011 offers tools to define team strategy during a game, with different levels of detail. It allows the definition of the players' positions in the field and mentality. Team management games present richer functionalities to define strategies. Championship Manager 2010 was the only tool providing the possibility to build setplays and thus inspired SPlanner's design.

[3]URL: `http://www.coach-helper.com`
[4]URL: `http://www.forcoach.com`
[5]URL: `http://www.ea.com/soccer/fifa`
[6]URL: `http://www.championshipmanager.co.uk/`

Several tools have also been developed by the robotic soccer community. Playmaker (Lopes et al., 2010) was a previous effort, inside the FCPortugal team, for the graphical definition of Setplays. The provided interface was not adequate for use by non-expert users, which led to the decision of building a new, more user-friendly tool, inspired in applications for real soccer.

4.3.2 General Architecture

SPlanner was developed in C++, using the Qt graphical libraries (Nokia, 2011). A general overview of the SPlanner tool architecture is presented in Fig. 4.10.

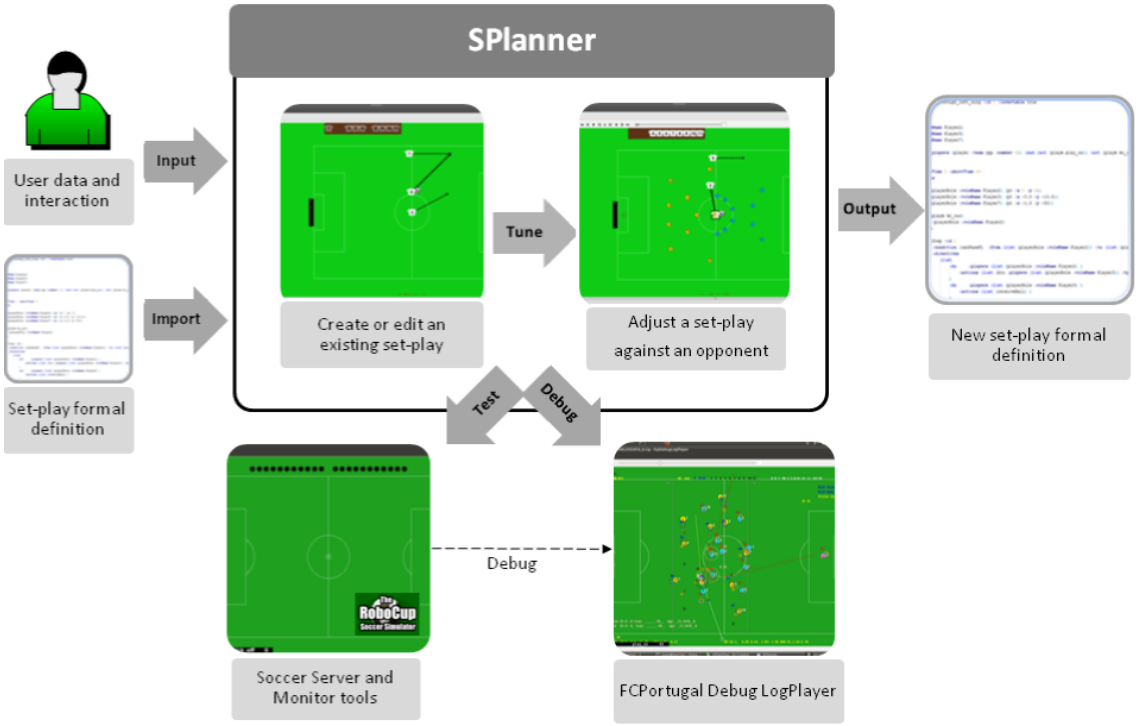


Figure 4.10: SPlanner tool architecture overview

The development of the tool was done with modularity in mind, in order to easily allow the future integration of new strategy modules (e.g. formation editor, tactics manager). Three different interfaces were defined to separate the import and export functionality from other plan-based frameworks, external programs and game log viewer, to analyze all situations in past games.

This application includes shortcuts that allow the testing and debugging of a Setplay, using the Soccer Server and Monitor tools (Noda et al., 1998), as well as the FC Portugal Debug LogPlayer (Lau and Reis, 2007).

4.3.3 Interface design

The interaction with the main interface of SPlanner, presented in Fig. 4.11, is done primarily using the mouse and keyboard.

Figure 4.11: SPlanner main GUI for the definition of a Setplay

This interface is organized in 3 main areas: the menu bar, the Setplays separators (for editing more than one Setplay at once) and the Setplay workspace. Possible actions are to create, import and export a Setplay. In order to swiftly execute tests, one can start the execution of a Setplay, or debug a previously executed Setplay through the Soccer Server and Monitor tools and the FCPortugal Log Player Debugger.

The central area of the Setplay workspace is essentially an image for the soccer pitch, where the players participating in the Setplay will be positioned and their respective actions defined, and a substitution bench, which will contain the players not participating in the Setplay. In the left panel, one can see general information that describes the Setplay and its abort conditions (step-independent), the flow of executions of a Setplay mapped into a graph (further explained in Section 4.3.4) and the abort and wait times for the currently chosen step. In the bottom area of this workspace a collapsible panel was created to provide additional relevant information regarding the current active player.

A wizard was designed to assist the user in the creation of Setplays in a more intuitive fashion. Based on the experience gathered from the analysis of similar software tools and after studying the typical task flow for the creation of a Setplay the following steps were defined:

1. Define the type of Setplay (offensive or defensive) to create;
2. Choose the game state (e.g. free-kick) where the Setplay is possible;
3. Select the location in the field (e.g. opponent front wing) that combined with the game situation will trigger the Setplay. It should be noted that some positions will be omitted based on the chosen game situation (e.g. kick-off) as they can only start from one specific location.

The location on the field chosen in the third step of the previous procedure will be visually highlighted with a textured shape for the user to know where the Setplay will be possible. If a region on the field is chosen, the tool will constrain the ball position and consequently the leader, which is assumed to own the ball in every step. Although the Setplay grammar allows a leader to be an arbitrary player participating in a step, in the majority of cases, the ball owner will lead the Setplay.

The players that will be participating in the Setplay must all be added in its first step. Afterwards, the user can only define new positions for Setplay in subsequent steps by means of assigning actions to them in the transitions that lead to those steps. By estimating the effects of the actions specified for players using an estimation model based on the ones applied by Soccer Server to the players and the ball, the tool estimates the players and ball future positions in subsequent steps of the Setplay, and they cannot be changed.

In order to assign a player to the set of participants of a Setplay, the user just needs to drag it from the team's bench onto the field area. It is also possible to add a player not participating in a step (it will have a jersey with transparency in Fig. 4.12) as a participant, if he has already been added to the list of Setplay participants in the first step. This is done automatically through the allocation of an action to this player. The execution of the inverse action, dragging a player to his team's bench or simply out of the field, removes him from the list of the step participants. This metaphor will appear natural for users with some knowledge of the domain as a strong connection exists with the reality in which players that are sitting on the substitution bench are inactive and thus not participating in the running game, contrarily to the players that are positioned on the field.

Players are represented by a numbered and colored jersey that provides information to about their status (see Fig. 4.12) and can be selected or dragged.

Figure 4.12: Types of players jerseys in the SPlanner GUI

A right mouse-click on a step participant pops up a context menu, as depicted in Figure 4.13, where the user can specify its actions and initial position, or simply remove

it from the step. In this context menu, only feasible actions will be presented to the user (e.g. a pass action will only be present if the player owns the ball). Furthermore, for some specific feasible action, there might be only small set of relevant options to choose from and these are thus filtered for the user. For example, in a step scene where 3 players A, B and C participate, a pass action chosen for A will have as possible destination only players B and C.

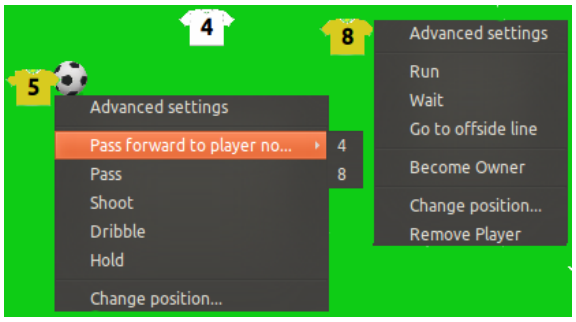


Figure 4.13: Menu with feasible actions for the active player

4.3.4 Execution flows of a Setplay

In order to provide a general comprehension of the defined Setplays execution flows a graph visualization was designed. This graph contains a representation of identifiable steps (numbered circles) and the defined transitions between them (directed arrows) in the Setplay, allowing the user to understand all its possible execution flows as depicted in Fig. 4.14. There can only be one start step, but an arbitrary number of intermediate and final steps.

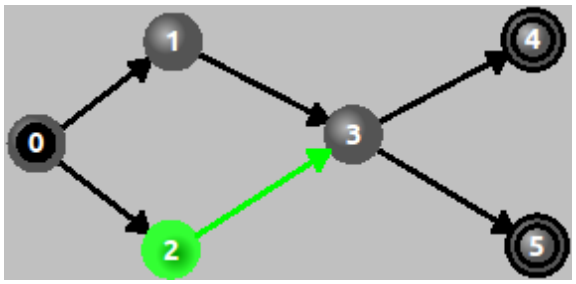


Figure 4.14: Graph of a Setplay with all possible execution flows

To facilitate the distinction between different types of steps (start, intermediate or final) three different colors were applied. The start step is filled black and has the number 0, while the other steps are filled gray. Moreover, the final step is distinguished through a double black contour around it. The rendering of the graph is done automatically in real-time as the Setplay definition is updated.

4.3.5 Defining actions for participants

SPlanner has currently built-in support for 8 actions, for which different icons were defined (see Fig. 4.15), in order to make them clearly distinguishable for a user analyzing a Setplay.

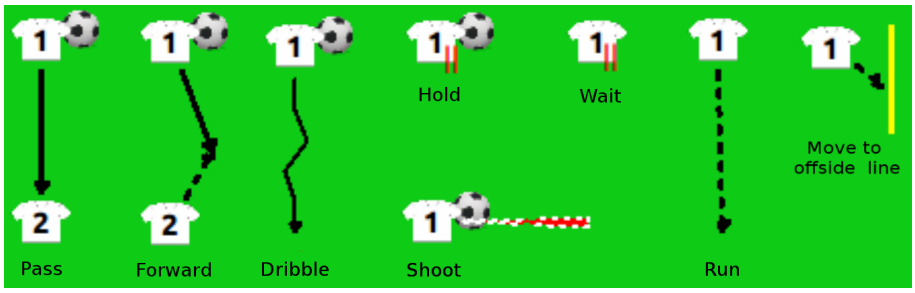


Figure 4.15: Icons for player actions

Among these actions, five require the step participant to own the ball at the time of its execution (direct pass, forward pass, dribble, hold the ball and shoot), contrarily to the remaining three (wait, run and position near the opponent offside line). The actions that are presented to the user when he selects a participant are filtered based on this criteria to prevent semantical errors (e.g. defining a dribble action for a participant that does not own the ball).

4.3.6 Positions of players and action targets

The positions of the Setplay participants can be undefined, absolute field coordinates or relative to the ball or another participant. Each position type is distinguished visually in the Setplay representation using the icons shown in Fig. 4.16.

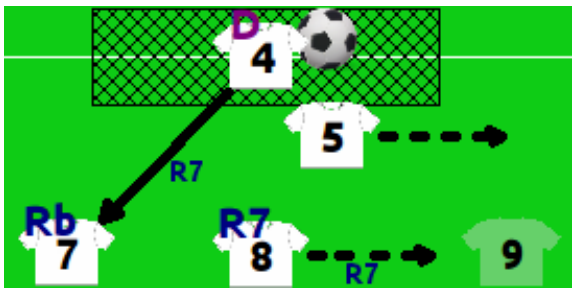


Figure 4.16: Types of positions (relative, absolute and undefined) for players and action targets

In Fig. 4.16, player nr. 4 has an undefined position (although in this case it will be within the marked region), player nr. 7 is positioned at (-10,-10) relative to the ball (which

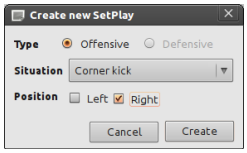
in this case is supposed to be held by player nr. 4), player nr.5 is absolutely positioned at (-23,30) and player nr. 8 is positioned at (10,0) relative to nr. 7, and also performs a run to position (18,0) relative to player nr. 7. Player nr. 9 is shown with an opacity because it participated in the previous step, but in the current step it has no defined action and thus its position remains the same.

The letter *R* is placed on the top left corner of a player's jersey and on the middle of the iconography of an action to represent an initial relative position towards a player and for the intended action respectively. This letter can be followed by a number identifying the player to whom that player is relatively positioned (e.g. *R*2) or the letter *B* if he is positioned relatively to the ball. When the letter *D* is placed on the top left corner of a player's jersey it represents an undefined position for that player.

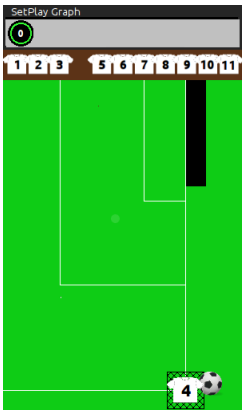
In order to demonstrate the simplicity in defining a Setplay, a step-by-step example of the process for defining a corner-kick Setplay with 3 participants is described in Fig. 4.17, and its manual definition is presented in Fig. A.1 in the Appendix.

4.4 Summary

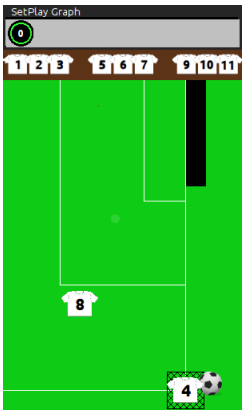
This chapter described the implementation, as a C++ library, of the classes supporting the Framework, as well as the auxiliary tools: Setplay parser, mechanisms for choosing participating players, execution engine, CBR-based Setplay selection algorithm and graphical designer. With these tools at hand, a user/developer can quickly proceed to applying the Framework to a team. Two such efforts, in the Simulation and Middle-size leagues, are described in the next chapter.



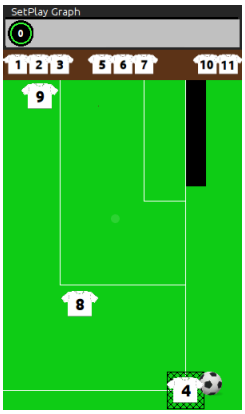
(a) Choose the type, situation and start position



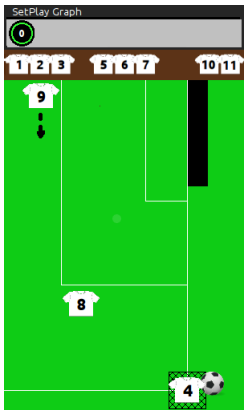
(b) Pl. 4 is placed automatically in the corner



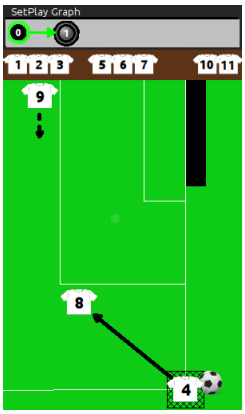
(c) Pl. 8 is dragged from the bench to its initial absolute position



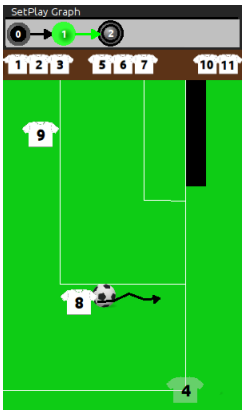
(d) Pl. 9 is dragged from the bench to its initial absolute position



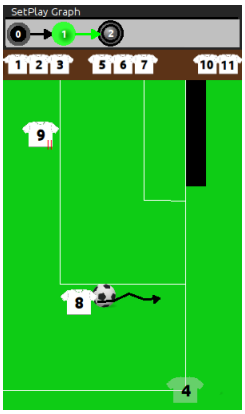
(e) Pl. 9 runs to the entry of the penalty area



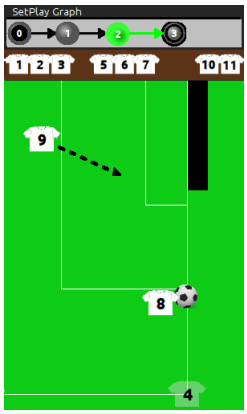
(f) Pl. 4 passes to pl. 8



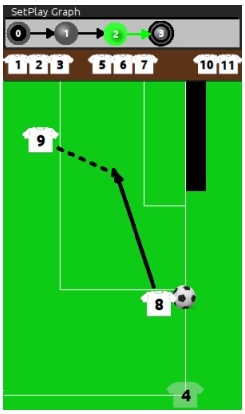
(g) Pl. 8 dribbles towards the goal line



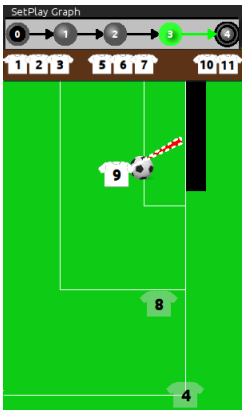
(h) Pl. 9 waits for pl. 8 to complete his dribble



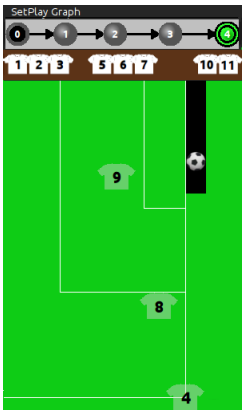
(i) Pl. 9 runs to the border of the goalie area



(j) Pl. 8 performs a forward pass to pl. 9



(k) Pl. 9 shoots at goal



(l) Player nr. 9 ultimately scored a goal

Figure 4.17: Step-by-step definition of a corner-kick Setplay with three participants

Chapter 5

Framework Application to RoboCup Teams

5.1 Introduction

The Setplay Framework provides, as described in the previous chapters, the main components for Setplay execution: parser, execution engine, Setplay selection, and other auxiliary tools. While applying the Framework to specific teams and leagues, there is, though, the need to ground the Framework on league-specific concepts. In the Simulation 2D league, passes and kicks can only be done at ground level, while in the Middle-size league, some teams are capable of executing chip-kicks, and other teams are not. This means that the same abstract action, in this example a kick, will be performed very differently in different teams and in different leagues. This implies that each application of the framework is team and league specific.

In order to be able to deploy the Framework to its full extent, a team will have to accomplish the following tasks, which can be considered a check-list before implementation (see section 3.3):

Implementation of Actions: The team must be able to execute the abstract skills defined as *Actions* in the Framework, like kicks, passes to regions and shots, as well as movements to points, marking of opponents and regions, and ball reception and interception.

Implementation of Conditions: Concepts like ball possession and evaluation of pass and shot success likelihood must be evaluated by the team. Such concepts correspond to *Conditions* in the Framework model.

Setplay Selection for Execution: The team must either decide autonomously what Setplays to execute, and when, or let the Framework select the Setplays it considers more promising at a given moment, as described in section 4.2 .

Action Choice and execution: At each execution cycle, the team must check for actions needing to be executed, as determined by the Framework, and select one to execute, when there is more than one available. When no such action exists, each player may opt to proceed to other tasks, like positioning itself.

Inter-robot Communication: The lead player, that can be a fixed player like the coach, or change along *Steps* according to Setplay definition, must be able to send simple text messages to the other players. This communication can be uni-directional.

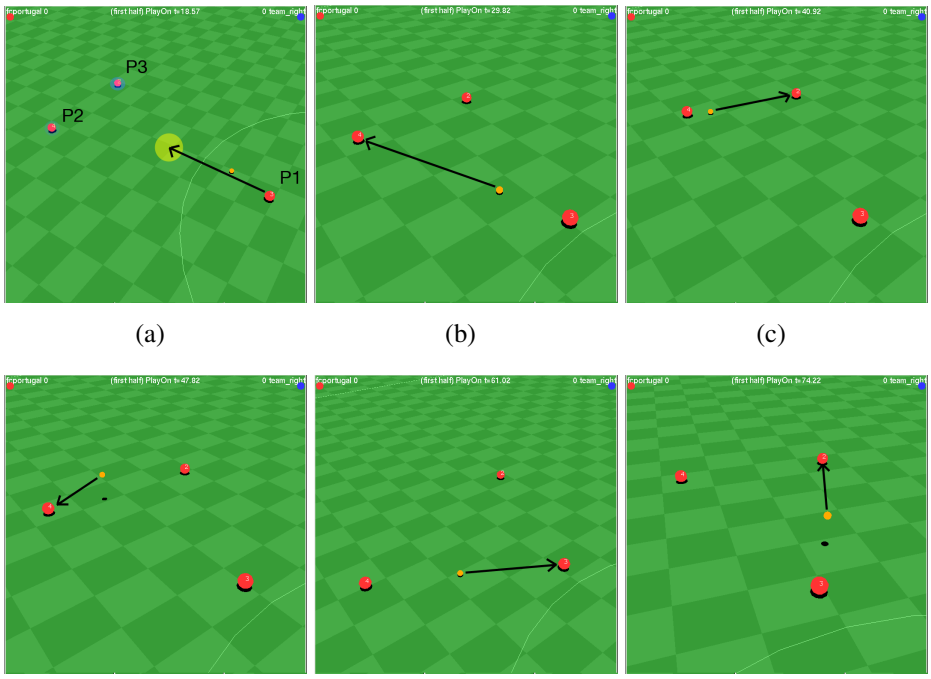
The Framework has been applied to three different teams as part of the development that supports this thesis. This application to teams required the fulfillment of the tasks that have just been described. The next sections describe how these tasks were achieved in the selected teams.

5.2 Application to the 3D Simulation League

As an initial testbed for the *Setplays*, the code of the 2006 champion in the 3D simulation league, FC Portugal (Lau and Reis, 2006), was used. This code already had the main building blocks for the application of *Setplays*: *Conditions* and *Actions* were implemented based on the existing code dealing with world-state, skills and actions.

The Framework was applied to a functional state: some Setplays were tested, namely one with players passing the ball among themselves. Some screenshots of the execution of such a Setplay, in this case with three players arranged on a triangle, can be seen in Fig. 5.1.

Setplays were tested experimentally and ran smoothly, with passes being done with success. At this development phase, the focus was on Setplay execution: one examined the execution of different Setplays and looked any for unexpected behavior, correcting the problems initially present on the Framework. There was, at the time, still no work done on Setplay evaluation and selection. Also, the Framework's model only contained limited sets of *Actions* and *Conditions*: only later, while applying the Framework in competitions, did one feel the need to define more complicated Setplays. The results reached at this phase were published in Mota and Reis (2007a).



(a) (b) (c)

(d) (e) (f)

Figure 5.1: Setplay example: 3D Simulation League (Sphere model)

The rules and organization for the 3D competition were largely changed from RoboCup 2007 on: the robots changed from the sphere model to humanoids, and each team was initially composed of only two players. Since the new simulated robots initially demanded research on low-level questions like robot movement and ball kicking (and, from a high-level point of view, still do), there was temporarily no place for high-level concepts like Setplays. The development in this league was thus suspended and can only be picked up again when these elementary skills are better developed, which will happen in the near future: as soon as teams reach a development state where the game flow is continuous and allows inter-player skills like ball passing, the Setplay Framework can again be applied to this league. Before such a development state is reached, Setplays will not be able to play a significant role in the strategy, which makes its use inappropriate. Due to this change, the development of the Setplay Framework had to be put forth in other leagues, presented hereafter, where the basic skills and moves are more advanced, and hence game pace is faster. Such is the appropriate application scenario for Setplays.

5.3 Application to the 2D Simulation League

The 2D simulation league is the most developed league in RoboCup, in terms of game flow and high-level cooperation. FCPortugal has been participating in this league since 2000, with very competitive results, including several titles, namely champions at world and european tournaments.

The FCPortugal code has several features available: a mature State-of-the-World, which considers both the player's own observations and information shared by other players, and which includes prediction of action and interaction effects; and a set of actions and skills that allows the mapping of abstract actions as defined in the *Setplay* framework to concrete executions in the 2D simulator. The application of the Setplay Framework to this team was presented in Mota et al. (2010b).

This application was achieved after following the steps mentioned in section 5.1, which are described in the next sections.

5.3.1 Implementation of Actions

The Simulation 2D server only accepts very low level actions, like dash (accelerate), turn, kick (in a certain direction, with a certain power) and tackle. To support the development of a player in such an environment, it is easier to think in terms of more abstract actions, like going to a point, passing to a teammate and shooting at goal. Such abstract actions generally have to be defined as a sequence or combination of several of the more low-level server-specific actions.

Since the FCPortugal team has a story that dates back to 2000, and also due to the changes in the Simulation 2D rules, the original set of high-level, abstract actions has evolved, in a way that led to the existence of several versions of some of these skills, which may be used alternatively. Therefore, the array of available actions is numerous.

The *Actions* in the Setplay model are, though, in a still higher level of abstraction, since sometimes they will demand some kind of preparation phase, before executing the actual, nominal action. Namely, in order, e.g., to pass the ball to a teammate, a player will need to get possession of the ball, position itself adequately, and finally kick the ball towards the teammate. This procedure is depicted in Fig. 5.2. The execution of the action only ends, as depicted, when the player has definitively kicked the ball, since this action might be done as a multi-kick, i.e., applying sequential kicks in different execution ticks. The implementation of this particular action was quite straightforward, since the Action definition considers several execution options, namely the pass type, which in this case is used to characterize the passing speed. In other application scenarios, e.g. in other

leagues, there could possibly be other values for this pass type, like plain or chip kicks in the Middle-size league. In any case, the pass action execution merely has to get hold of the ball, position the player appropriately and perform the kick according to the pass type.

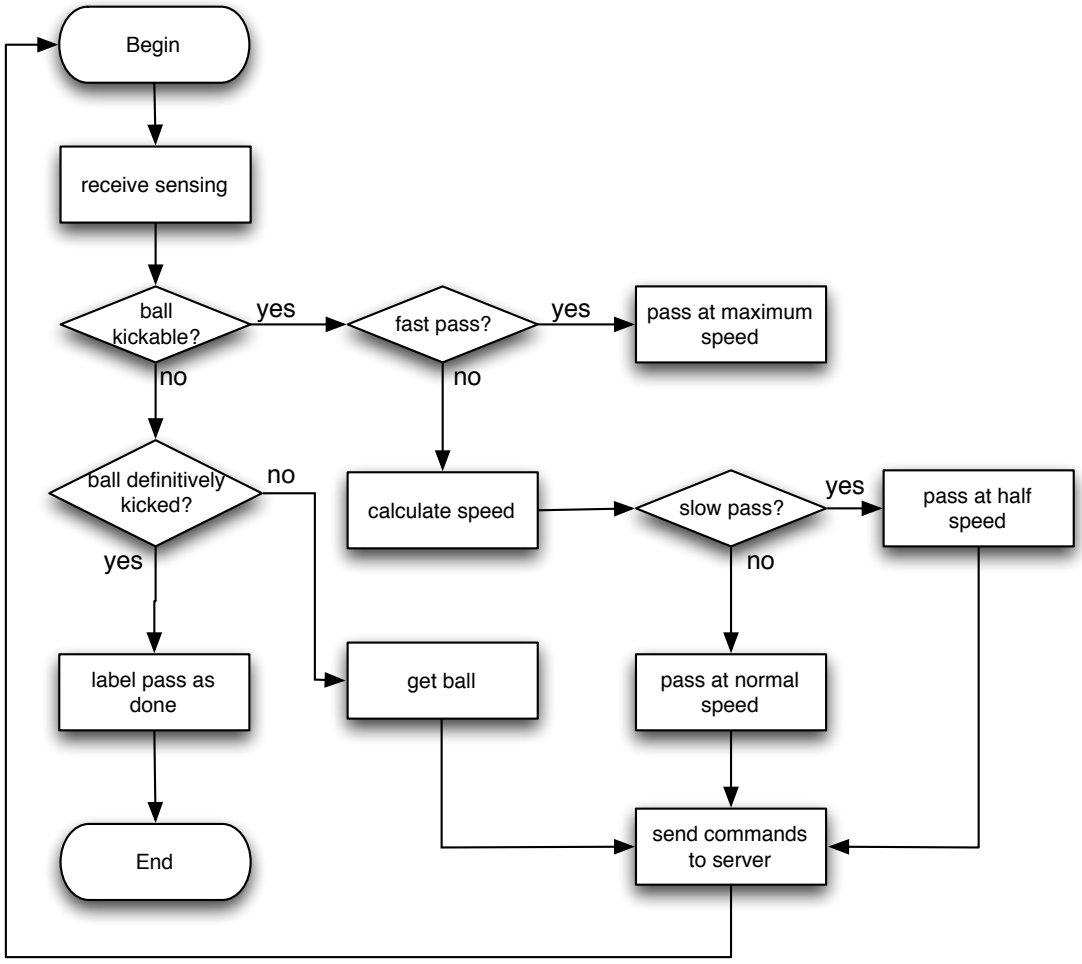


Figure 5.2: Decision process for a pass

Other actions require a more complex planning for their execution. For instance, the '*Shoot*' action, which models a shot towards the opponent goal, does not consider any kind of options. However, the execution of this action should always try to effectively score a goal, by trying to execute the shot with the maximum success probability. Every time that this action is executed, the State-of-the-World will be different, and there will be different, concurrent shooting options, All these options will have to be taken in consideration, in order to determine the shooting speed, direction and starting moment. This choice is complex, considerably more than in the aforementioned case of the passing action. In

the current stage of development, the shooting point is calculated by the State-of-the-world module. The kicking speed in presently set to the maximum speed. This is only a sub-optimal choice: if the speed was considered in more detail, one might choose a kick whose intermediate points would be outside the goal-keeper's catching area. The details to the execution of a shot can be seen in Fig. 5.3. All other *Actions* were implemented according to similar processes: check what domain actions could be used, and sequence their execution, while calculating the appropriate parameters for these.

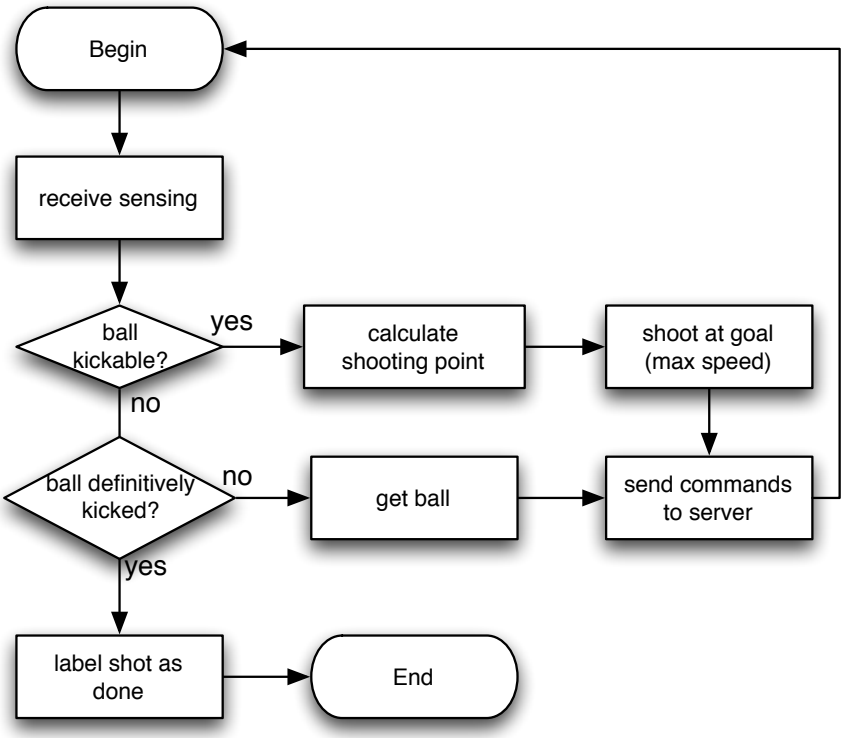


Figure 5.3: Decision process for a shot at goal

In terms of class structure, it was decided to make the class modeling the player in FC-Portugal (*PortugalInfo*) an extension of class *Action::Executor* from the Setplay Framework, as described in section 3.4. This way, a set of methods is inherited and has to be implemented: there are 20 *execute* methods, with different signatures, one for each *Action* type. With these methods implemented in this way, any *Action* can simply be given as argument to the method *execute*.

5.3.2 Implementation of Conditions

The Framework requires the abstract class *Context* to be implemented or inherited by some class in the team code. This is necessary to evaluate the *Conditions* used in the

Setplay. Since *Conditions* refer mainly to physical concepts in the soccer game and to the game state (score, time, etc), this should be done by the class that models the State-of-the-World. In the case of the FCPortugal team, this is modeled by the class *WorldState*. This class implements all the *lookup* methods referred in section 3.4 and visible in Fig. 3.9 on page 39.

As an example, one can look at the way ball possession is tested. This condition takes a list of player references as argument. The *lookup* method for this condition will iterate every player in the list received as argument, and test two conditions: if the distance of the player to the ball is below a certain threshold (currently, 1m), and if this player has the lowest ball interception time, which is different from, and more meaningful than, being the player closest to the ball. Other conditions are evaluated through checking of other concepts in the State-of-the-World.

5.3.3 Setplay Selection for Execution

The Setplay usage scenario chosen, taking in account the current level of play, was decided to take place in situations like corners, kick-ins and free-kicks, since these situations are non-play-on and clearly announced by the server, and thus all players can prepare to participate in the Setplay. There will always be some time available before Setplay start, which allows the start-up message to be repeated by the lead player, and therefore heard by all players participating in the Setplay. Ideally, the lead player would be the team coach, since it has access to an error-free state-of-the-world. The 2D Simulation rules do, though, strictly restrict the sending of messages by the coach, in order to avoid that it coordinates the players in real-time. Specifically, all messages sent by the coach are subject to a delay. With such restrictions, the coach cannot efficiently act as lead player. Thus, in the Setplays executed in the 2D Simulation league, the lead player will be a field player, which can possibly change from Step to Step, e.g., in order to allow the current ball owner to act as lead player.

In the initial experiments with Setplays in this 2D team, the algorithm for Setplay selection was of utmost simplicity. Since there were only one or two Setplays for each game situation (kick-in, corner, etc), the options were few and the choice could be done arbitrarily, in this case randomly. This procedure allowed the use of Setplays for optimization purposes: one could set up a limited set of Setplays and perform tests with them, in order to empirically evaluate their execution, or fine tune the implementation of *Actions* and *Conditions*.

These empirical experiments are appropriate for getting a good understanding of the

practical functioning of Setplays, and to perform debug procedures. However, the successful use of Setplays in a competitive environment depends on the selection and fine-tuning of particular Setplays, preferably with specific opponents in mind. Since such a process would be very time consuming, there should be, with this goal in mind, some (semi-)automatic procedure to test, evaluate and select Setplays for each opponent.

To empower the Framework to store the results of each Setplay execution, a system inspired by Case-based Reasoning (CBR) was developed, as described in section 4.2. Further, this system can choose which Setplays to execute, based on the stored evaluation, which includes data representing the success in execution, but also spatial information and the identification of the opponent.

This CBR-based Setplay selection tool was applied in the 2D Simulation FCPortugal team, which means that the Setplay selection process is performed by the Framework autonomously. This option of auto-selection of Setplays for execution was described in section 3.3.1.2, and includes the selection of the players that will participate in the Setplay.

The usage of this CBR-based system faced some difficulties. Since there is no fully shared State-of-the-World among the players, it is both possible and common that different players see events differently. It might happen, for instance, that a striker, acting as the lead player, sees that some particular condition is satisfied, and thus concludes that the Setplay has reached its end. This will trigger, according to the communication strategy, a Setplay ending message to be sent to all players through the communication channel. It might happen, though, that a defender, or the goalie, are located too far away to receive this message, due to the server's communication limitations, as described in section 5.3.5. This defender and the goalie would wrongly conclude that the Setplay ended unsuccessfully if, for instance, some abort timeout elapsed meanwhile, or some abort condition was met. Such a discrepancy would result in a distinct evaluation of that Setplay's success, and thus would result in different registries in the different players' Case Base. This discrepancy should obviously be avoided, but that is difficult to prevent with the current communication limitations.

5.3.4 Choice and execution of Actions

When the Setplay is being executed, all main management tasks and flow control are done internally by the Framework. The agent just needs to decide which action to execute, when there is more than one available, and, subsequently, take care of the actual execution of the chosen action. Besides these mandatory tasks, in the moments where there is no action to be executed, the agent may also position itself adequately for Setplay execution,

and direct its gaze towards the relevant players. The general execution flow is depicted in Fig. 5.4, and will be described in detail in the next paragraphs.

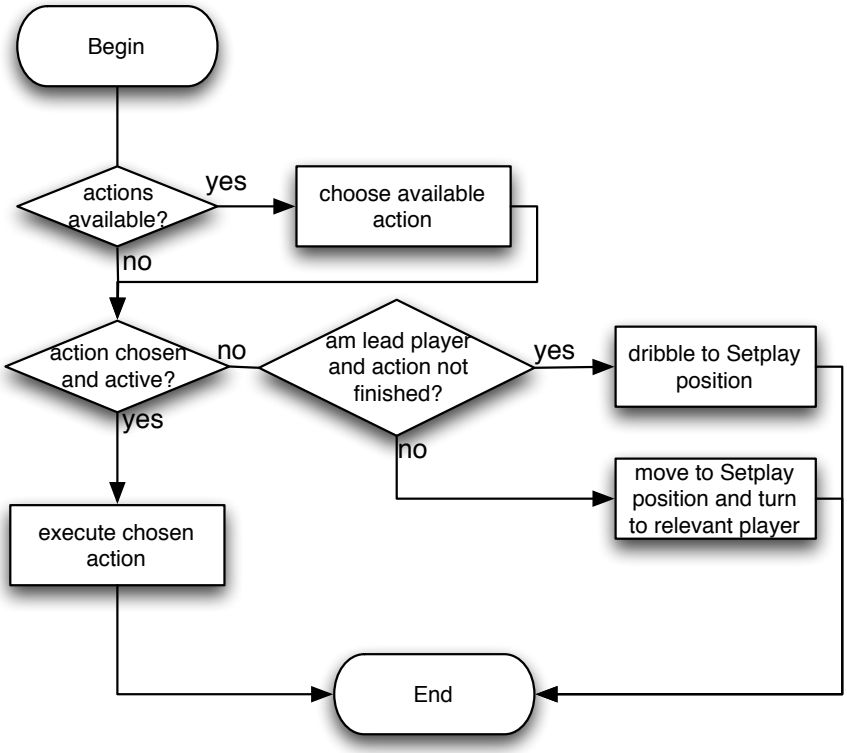


Figure 5.4: Decision process for Action choice and execution

The choice of actions was implemented very straightforwardly, as the first available action is chosen. This option was made since the Setplays in usage during testing and in actual games never considered alternative transitions between *Steps*. In such cases, there will be at most one action available at every moment in Setplay execution, and thus no real choice needs to be done. Is this setting, the simple algorithm of choosing the first available action trivially does what is needed. If more complex Setplays were to be used, namely with the mentioned alternate transitions, the action decision method could be changed, e.g., through an aleatory decision process, which would avoid similar, repeated Setplay executions, or through an enhancement of the CBR system, which would be extended to consider these choices in Setplay execution storage.

At particular stages in Setplay execution, there will be no actions active for execution by the robot: this may happen due to '*waitTime*' settings in *Steps*, because a robot has no action to execute in the current *Step*, or because the active action has already been executed. In these situations, the robot might remain inactive, or, with advantage to the Setplay execution, position and orientate itself according to the Setplay settings. In these

situations, the robot is able to access the information regarding its current Setplay position, and which is the relevant player in the current *Step*. The player's position is defined explicitly in each *Step*'s definition. As for the relevant player (or players), there is a method which returns a list of these, defined as *PlayerID*'s (method *relevantPlayers*, with no arguments). In some situations, these two issues might be contradictory: the move towards the Setplay position can be irreconcilable with looking towards the relevant player. To prevent potential conflicts between these two issues, it was decided to give a higher priority to the movement towards the Setplay position, while the player is far from it.

When the player's distance to the Setplay position is below a threshold, the focus will then be on looking towards the relevant players. Since the player's viewing area is limited, when there are more than one relevant players, the robot will direct its gaze towards the first player in the list. This list depends on the active action: if the player is, e.g., passing the ball to a teammate, the receptor will be the relevant player, while when the active action is to tackle a player, the relevant player will, naturally, be the one to tackle.

5.3.5 Inter-robot Communication

One major challenge in this application was how to deal with the limited communication means allowed by the server, which manages all allowed messaging. To cope with the restrained, single-channel communication, the lead player, in this case usually the player with ball possession, will be the only player allowed to send messages.

The content of communication must be as concise as possible, in order to follow the 2D simulator's limitations (messages under 10 bytes in length, only one message per team and cycle) and to leave enough place for the sharing of world-state information, necessary for the maintenance of a satisfactory and up-to-date world model by all players. In order to comply with these limitations, it was chosen to only use Setplays without extra arguments, since these would use much space in the startup messages (as explained in section 3.2). The Setplay startup message does therefore only contain the participating player numbers as arguments, besides the identification of the Setplay, through its numerical *id*.

5.3.6 Summary

The application of the Setplay Framework to the 2D Simulation league was quite a successful one: actions and conditions were implemented based on the existing code, and it was possible to deploy complex Setplays, involving several players and attaining the foreseen goals. Though the league's settings presented some limitations, namely the restrictions on communication, it was possible to execute Setplays successfully, involving

several players, even if some simplifications were made: the executed Setplays do not use parameters besides the participating players, in order to comply to the allowed message length.

The general evaluation on the application of the Framework to this league must thus be a strong positive one, as stated in chapter 6, where experimental results are presented and discussed.

5.4 Application to the Middle-size League

The Middle-size league has been a part of RoboCup championships since their first edition. Teams are presently made of five multi-wheeled, fully autonomous robots. These robots must thus carry all the hardware to fulfill their tasks, namely cameras and other sensors to estimate the State-of-the-World, as well as the actuators (wheels, kicking devices,...) which enable them to actively interact with the ball and other robots. According to the present rules, the field measures 12 by 18m, and the game is played with an official soccer ball. Teams may deploy a coach in an external computer, and all agents (players and coach) may freely communicate through a wireless network supplied by the competition organization, as long as some bandwidth and protocol restraints are respected.

After successful application of the Framework to the 2D Simulation league, it was decided to apply it to this real-world environment, the Middle-size League. This league's development has, in recent years, seen the leading teams' effort to enhance the high-level performance, through team level coordination and cooperation schemes, as seen in chapter 2. Namely, team CAMBADA[1], from the University of Aveiro, in Portugal, has previously developed configurable team-play for set pieces, which have, though, a fixed structure: the participating players, and the actions between them are pre-determined, but can be partially configurable, through parameters, and can be executed in different spots on the field. This cooperation scheme has been developed and fine-tuned, and has led to very successful competitive results, namely RoboCup champions in 2008. It remains, though, limited by its rigid structure, and hard to develop further. It was thus considered timely to take this kind of team-play one level higher: fully integrate the Setplay framework, which allows the execution of freely definable collective Setplays.

Development was, in a first moment, conducted in a simulated environment: CAMBADA has developed a full-blown simulator to ease the development process and avoid the constant use of the robots. The simulator fully emulates the robots' low-level functions (sensors and actuators), and hence the high-level software can be directly applied,

[1]URL: `http://www.ieeta.pt/atri/cambada/`

without any modification or adaptation, both to the robots and to the simulator, with only minor inconsistencies in the robot behavior, when compared to the real-world deployment. Development in the simulated environment is easier, and, after implementing the abstract *Conditions* and *Actions*, the robots were quickly executing Setplays.

In the CAMBADA team, agent's behavior is modeled through so-called *'Roles'*, which manage the player's decisions from an high level of abstraction. Currently, in a normal play-on situation, examples of Roles would be *Goalie*, *Defender*, *Midfielder* and *Striker*. There are, also, Roles for specific situations, like penalties and free-kicks.

Since, when taking part in a Setplay, a player will have a particular kind of control strategy, which in fact is following the Setplay's *Steps* and execute the corresponding *Actions*, it was decided that all players participating in a Setplay will be running a dedicated Role, in this case called *RoleSetplay*.

5.4.1 Setplay Selection for Execution

Setplay instantiation and startup is taken care of by the team's *coach*, which has access to the team's shared State-of-the-World and will act as lead player. This intangible agent, that does not have to deal with low-level activities, has plenty computing power to manage Setplay execution, namely Setplay selection through the CBR-based sub-system. This scenario differs significantly from the one found in the 2D Simulation league, where restrictions in communication made it impossible for the *coach* to act as lead player. This means that, in the Middle-size league, only the *coach* manages Setplay execution, and thus needs to send messages to the other agents, which makes Setplay management far easier.

In every execution cycle, the *coach* will monitor the Setplays loaded in the Framework and check if any of these has its pre-conditions satisfied. If this is the case, a Setplay will be chosen with assistance from the CBR sub-system, described in section 4.2. When a Setplay is chosen, a instantiation message will be posted on the shared-memory space, accessible to all agents, as described in section 5.4.5. Each player will see this message and, if it participates in the Setplay, will activate RoleSetplay, which will deal with Setplay execution appropriately.

As the *coach* is the agent responsible for Setplay selection, it will also be its task to manage the Setplay execution and, finally, to register Setplay success or failure.

5.4.2 Choice and execution of Actions

As seen in the previous section, the *coach*, acting as lead player, is the sole responsible for Setplay choice and activation. Since the *coach* will centralize the Setplay management issues, it will also be up to it to do all other management tasks and decisions, since it has access to the shared State-of-the-World, and is thus able to supervise Setplay execution without interference from other players. Complying to this strategical option, the *coach* will also do other relevant tasks, such as updating the Framework's internal states, choosing among possible, alternative actions included in the Setplay's definition, and checking for Setplay termination. As consequences of these tasks, the *coach* will update the Setplay-related messages, which will subsequently be read by the players.

Every player is able to detect if a Setplay is being run by checking the Setplay-related message written by the *coach* on the shared memory space, as described in section 5.4.5. Each time this message is changed from an empty text to an instantiation message, the player feeds the received message to the Framework, and the corresponding Setplay is set up. Always when the Setplay message is non-empty, the players are managed by the dedicated *RoleSetplay*. In this role, the player's behavior is very simple: it simply needs to check, at every execution cycle, if the Setplay-related message has changed, and feed it to the Framework. If some action is active, the player reads its active action, through the dedicated method, and sets up the corresponding own behavior. If there is no active action, due to the player not participating in this step, the wait time not having yet elapsed, or the action being already completed, the player will move to its Setplay position or orientate itself towards the present relevant player, in this respect similarly to what happens in the 2D Simulation league, as described in section 5.3.4.

The application of the Framework to the Middle-size league differs considerably from the application in the 2D Simulation league: since all choices are done by the *coach*, the player simply has to execute the action determined by the Framework, if it is active, or else position and orientate itself according to the Setplay's settings. Fig. 5.5 depicts the execution flow, and should be compared with Fig. 5.4: one can note that in this case, the action choice step is missing, since this is done by the *coach*.

5.4.3 Implementation of Actions

As to the implementation of the Framework's abstract actions, the process was considerably more meticulous than the one in the 2D Simulation league, since actions are more difficult to prepare and execute, as they are situated in a real environment, and not under discrete, simulated settings. This implied that some of the execution flows had to recur to some extra preparation steps.

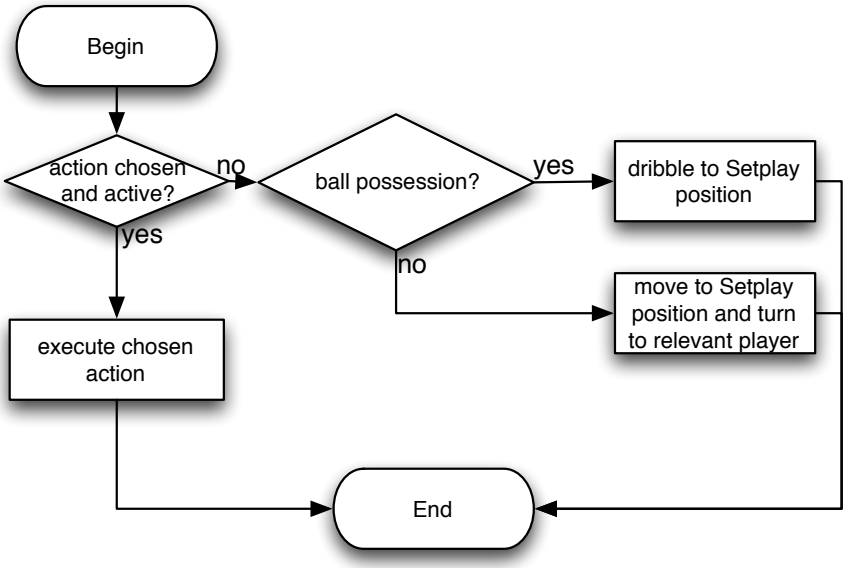


Figure 5.5: Decision process for Action execution

As an example, one will look at the implementation of the passing-related actions: the actual pass on the one hand and the ball reception on the other. In this scenario, the kicker cannot simply turn to the receiver and pass the ball: the latter has to be properly positioned, and aligned with the kicker, for the reception to be successful. If the alignment is imperfect, the ball will probably bounce at the receiver, and consequently go astray. This problem had naturally already presented itself to team CAMBADA, and had been solved by the use of some communication flags, to signal when the players are in position. One will thus look at the execution procedure of these two actions, which will make use of two different communication flags, and are depicted in Fig. 5.6. All code to execute these and all other actions is defined inside RoleSetplay.

The ball passer will start by getting possession of the ball and engaging it in the kicking device. At this phase, the passer's coordination flag will be set to *'TryingToPass'*. At the same time, the ball receiver will examine the other players' flags, to locate the player presently trying to pass the ball. When that player is determined, the receiver will position itself to receive the pass, and then set its own flag to *'Ready'*. The passer will be monitoring the receiver's flag, and will only pass the ball when it verifies that the flag's state has switched to *'Ready'*. The passer may then execute the pass, and will set its own flag to *'BallPassed'*. At this point, the receiver will actively position itself for ball reception, which implies slightly moving away from the ball upon contact with it, to avoid bounces.

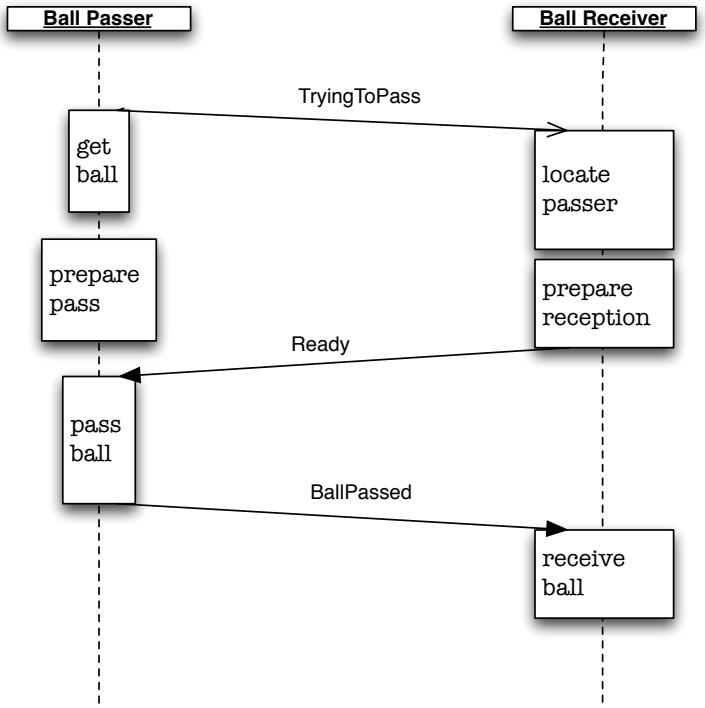


Figure 5.6: Decision and synchronization process for a ball pass

5.4.4 Implementation of Conditions

As mentioned before, the Framework requires the abstract class *Context* to be implemented by some class in the team code, in order to evaluate the *Conditions* used in the Setplay definitions. As *Conditions* refer mainly to physical concepts in the soccer game and to the game state (score, time, etc), this implementation is preferably done by the class that models the State-of-the-World. In the case of team CAMBADA, this class is called *WorldState*, and thus it was made a subclass of the *Context* class in the Framework.

Special attention was given to passes between players: the new *canPassPl Condition* has to assure that both players have no obstacles between them, or else the pass will frequently fail, since the ball catching capabilities of the robots are limited, and obstacles might hinder ball visibility.

5.4.5 Inter-robot Communication

Communication in the Middle-Size League is not subject to strict limitations: although the robots are fully autonomous, they can freely communicate through a wireless network, using standard protocol 802.11. In championship settings, it is common that the

network has high traffic and is consequently over-loaded, but this situation is normally under control.

The CAMBADA team uses a black-board approach with a shared State-of-the-World for communication. This architecture, called RTDB (Real Time Database - Lau et al. (2008); Santos et al. (2009)) has been used both for posting perception information and coordination flags in previous tackles at team-level coordination. The RTDB was therefore used for the posting of the Setplay messages by the *coach*, since these can be read by all the team robots. All kinds of Setplay-related messages, namely for instantiation, *Step* change and Setplay end, are written in a particular location in the shared memory, which is known and accessible by all players. As this message changes along with Setplay execution, players will use it to update the Framework's internal state, by calling the *processReceivedMessage* method. This actualization will determine the active actions at each moment, as described in section 5.4.2. As this communication framework was already fully functional and permanently used, its usage by the Setplay Framework only required a minor change in the information architecture, namely the addition of a specific field to convey the Setplay-related messages.

5.4.6 Summary

The application of the framework to the CAMBADA team was considerably different from the application to the 2D Simulation team. For once, the low-level perception and action mechanisms are far more complex and thus subject to malfunctions and other uncertainties or unexpected events. This implied that the application of the Framework to this team, and the necessary adaptation of components, had to be more finely tuned. On the other hand, the existence of a central shared memory, with free access by all agents, made the process of Setplay choice and management clearly considerably simpler.

The resulting Setplays were executed smoothly and effectively. In a first stage, the already existent, rigid Setplays were successfully reenacted, by defining new Setplays with similar behavior. These Setplays could further profit from the advantages offered by the CBR-based system: when competing Setplays had different success rates, the more unsuccessful alternatives were chosen more seldomly. Additionally, since the Framework is free from the previous limitations, it was possible to define new Setplays, namely for usage in play-on situations. The results of the execution of such Setplays are described in chapter 6.

Chapter 6

Evaluation of the Setplay Framework

As the Setplay Framework aims at being a practical tool, that is ready to be integrated in real-game situations, its evaluation must be based on credible situations, i.e., on games according to the actual rules. Since the Framework application to a team brings several advantages that should be evaluated, several of its characteristics must be tested.

One should remember, from page 4 of the Introduction in Chapter 1, that Setplays should be considered "small multi-player plans in the robotic soccer domain", and they should be applicable "to any league in RoboCup". Specifically, the Framework claims to exhibit several qualities:

Efficacy: a Setplay should be effective in gaining advantages in the particular situations and against the opponents for which it was designed.

Generality: the Framework should be applied seamlessly to different teams and leagues, as it has already been described in the previous chapter. Further, the same Setplay should possibly be executed in different leagues, with different constraints in terms of action execution and communication.

Configurability: a Setplay can possibly contain several parameters, and its definition is highly configurable, in terms of player positions, as well as which actions to execute, and when to do so. This configurability provides a high level of fine-tuning, that can be decisive in the Setplays success.

Adaptability: when the Framework is managing different Setplays in the course of a game, it should adapt to the current opponent, by choosing the most promising Setplays, while avoiding those which prove to be unsuccessful.

Experiments will focus on the comparison of the team behavior in specific situations, where alternative executions with use of Setplays will be compared to situations where the

Setplay Framework is inactive. The qualities just mentioned will be subject to qualitative and quantitative testing in the remainder of this chapter.

6.1 Testing in the Middle-Size League

Over the recent years, Set-pieces have been used extensively by team CAMBADA: in fact, these collective moves have been one of the strong points of the team, giving them a decisive competitive advantage. Set-pieces are fixed, yet configurable, collective collaboration schemes, and have been applied to situations like free-kicks, corners and kick-ins. Setplays are particularly suited to this kind of situations, even if they can be applied in other settings. As such, the initial experiments made tried to reenact this kind of collective moves, and situations like kick-offs and kick-ins, among others, were effectively executed and their outcome monitored.

6.1.1 Set-pieces: Throw-in

In order to illustrate Setplay execution, a throw-in will be used as example. There are three robots involved: the *replacer*, which kicks the ball into play by passing it to the *receiver1*, that positions itself at a configurable *x_offset* behind the ball. An auxiliary *receiver2* simply positions itself at a fixed offset relative to the ball and has no further participation in the Setplay. Naturally, there can be more complex Setplays, with more parameters or varying receivers: such complexities are avoided in this example for clarity's sake. This Setplay's definition can be seen in Fig. A.2, in appendix A.

The Setplay is initiated by the *coach* upon announcement, by the referee, that the throw-in in favor of team CAMBADA can be taken. The participating players are chosen according to their distance from the Setplay positions. At this moment, the coach posts the following message on the team's black-board (RTDB), stating that the participant players have jersey numbers 4, 2 and 3, and 2m for the *x_offset* argument:

```
(startSetplay :setplayID 0 :participants 4 2 3 :parameters 2)
```

Upon reading this message, these players position themselves in the desired positions (see diagram on Fig. 6.1(a)). After the players reach their positions and the referee sends the 'start' signal, the *coach* will announce, as follows, that the desired next step is nr. 1, which in this case is the only available option:

```
(stepMessage :currentStep 0 :nextStep 1)
```

The *replacer* will then pass the ball to the *receiver1* (see, again, Fig. 6.1(a)). This robot will accomplish the '*receiveBall*' action by waiting for the ball and, when it comes close, moving back to avoid it to bounce and go astray (Fig. 6.1(b)). After catching the ball, the *coach* will check if it considers a shot at goal possible, in which case it will post a new step change message, which triggers the a dribble to the shooting position and shot at goal through a kick behavior, visible on Fig. 6.1(c):

```
(stepMessage :currentStep 1 :nextStep 2)
```

In case the shot is not possible or desirable, the setplay will be aborted, with the corresponding message being sent, where -1 stands for an inexistent state:

```
(stepMessage :currentStep -1 :nextStep -1)
```

If the shot is successful and a goal is scored, the pre-condition to step 3 will be satisfied. This will make the Setplay to follow a '*Finish*' transition, which represents a successful Setplay end. This would be signaled to the players in the team through the message that follows:

```
(stepMessage :currentStep 3 :nextStep -1)
```

6.1.2 Setplay in play-on mode

In order to illustrate Setplay execution in a play-on situation, a very simple interaction will be used as example, possible when the ball is in possession of our team, in a specific region (intersection of *mid_left* and *their_back*). There are two robots involved: the *striker* will turn and then pass the ball to the *shooter*, which will position itself in a central point and, upon reception of the pass, will shoot at goal. This Setplay's definition can be seen in Fig. A.3, in appendix A.

The execution of this Setplay will be illustrated through images, displayed in Fig. 6.2, of an actual execution in CAMBADA's simulator, which uses the same code that runs on the actual robots.

The Setplay is initiated by the *coach* upon verification that the conditions for entry in step 0 are satisfied: play-mode is play-on, ball is in the desired region and one CAMBADA player has ball possession. The participating players are chosen according to their distance from the Setplay positions. At this moment (see diagram on Fig. 6.2(a)), the *coach* posts the following message on the RTDB, stating that the Setplay with '*id*' 1, with the participant players with jersey numbers 6 (representing the coach), 5 and 3:

```
(startSetplay :setplayID 1 :participants 6 5 3 :parameters )
```

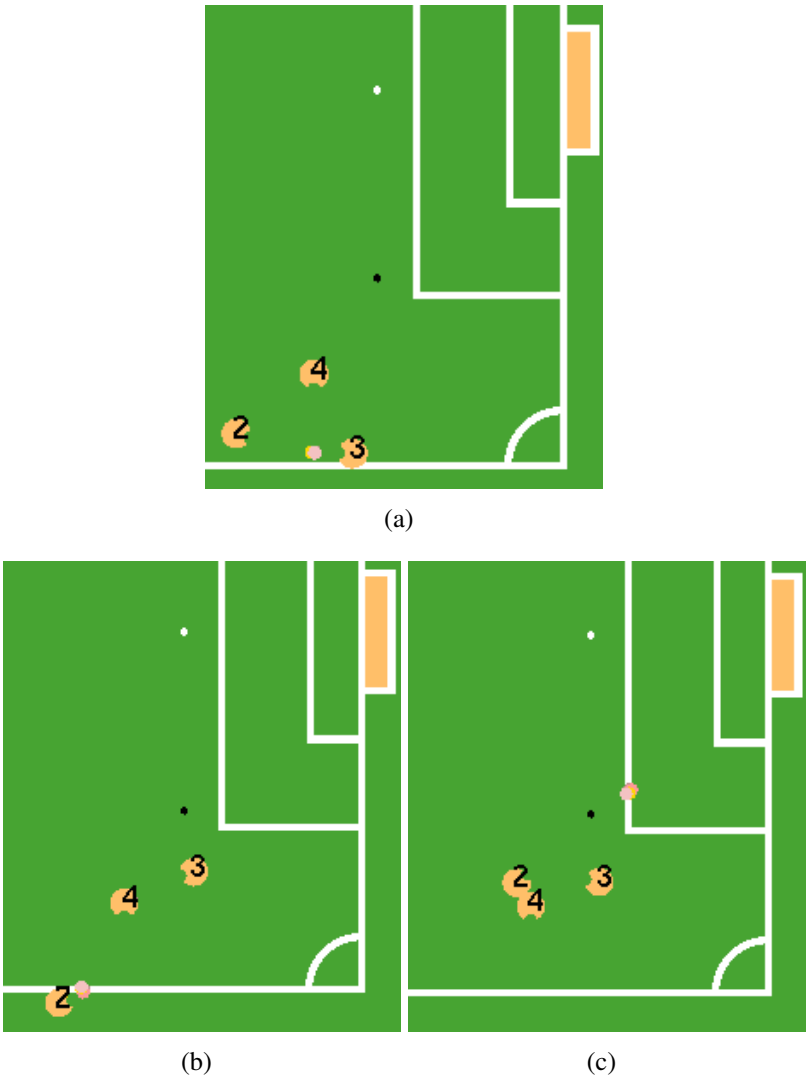


Figure 6.1: Throw-in Setplay example

Upon reading this message, these players position themselves in the desired positions. Since the desired pass can be accomplished, the *coach* will announce, as follows, that the desired next step is nr. 1:

```
(stepMessage :currentStep 0 :nextStep 1)
```

This will allow the *striker* to rotate towards the *shooter* (Fig. 6.2(b)). After the players reach their positions, another step progress is announced by the *coach*:

```
(stepMessage :currentStep 1 :nextStep 2)
```

The *striker* will then pass the ball to the *shooter* (see Fig. 6.2(c)). The *shooter* will accomplish the 'receiveBall' action by waiting for the ball and, when it comes close,

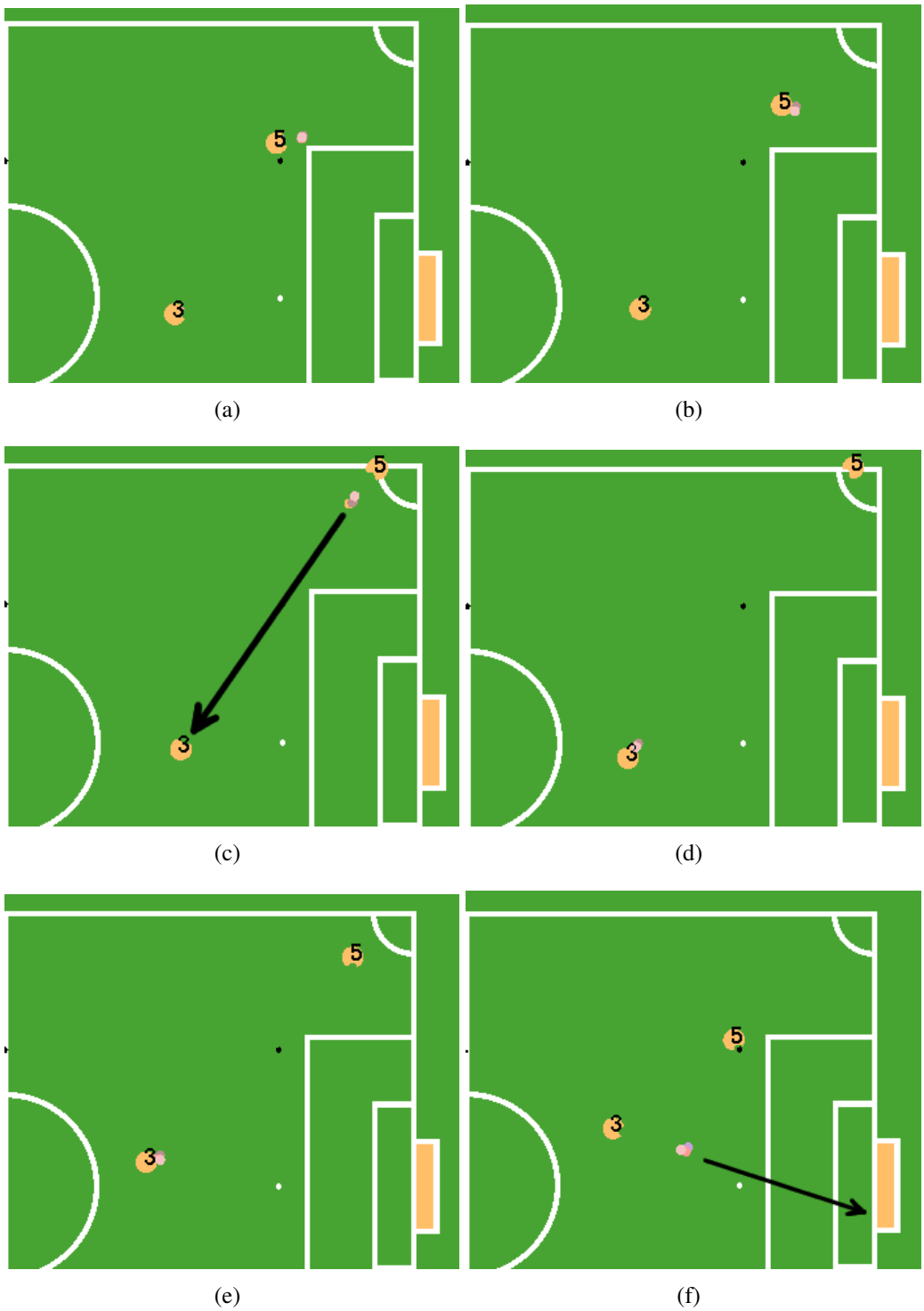


Figure 6.2: Play-on Setplay execution steps.

moving back to avoid it to bounce (Fig. 6.2(d)). After catching the ball, the *coach* will evaluate if it considers a shot at goal possible, in which case it will post a new step change message:

```
(stepMessage :currentStep 2 :nextStep 3)
```

This triggers the preparation (see Fig. 6.2(e)) for a shot at goal through a kick behavior, which in turn is visible on Fig. 6.2(f). In case of a goal being scored, the Setplay will be finished, with the corresponding message being written on the RTDB by the *coach*, where `-1` stands for an inexistent state:

```
(stepMessage :currentStep 3 :nextStep -1)
```

6.2 Testing in the 2D Simulation League

The main testbed for Setplay execution was the 2D Simulation FCPortugal team. In this team, most actions and conditions were implemented and allowed the appropriate execution of various Setplays, in different settings, namely in real games in competitions. Some actions, like marking a player, were not actually tested, as they are useful only in defensive scenarios, while, at this point, only offensive Setplays were applied.

Even though some of the actions may still lack a final round of fine-tuning, which might enhance performance, one may conclude, based on evidence shown in the next sections, that the results reached are a considerable improvement upon the original team's behavior.

6.2.1 Own Goalie Catch

The goalie catch is a very common situation in real games, since opponents frequently try to forward the ball to FCPortugal's penalty area, and the goalie is capable of intercepting the majority of these. After a goalie catch, a free kick is awarded, and the goalie may move ("warp") twice to different positions before actually taking the free kick.

The team's original handling of these situations showed there was room for improvement, since it allowed, too frequently, ball interception by the opponent inside the own half, originating difficult situations and, sometimes, goals by the opponent.

6.2.1.1 Nemesis

In order to overcome the weaknesses in goalie catch situations, several different Setplays were designed, namely for usage against one frequent and competitive opponent: Nemesis, from the Amirkabir University of Technology, in Iran (Norouzitallab et al., 2010). This team ranked 5th in RoboCup 2010 and 8th in the previous year. In both occasions, Nemesis' rank was near, but above, FCPortugal's rank. One might thus say that Nemesis

is a team with which FCPortugal competes for rankings in the same range, and is, due to this fact, a frequent opponent. As such, Nemesis' 2010 binary was an obvious opponent to be considered when Setplays were designed to be used in competition. It was believed that, if existent flaws could be exploited, Setplays could make a difference in the final score of the games.

For a better understanding of the advantages of Setplay use, one will first analyse FCPortugal's performance without the application of Setplays. In order to centre the analysis on this kind of situations, goalie catchs were repeatedly provoked, through operations on the 2D Simulation monitor[1]: the ball was dropped on the penalty area, in play-on mode, in a place where the goalie could catch it. The game continues normally from this point on, until the opponent gains persistent possession of the ball, or until the game is interrupted by the referee, due to, e.g., some foul or the ball getting kicked out of the pitch. At that moment, the momentary position of the ball and game mode are recorded, and a new goalie catch is induced.

To analyse the performance of this kind of situations, one can not rely on the number of scored goals, as rarely a goal is reached as a direct result of a goalie catch. A simple, empirical utility function was designed to rate the outcome of each of the trials. The ball's final position was awarded points according to how far from the own goal it was situated, as defined in Table 6.1. A special category was created for situations where the ball was shot at goal. This spatial ranking will be doubled in situations when the ensuing game mode will result in FCPortugal's ball possession, like own kick-ins and free-kicks.

Area	**Points**
shot	12
their_back	6
their_middle	5
their_front	4
our_front	3
our_middle	2
our_back	1

Table 6.1: Ranking according to final ball position.

Ten different games were run, and, in each game, five goalie catches were induced, in favor of FCPortugal. Fig. 6.3 depicts graphically the performance of this series of tests: each column represents a game, for a better representation of the disparity between different games. The x-axis shows the number of goalie catches, while the y-axis represents the

[1]URL: `http://sourceforge.net/projects/sserver/files/rcssmonitor/`

accumulated evaluation. For instance, in execution number three, the value depicted will be the sum of evaluations from executions one to three, shown separately for each game. In general terms, the average outcome amounted to 3.4, which spatially would roughly correspond to the half-way line. Globally, the ball was handled across the half-way line in only 40% of all goalie catchs.

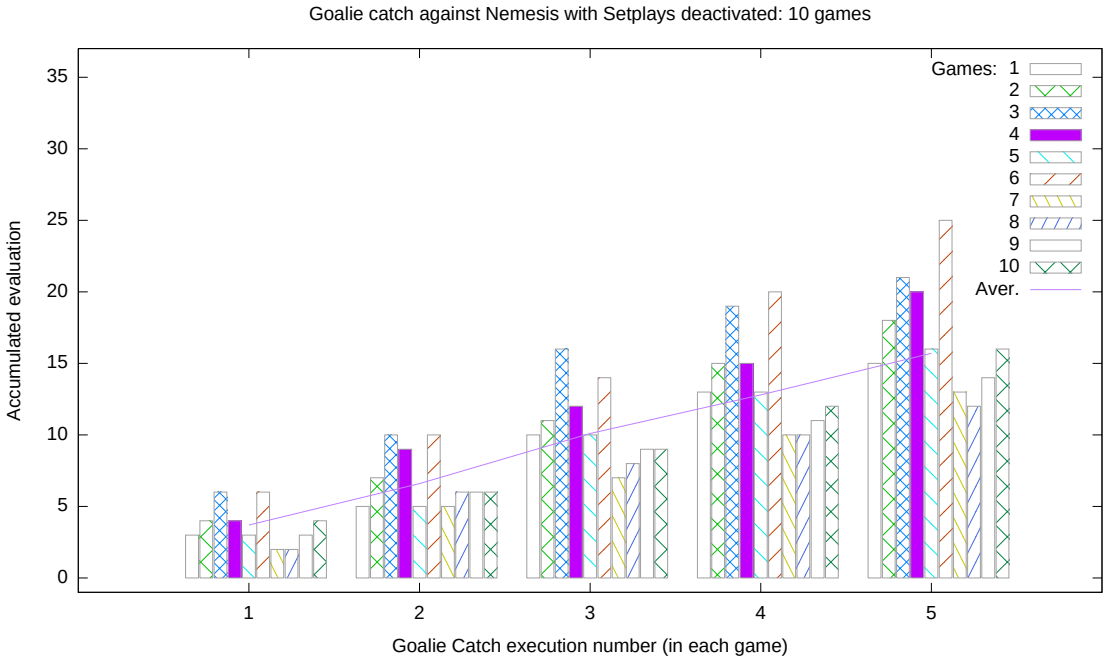


Figure 6.3: Goalie catch with Setplays deactivated

The diagram on Fig. 6.4 gives useful information in terms of speed of handling the ball across the half-way line. Each execution in each of the ten games is characterized by the time taken for the ball to cross to the opponent's half. Executions where the ball never crossed the half-way line are labeled with the value -1.

Setplay efficacy Different Setplays were designed with the goalie catch situation in mind, and then fine-tuned to take advantage of Nemesis positioning on the field in this situation. Some of these Setplays showed better results, and were thus chosen to be used in competition. To evaluate the advantages brought by these Setplays, two of them were chosen to be presented here. These will be distinguished in this text by the number of participating own players: four in the first case, and six in the second.

The Setplay with four players is quite simple, and is depicted through screenshots on Fig. 6.5: besides the goalie, three other players, the right defender, mid-fielder and forward position themselves half-way near the right touch line, as seen in Fig. 6.5(a) . When the goalie is about to kick the ball, it moves to the right of the penalty area, and all

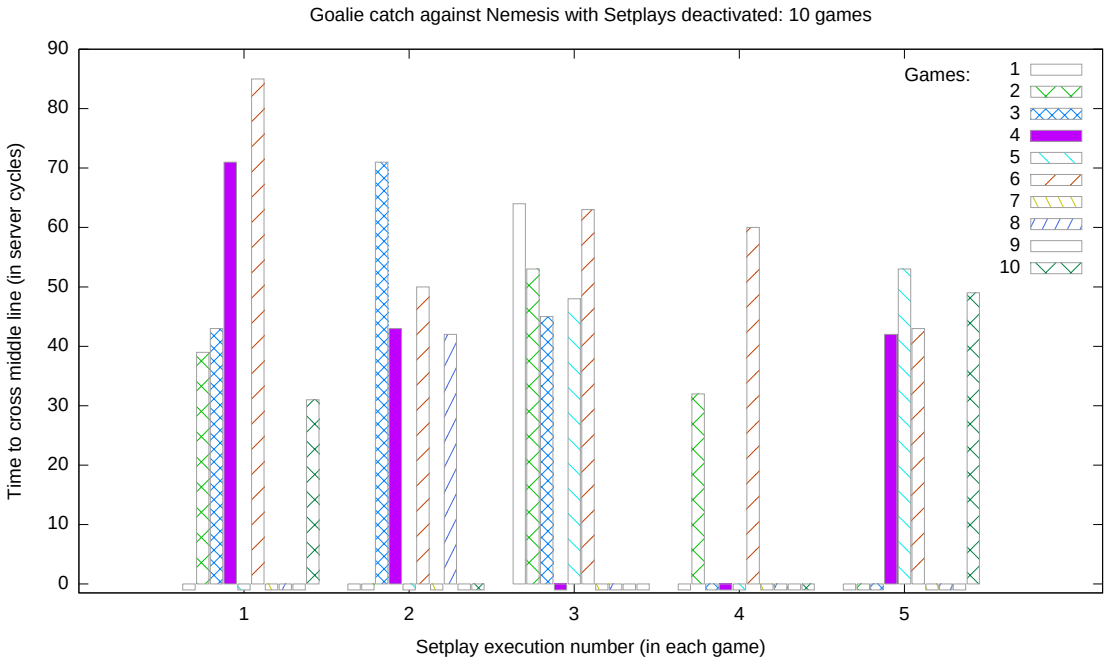
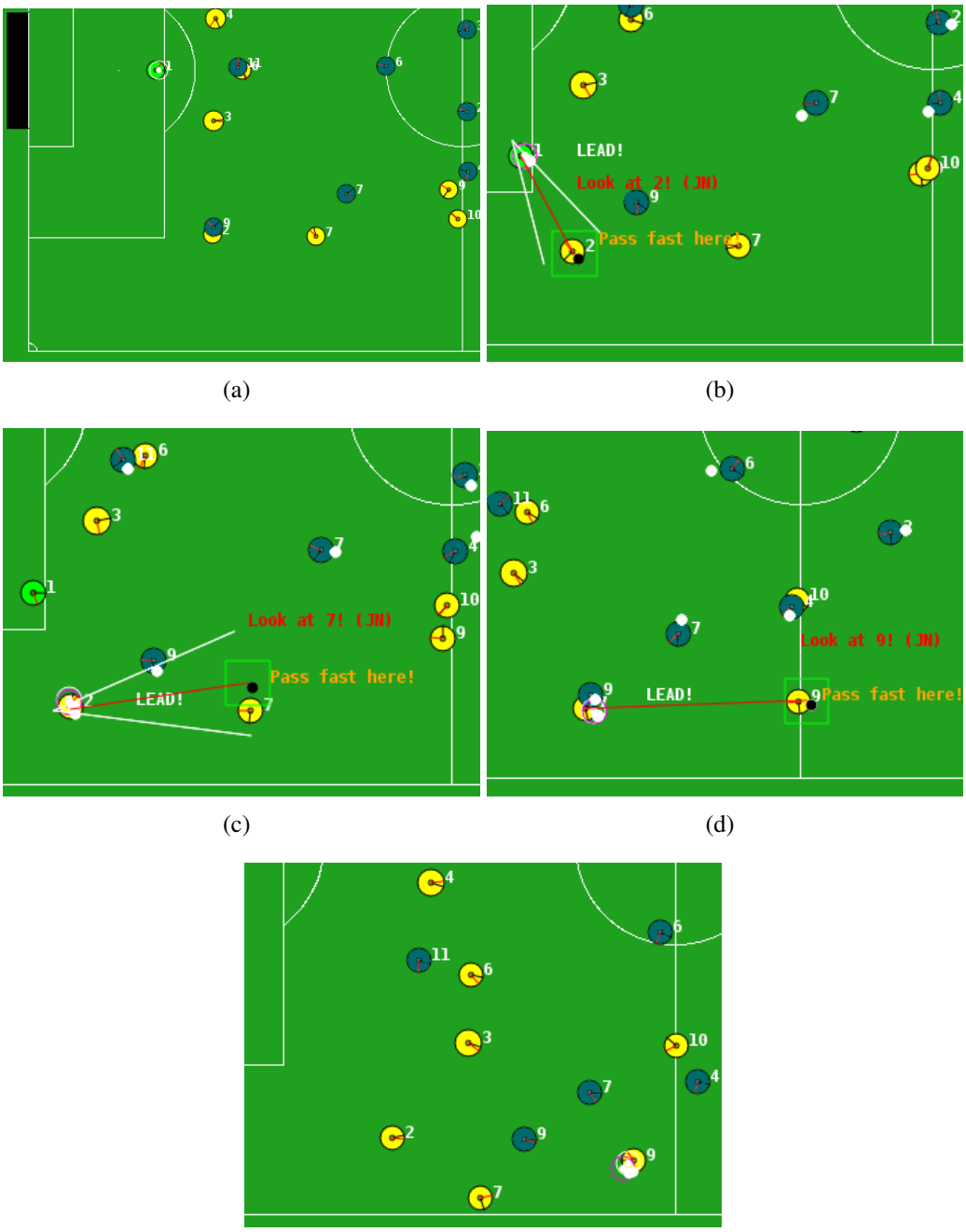


Figure 6.4: Goalie catch with Setplays deactivated: time to cross middle line

other players move towards the touch line. The ball is then passed from the goalie to the defender (Fig. 6.5(b)), and then from the defender to the mid-fielder (Fig. 6.5(c)), and on to the forward (Fig. 6.5(d)), at which point the Setplay ends, close to the half-way line. This Setplay's textual definition can be seen in Fig. A.4. Since this Setplay is invertible, it was also used, during the tests, on the left side of the field.

The Setplay with six players is both more complex and more ambitious, and is depicted through screen-shots in Fig. 6.6. Similarly to the previous Setplay, some players, in this case four, position themselves half-way to the touch line (Fig. 6.6(a)), and the ball is sequentially passed forward along these (Fig. 6.6(b)-6.6(d)), until it reaches the last player, which in this Setplay is named '*Kicker*' (nr. 9). Upon gaining the ball possession (Fig. 6.6(e)), the '*Kicker*' will dribble towards the opponent's goal line, as it was observed that Nemesis left this whole right flank unattended. While the '*Kicker*' is dribbling, another player, called '*Runner*' (nr. 10), will run towards the opponent's offside line. The Setplay might end successfully in two different situations. If the '*Runner*' reaches the offside line and the '*Kicker*' judges possible to forward the ball to a point ahead of the '*Runner*', this kick forward is executed (Fig. 6.6(f)). If, before such a forward kick is possible, some opponent comes near the '*Kicker*' and tries to dispute the ball possession, the Setplay is immediately terminated, and the game continues normally with the ball at the '*Kicker's*' discretion. Fig. A.6 contains the textual definition of this Setplay. This Setplay is also invertible.



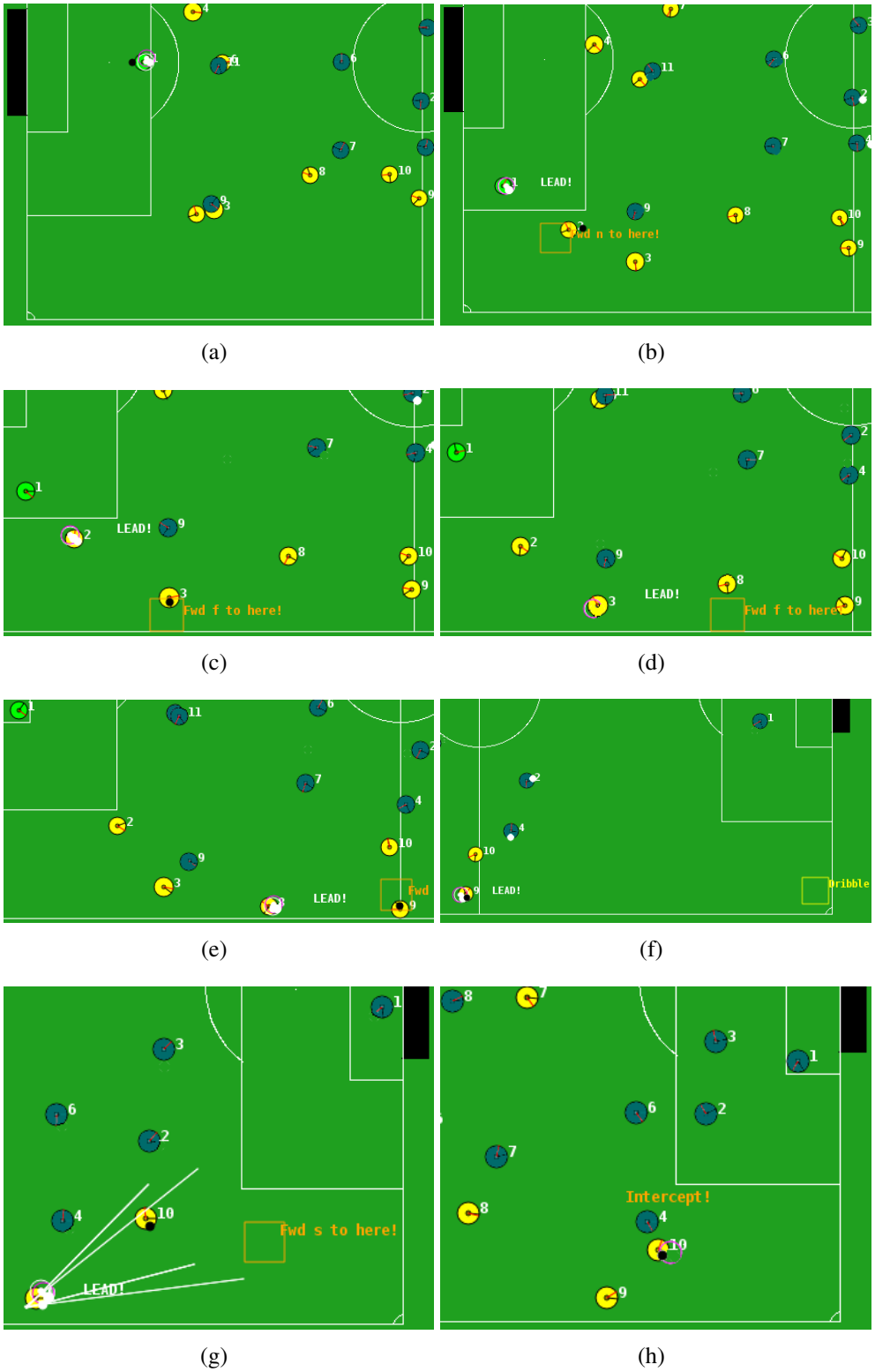
(a) (b)

(c) (d)

(e)

Figure 6.5: Goalie catch with four players: FCPortugal pictured yellow, Nemesis dark blue, ball in centre of the pink circle.

The performance of these Setplays was evaluated in exactly the same terms as in the case with no Setplays, presented previously. Ten games with five goalie catchs were run, and the out-coming ball position and game mode were rated according to the same algorithm, based on the points shown in Table 6.1. These results are presented graphically



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 6.6: Goalie catch with six players: FCPortugal pictured yellow, Nemesis dark blue, ball in centre of the pink circle.

in Fig. 6.7.

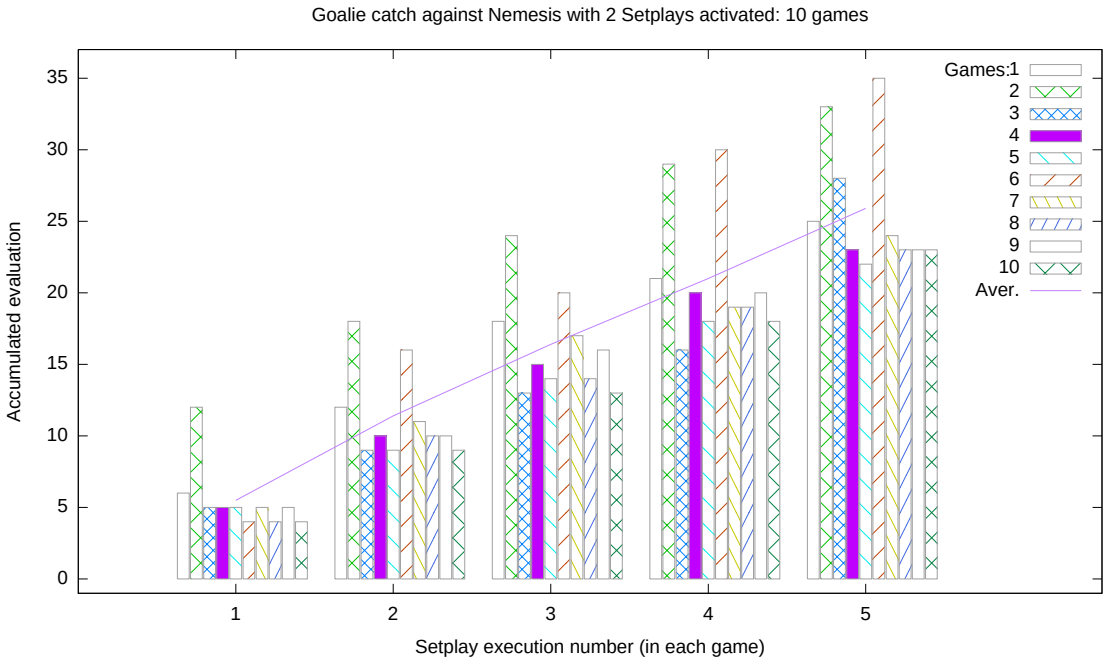


Figure 6.7: Goalie catch with 2 Setplays activated

In terms of Setplay evaluation, the total average amounts to 5.18, which is approximately a 50% increase with respect to the evaluation of the games without Setplays. Other interesting results concerning the time the ball took to cross the halfway line can be seen in Fig. 6.8. Not only are these times typically in the range of 30 to 40 cycles, but one should also note that only in 5 executions did the ball not cross the half-way line. This number of flawed executions is only 10% of the total number of executions, which differs considerably from the 60% in the previous case with no Setplays.

There were, in these tests, two Setplays available, and choice among them was done by the CBR-based selection algorithm. No previous experience was, though, considered, which means that only experience in each game was taken in account. Even so, there was a considerable bias towards the choice of the Setplay with 4 players. This result is understandable: that Setplay is simpler, with less participating players, and ends earlier than the Setplay with six players. In fact, the Setplay with 4 players had a 84% success rate, while the one with six players managed to be successful in 67% of the cases. Since the CBR algorithm favors considerably the Setplays with more success, the Setplay with 4 players was chosen in 76% of the total of executions.

From another point of view, the average evaluation of the Setplay with 4 players was 4.79, while for the Setplay with 6 players the figure was 6.42, which is considerably better. Moreover, all executions where the ball did not cross the half-way line were of

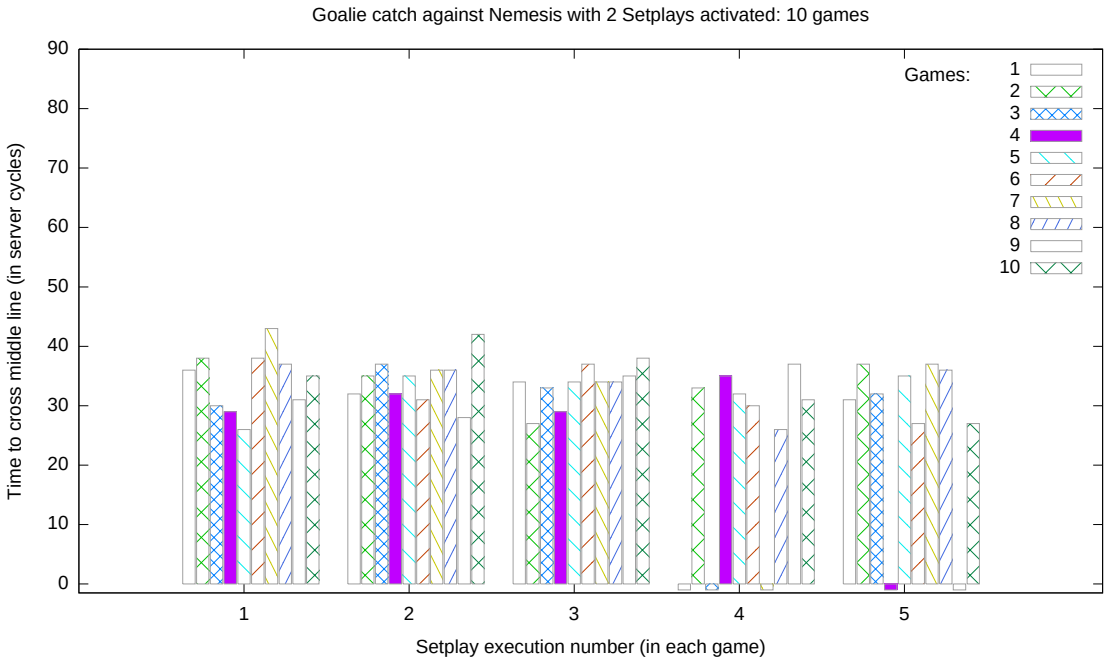


Figure 6.8: Goalie catch with 2 Setplays activated: time to cross middle line

the Setplay with 4 players. This evaluation data is, though, not considered on CBR-based Setplay choice, which relies solely on past Setplay success rates. One must thus note that the Setplay with the best evaluation was not chosen more frequently. This result should be considered, and possibly motivate enhancements in the CBR-based selection algorithm.

6.2.2 Corner Kick

Corner kicks are situations that are clearly suitable for Setplay execution, since opponents tend to use a stable player positioning strategy for field covering that often depends on ball position, and is thus always similar in corner kick situations. Such a situation is therefore easy to exploit. Similarly to the goalie catch, several Setplays were designed for corner kicks. This section will look at the performance in corner kicks both against Nemesis and team Bahia.

6.2.2.1 Tests against Bahia

Team Bahia[2] has started participating in RoboCup competitions only recently, since 2007. The team has been continually developed, but it still remains a not very competitive team. FCPortugal has performed efficiently in games against Bahia, but it is interesting to see how this performance could be enhanced by the application of Setplays. As such, games

[2]URL: `http://www.acso.uneb.br/brt/`

against Bahia can be a suitable testbed for different aspects in Setplay development, and Bahia has thus been extensively used as an opponent for Setplay development.

Specifically with respect to corner kicks, Bahia's 2010 binary has been used as an opponent to test different aspects of the Setplay Framework, which will be presented in the remainder of this section. Initially, a set of tests was done with the Setplay Framework deactivated. The results of these executions can be seen on Fig. 6.9. From these result, one may highlight that 13 goals were scored, which corresponds to 26% of all executions.

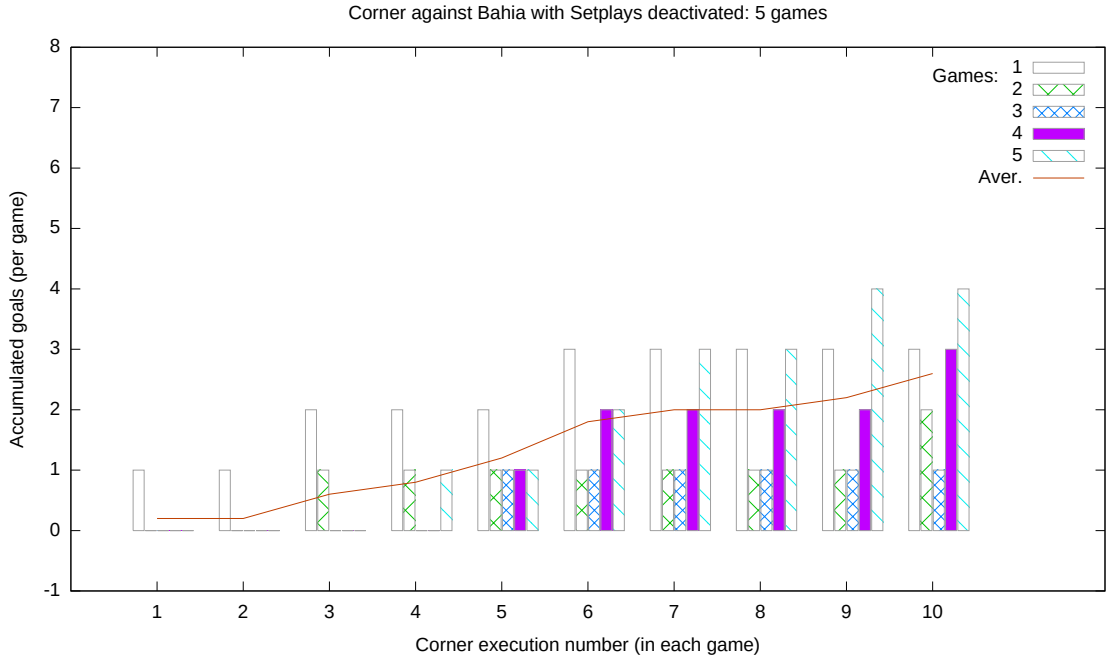
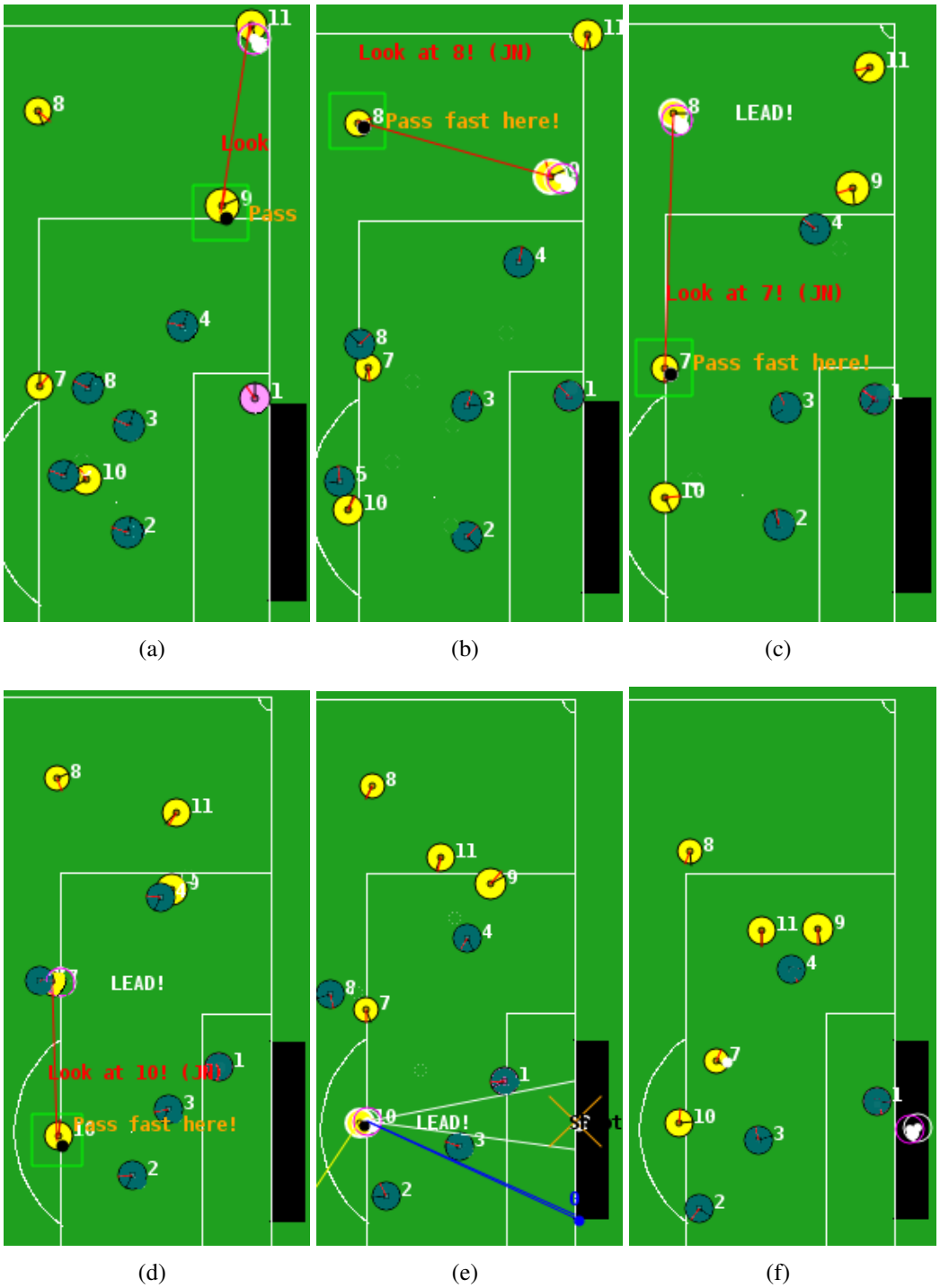


Figure 6.9: Corner against Bahia with Setplays deactivated.

Setplay efficacy A Setplay was developed specifically to take advantage of Bahia's positioning on the field, and has been fine-tuned in sequential iterations. The definition of this Setplay can be seen in Fig. A.9, in Appendix A.

This Setplay involves the participation of five players, as depicted in Fig. 6.10. The kick taker (nr. 11) passes the ball to the receiver (nr. 9, Fig. 6.10(a)), which forwards the ball to the left striker (nr. 8, Fig. 6.10(b)). This player is in line with two other strikers (players nr. 7 and 10), to whom the ball is passed along (Fig. 6.10(c) and 6.10(d)). When the ball reaches the right striker, it is shot at goal (Fig. 6.10(e)).

The results from the execution of this Setplay were quite encouraging, since they clearly outperformed the executions without Setplays, as seen on Fig. 6.11, which shows the number of accumulated goals per execution, in five different games. The total number of goals was 28, from a total of 38 shots at goal. This amounts to a 56% success rate,



(a) (b) (c)

(d) (e) (f)

Figure 6.10: Corner with Setplays activated: FCPortugal pictured yellow, Bahia dark blue, ball in centre of the pink circle.

up from 26% in the case of executions with the Setplays deactivated. As seen by the line representing the average on the figure, the accumulated number of goals increases steadily as more corner kicks are executed.

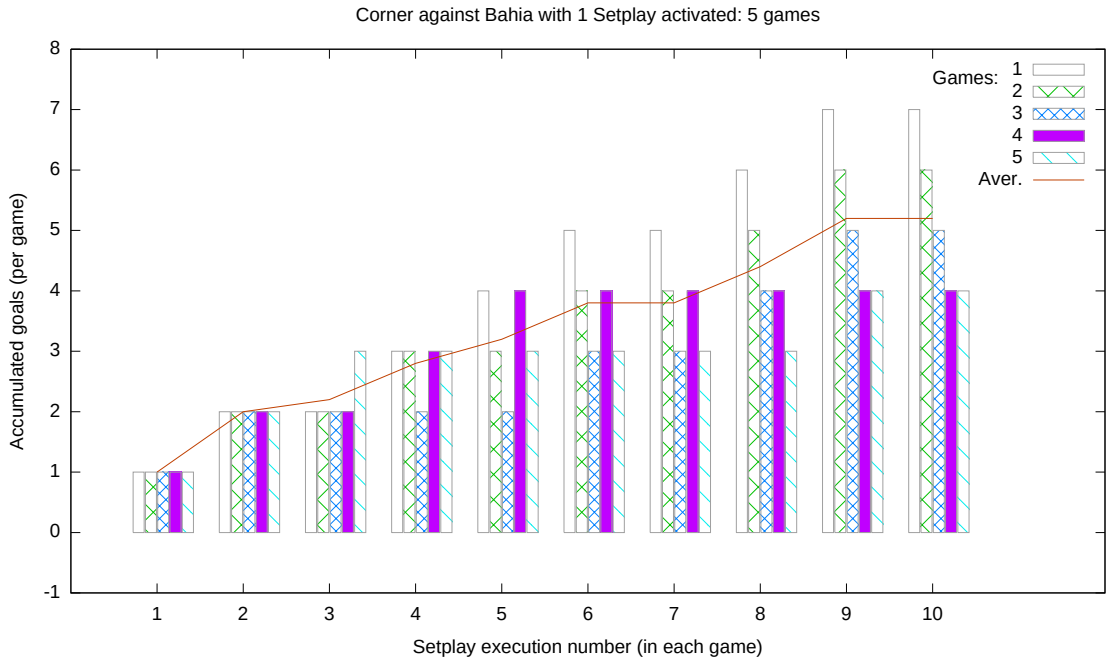


Figure 6.11: Corner against Bahia with an activated Setplay.

To get a clearer view of the difference of performance with and without Setplays, the diagram in Fig. 6.12 was drawn, putting the average performances side to side. The error bars depict the standard deviation, in terms of accumulated goals, in each of the executions. As clearly visible on the diagram, average performance is very different, and there is no overlap between the standard deviations. This emphasizes the difference between the two performances.

Since the samples with and without Setplay usage have an equal dimension, in this case above 30, one can assume that the average of the experiences have a normal distribution. Thus, one will apply a two-tailed t-student test to the null hypothesis $H_0 : \mu_1 = \mu_2$, i.e., that the average of both samples, with and without the usage of Setplays, is equal.

To evaluate this hypothesis, one will calculate the following t-student test:

$$T = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{S_1^2 + S_2^2}{n}}} \tag{6.1}$$

In equation 6.1, $\overline{X_1}$ and $\overline{X_2}$ represent the average of goals in the samples with and without the usage of Setplays, which are, respectively, 0.26 and 0.52. S_1 and S_2 represent the standard deviation in each of these samples, which amount to 0.44 and 0.50. n stands for the number of experiments in each sample, which is 50. Applying this data to the formula, one comes to 2.74 for the value of T. For a significance level of 0.01, the value on a t-student distribution table with 98 degrees of freedom $(2n-2)$ is 2.63. Since the

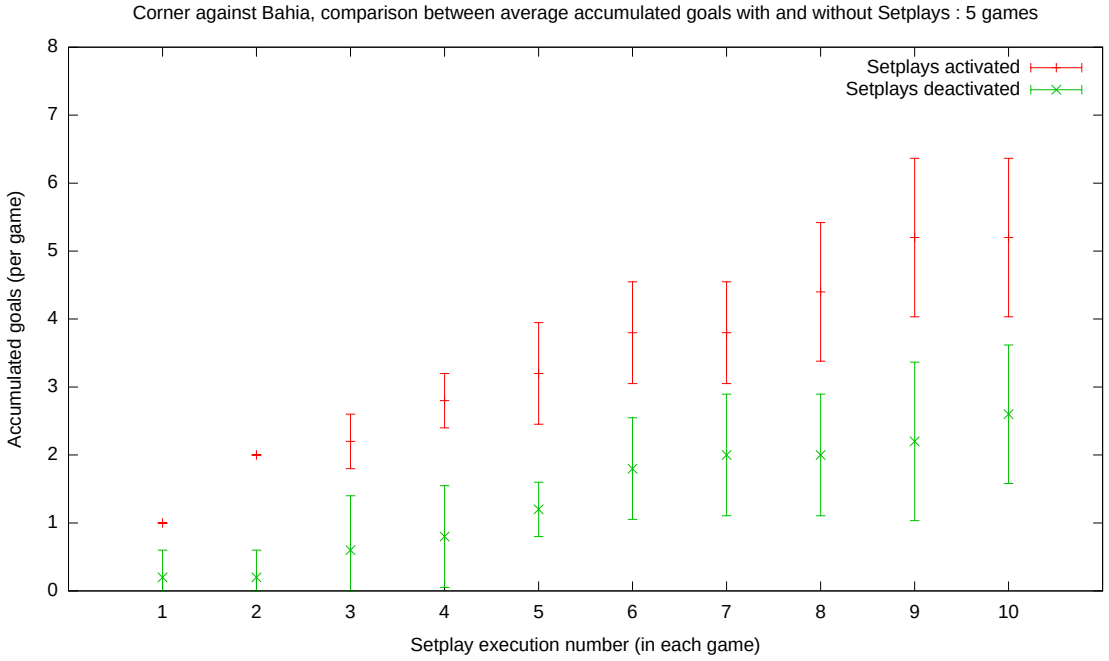


Figure 6.12: Corner against Bahia with and without an activated Setplay

obtained value of 2.74 is higher than 2.63, one rejects the null hypothesis, and thus there is statistical evidence to affirm that the average of goals with and without Setplay usage is different.

CBR performance The CBR-based Setplay selection algorithm was already present in the goalie catch tests against team Nemesis, but its influence in game playing was not adequately assessed. In this section, tests made specifically to examine this system's performance will be presented and critically discussed.

The previous section presented a Setplay, with five participating players, which performed quite satisfactorily against Bahia. In the present set of executions, again, five games were run, and ten corner kicks were induced in each game. FCPortugal will possibly execute two different Setplays: the competitive one, with five players, from the previous section, and a new Setplay, which simply consists of forwarding the ball to the front of the goal, which has a much lower success rate, and will be referred to as simple corner, and whose definition can be seen in Fig. A.11, in Appendix A.

The question to be clarified in this section is how the CBR-based Setplay selection algorithm behaves in discriminating successful Setplays from unsuccessful ones. In order to determine the departure situation, one should look at a scenario where the two available Setplays, the one with five players and the simple corner, are randomly selected. The results of execution in such a scenario are visible in Fig. 6.13. In the total number of

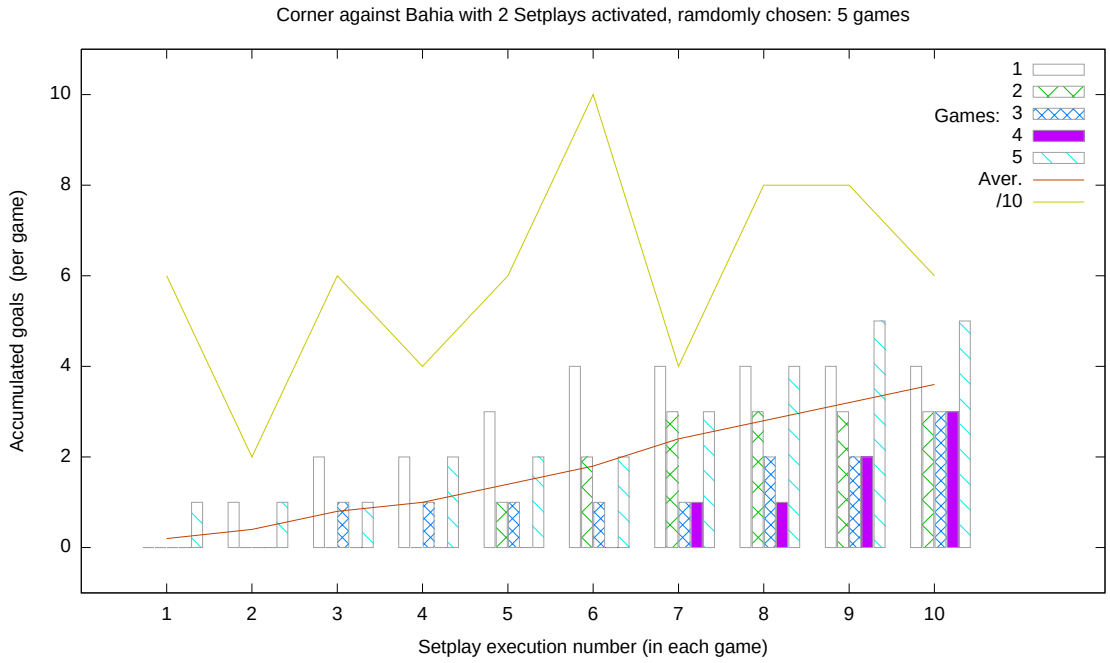


Figure 6.13: Corner with 2 random-chosen Setplays

executions, 18 goals were scored and, as seen from the figure, these accumulated number of goals grow regularly with the number of executions in each game. The line labeled as '/10' is simply a magnification by 10 of the percentage of choice of the Setplay with five players, to ease the visualization of this data. Since this line oscillates wildly around 5 in the diagram, one should recognize that the selection of each Setplay is done in roughly 50% of all executions, as expected.

In a second test, the CBR-based Setplay selection algorithm was activated, but it worked from scratch: no previous experience was considered, as the case base was initially empty. This was done for all ten games in this set of executions: the team would begin the game with an empty case base, and was able to build it as Setplays were executed. At the end of each game, the case base was retained, for later analysis, but was not considered in the subsequent games. As such, all ten games in this set depart from the same situation, with no previous experience. This set of experiences contains ten games, instead of the usual five, because Setplay choice, when there is no previous information, behaves initially quite randomly, an aspect which one wished to minimize in the analysis. The result of these executions can be seen graphically in Fig. 6.14. All figures are drawn with the same scale, to ease the comparison of results.

Looking at the diagram, one does verify that the behavior is not homogeneous, and different games show somewhat different performances. This is not surprising, as said before, due to the initial random component. If, in the first runs in a game, the Setplay

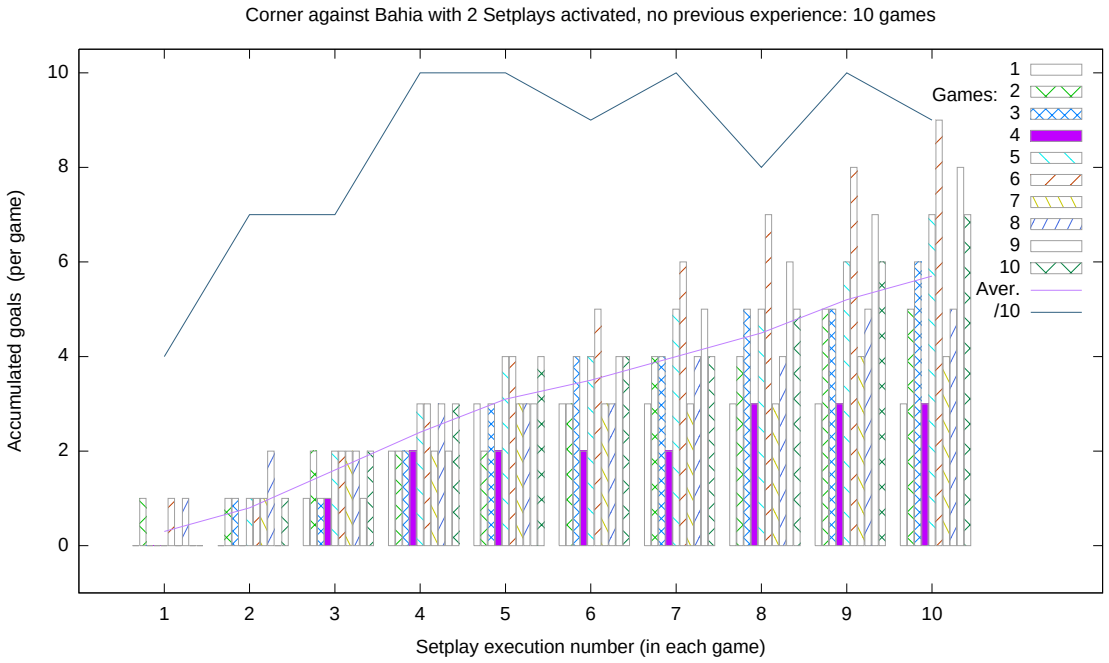


Figure 6.14: Corner with 2 Setplays: CBR choice without experience

with five players is not successful, or if the simple corner is chosen frequently, the overall result will be lower.

The total number of goals scored was 57, an average of 5.7 per game, up from 3.6 in the previous test with the random selection of Setplays, which corresponds to a 60% increase. The diagram still contains further interesting information: the percentage of choice of the more performing Setplay is shown, multiplied by ten, through the line labeled as '/10'. This figure, in the first few executions lies in the 0.4-0.7 range, but then rises abruptly to values in the 0.8-1.0 range. This is a clear proof that the CBR-based selection system performed well, prioritizing the choice of this Setplay.

The next testing scenario will explore, again, the selection of Setplays by the CBR-based system, but this time previous knowledge will be used to ground Setplay selection. Since all games in the previous test set recorded the Setplay execution results, one can simply choose one of these games as the case base for the new set of games. Since game 2 was the one closest to, but nevertheless below, the average values, its' resulting case base was chosen.

The results from these tests are presented graphically in Fig. 6.15. The total number of goals is in this case 29, which gives an average of 5.8 goals per game, a small increase relatively to the CBR-based choice system without previous experience. More interesting is that the percentage of executions of the more performing Setplay is high right from the first execution, with only minor drops, due to the intended randomness introduced

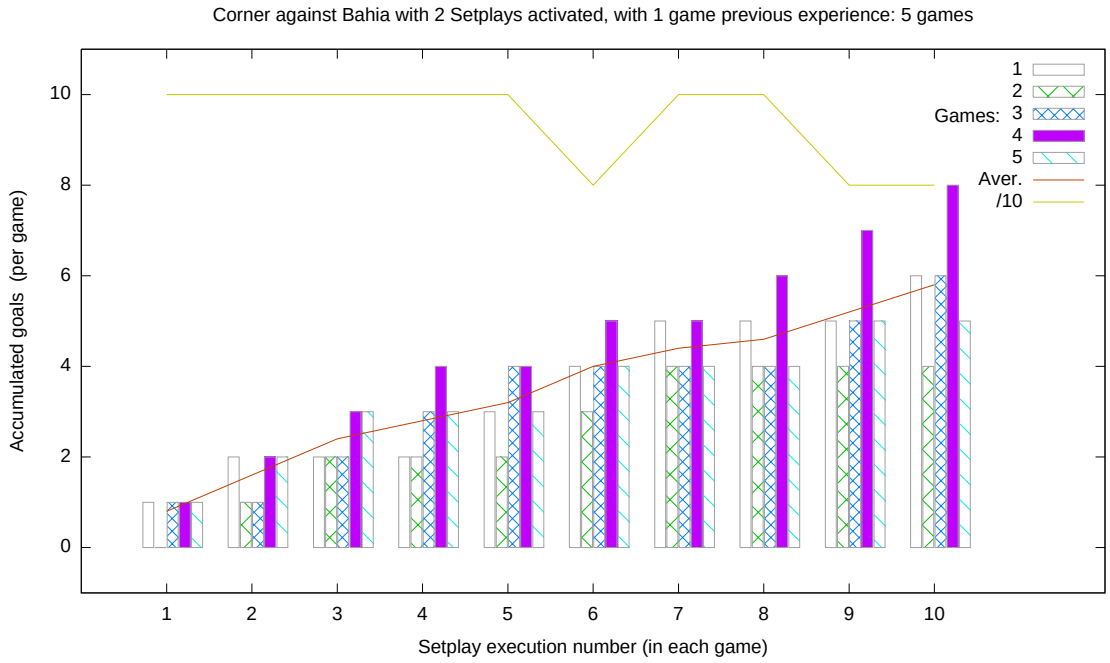


Figure 6.15: Corner with 2 Setplays: CBR choice with previous experience

in the Setplay choice algorithm. The number of goals in different games is also more homogeneous, when compared to the situation without previous experience, in Fig. 6.14. Other aspect to consider is that, on a normal game, a typical number of corner kicks will be quite small, either one or two. If one looks at the average of goals after two executions, one will have 0.8 for the situation without previous experience, and 1.6 for the one with previous experience.

Configurability After having thoroughly tested the corner kick Setplay against Bahia, several particular questions still lingered: was this the optimal Setplay for this situation? Would a small change in players positions alter considerably the performance?

In order to answer these questions, three alternative Setplays were created, by simply changing the players' positions. The general execution scheme of the Setplay remained similar, as the number of players, and the actions between them were kept identical.

In the first case, the right striker, which shoots at goal, was simply moved exactly three meters to the left, closer to the center striker. All other players' positions remained identical. The same set of tests (five games with ten corner kicks each) was run with this new Setplay. In this case, the total number of goals scored was 28, which is slightly better than the original Setplay, that managed to score 26. In this case, moving the shooter to the right did not make a considerable difference.

In the second alternative Setplay, the shooter was moved to the right, in this case two meters. All other players remained in the original locations. With this change, the total number of goals scored was 32, which means a 23% increase with respect to the original Setplay. In this case, the small change in positioning had a considerable, positive effect.

In the final alternative scenario, the whole line of strikers was moved three meters to the back, thus further away from the goal. In this case, the total number of goals scored was 19, a considerable performance degradation.

These results lead to the conclusion that minor changes in Setplay definitions might alter the performance considerably. The manual fine-tuning of Setplays has proven to be tiresome, time consuming and clearly sub-optimal, since even after having reached satisfactory results, described earlier in this section, it was still possible to enhance the performanceby simply changing players' positions. This result suggests that Setplays might benefit from the application of optimization (or machine learning) techniques that did automatically fine-tune the Setplay definition.

6.2.2.2 Nemesis

The corner kick was also considered for Setplay usage against team Nemesis, since some areas uncovered by the opponent were spotted. In this case, Setplay design was done privileging speed and swiftness.

The behavior originally displayed, without using Setplays, by FCPortugal in corner kick situations against Nemesis was quite poor. Five different games were run, and in each game ten different corner kicks were awarded. As seen in Fig. 6.16, there was only one goal scored, in a total of 50 corner kicks, a 2% success ratio.

Setplay efficacy The best performing Setplay against Nemesis is one with five participating players, and its definition can be seen in Fig. A.12, while some screenshots are shown in Fig. 6.17. Before the corned kick is actually taken, three participants gather close to the near penalty area corner, in order to attract opponents (Fig. 6.17(a)). Just before the kick is taken, one of these players (*Receiver*, nr. 8 in the image) moves to receive a pass, while other moves towards the goal (*StrikerLeft*, nr. 7), as seen on Fig. 6.17(b). Just as the '*Receiver*' gets hold of the ball, it forwards it to *StrikerLeft* (Fig. 6.17(c)). As soon as possible, *StrikerLeft* shoots at goal (Fig. 6.17(d)). The two remaining players, *StrikerCenter* and *StrikerBack* (nr. 10 and 6), take no direct action in the Setplay: they simply position themselves in front of the goal, in case the shot or *StrikerLeft*'s reception fail. If that is the case, these players are at a good position to intercept the ball and try to shoot at goal.

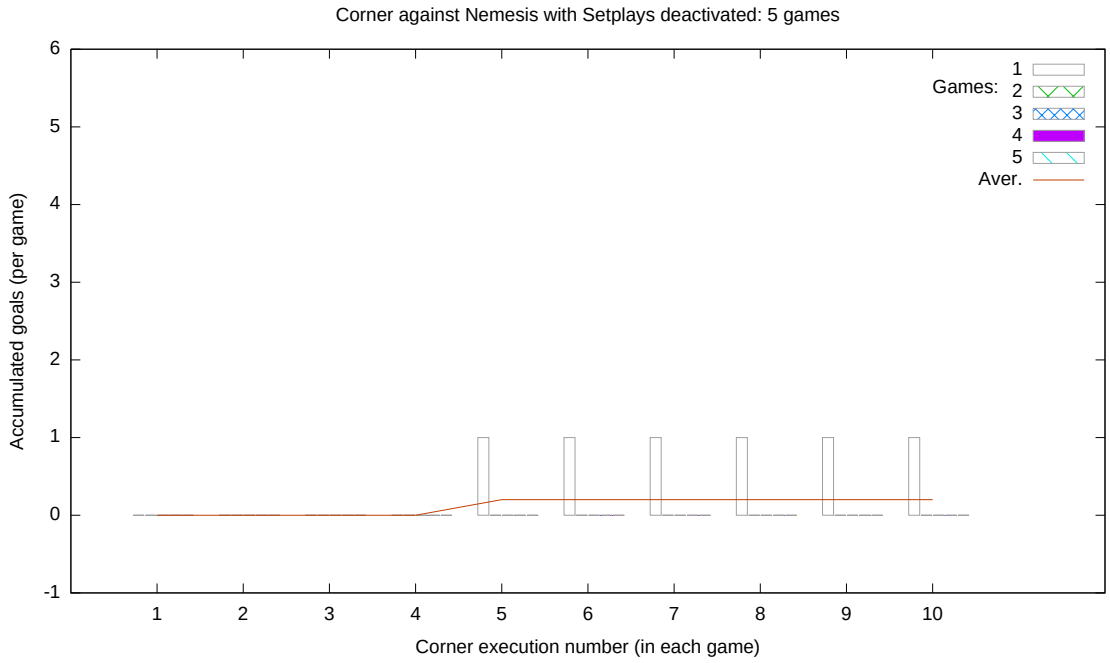
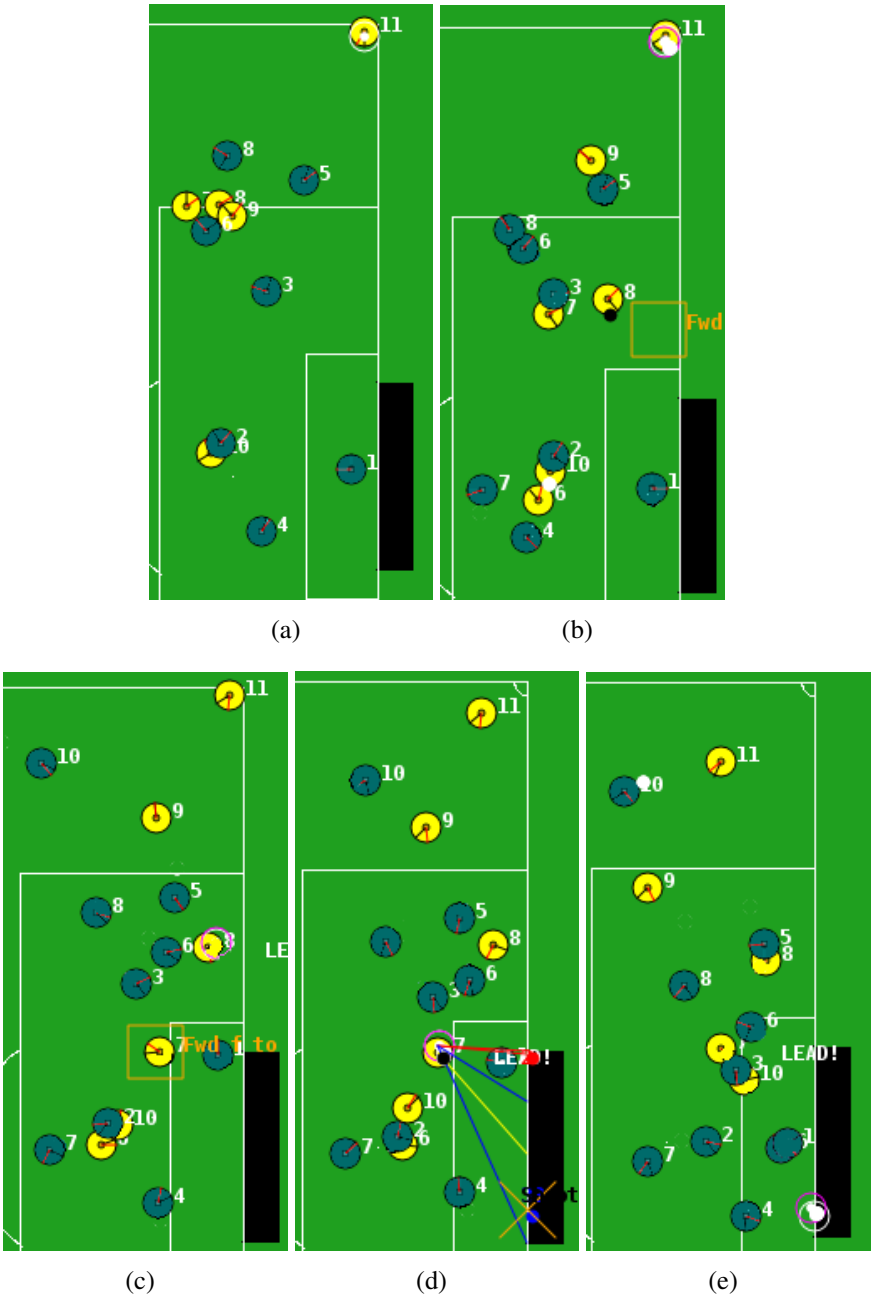


Figure 6.16: Corner with no Setplays

With this Setplay activated, the performance was quite better. For once, there were 23 goals scored, which corresponds to success in 46% of all executions, as seen in Fig. 6.18. The Setplay was successful in only 20 occasions (40%), which means that even in some situations where the Setplay was not led to its finish, players were still able to react and take advantage of the situation at hand, and scored extra goals. In the corner kick's case, one sees that the performance was dramatically enhanced by the use of Setplays.

For a comparative view of the difference of performances with and without Setplays, the data in Fig. 6.19 was compiled, with the average performances side to side. The error bars correspond to the standard deviation of accumulated goals in each execution. One can verify on the diagram that performance differs considerably, and the standard deviation bars do not overlap, similarly to the corner against team Bahia.

The t-student test defined in equation 6.1 was also applied to the data of the samples of executions of corner kicks against Nemesis, with and without usage of Setplays. In this case, the relevant values were $\overline{X_1} = 0.02$ and $\overline{X_2} = 0.46$, representing the average of goals with and without the usage of Setplays. $S_1 = 0.14$ and $S_2 = 0.50$ represent the standard deviation in each of these samples. n stands for the number of experiments in each sample, which is 50. In this case, one comes to a value of T equal to 5.95, which leads us to the conclusion that, at a significance level of 0.01, one can reject the null hypothesis, and there is statistical evidence fo affirm that the average of goals with and without Setplay usage is different.



(a) (b) (c) (d) (e)

Figure 6.17: Corner Kick with six participants: FCPortugal pictured yellow, Nemesis dark blue, ball in centre of the pink circle.

6.2.3 Kick-in

6.2.3.1 Bahia

To study the robustness of the Setplay Framework, an extra test was made: a Setplay originally designed for use with the CAMBADA middle-size team was executed in the FCPortugal 2D Simulation team. One should understand that this Setplay was designed

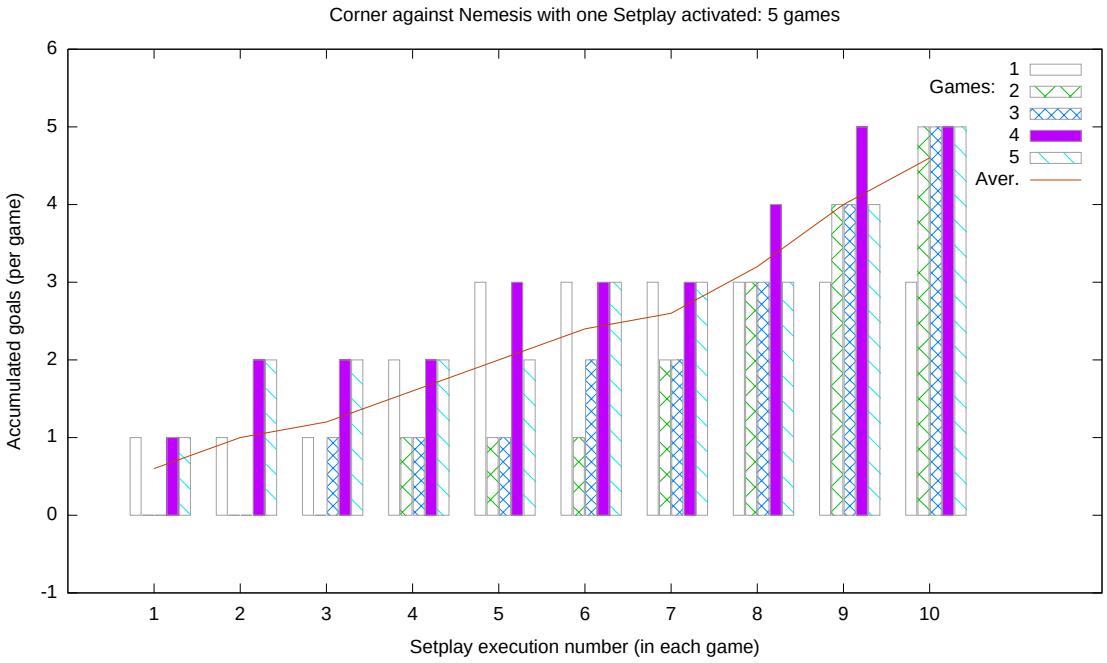


Figure 6.18: Corner with an activated Setplay

according to a typical middle-size strategy: priority is set on passing the ball securely to a team mate in a shooting position, which is just about anywhere on the opponent's half, since CAMBADA's kicking device is very powerful and frequently scores goals. With this aspect is mind, it was never expected to score goals in 2D Simulation with such a straightforward strategy: one simply aimed at verifying that the Setplay executed correctly.

Since the Setplay management, in CAMBADA, is done in all Setplays by the coach, the Setplay was subject to a minor adaptation: the lead player was changed to the player holding the ball in each Step, as usual in Setplays in this league.

This Setplay's execution is very simple: in a kick-in, the kick taker simply passes the ball to a teammate, which shoots directly at goal, no matter where he is located. It's definition can be seen in Fig. A.14, in the Appendix.

Setplay generality Ten executions were done against team Bahia, from different points on the left line, on the opponent's half. Two goals were scored, but never as direct consequence of the Setplay's shot action, since the shot was always done from a position very distant from the goal. Nevertheless, on two occasions the ball was not conveniently cleared from the opponent's penalty area, and the FCPortugal players, having already finished the Setplay, reacted adequately and managed to get hold of the ball and execute a

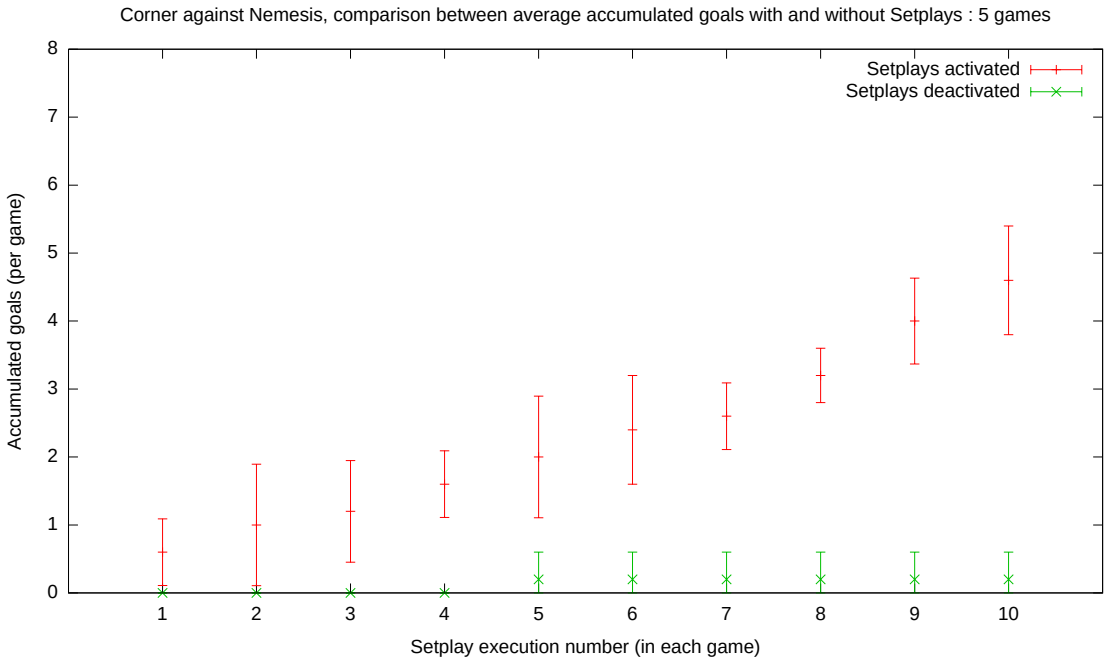


Figure 6.19: Corner against Nemesis with and without an activated Setplay

further shot at goal. In an execution very close to the opponent's goal line, one offside occurred.

These executions show that the Setplay adapted from the middle-size league was executed correctly, and even made it possible to score goals against the opponent.

6.3 Summary

This chapter presented numerous executions of Setplays, both in the FCPortugal 2D Simulation team and on the CAMBADA middle-size team. This application to such different teams shows that the Framework is very flexible, and relies on a high, yet appropriate, level of abstraction.

Setplays have proved to be effective, since the behavior exhibited by the teams running Setplays was consistently better than the original alternative, without Setplays. Evidence was also presented that Setplays are highly configurable, and that this characteristic allows the fine-tuning of the players' positions and interactions. Such changes in configurations were shown to have a considerable impact on Setplay success, and thus led to the suggestion of new research topics, described in section 7.3.

Finally, it was also shown that the CBR-based Setplay selection algorithm manages to prioritize the choice of the more performing Setplays, thus enhancing the team's performance in unknown circumstances. At the same time, this algorithm was capable of using

knowledge from previous games to select the best Setplays.

Chapter 7

Conclusions

This chapter aims at wrapping up the presentation of the work done in this thesis. The tools and results achieved are underlined, and attention is drawn to future lines of research and dissemination.

7.1 Achievements

The major outcome of the thesis is the Setplay Framework, which will be publicly available for application in any team. This will allow it to be subject of improvements and adapted to the evolution of the RoboCup domain. At this moment, the Framework provides several tools and definitions, which will be enumerated in the next paragraphs.

Setplay language. A language that defines a full vocabulary for robotic soccer and that allows the definition of multi-participant and multi-step Setplays was defined.

Graphical editor for Setplays. A graphical editor for Setplays allows the definition of the positionings and interactions of the participating robotic agents. This tool is specially suited for experts in the human soccer domain, who normally are not acquainted with common, raw computational formats. The usage of the graphical editor fosters a wider dissemination of the Setplay Framework.

Setplay parser. Setplay definitions in the textual format can be read using the dedicated parser that was developed, for usage inside or outside the Framework.

CBR-based Setplay evaluation and selection framework. The selection between possible Setplays at each moment can be performed through a CBR-based selection mechanism, where the most successful Setplays in past executions are chosen preferably.

Setplay execution engine. Setplay execution can be easily applied to an arbitrary robotic soccer team through the usage of the dedicated execution engine, which manages the progress along the Setplay and determines actions for the robots to execute.

Algorithms for the Selection of players. Some algorithms were developed to select the participating players when starting a Setplay. These were designed to minimize the robots displacements and thus speed up Setplay start.

Communication management tools. Setplay management relies on the limited exchange of coordination messages. The Framework offers tools to manage message emission and reception, which simply have to be sent when required, and fed to the Framework when received.

7.1.0.2 Results

The Framework was shown to be **effective**, through comparison of the performance in similar situations, with the usage Setplays activated and deactivated. Performance was measured to be considerably superior when Setplays were used.

The usage of the Framework was also shown to be **flexible**, as it was applied to the CAMBADA middle-size team, as well as to the 2D Simulation league and, as a prototype, to the 3D Simulation league. The application to different teams and leagues also showed the Framework's **generality**, in terms of abstraction.

Practical experiments were described in different game situations, and minor changes in Setplay definition were shown to have a considerable impact on the team's performance. Such experiments showed the Framework's **configurability** to different teams and situations.

The Framework was also shown to change the Setplay selection according to each Setplay's success history, considering data both against a specific team and against all opponents, which proves its **adaptability**.

These results show that the design, development and application of the Framework were successful procedures. This suggests that the Framework may be applied to other teams and leagues, and that the Framework may have significant impact on the performance of these teams in the future.

7.2 Publications

During the research work that led to this thesis, several papers were written and subject to peer review for publication. Their publication timeline clearly highlights the way development and research were conducted throughout the project.

Mota et al. (2006) describes a draft scenario where heterogeneous robots cooperate while playing for the same team, and discusses how communication could be used to foster teamwork. Mota and Reis (2007b) continued this line of research, with a proposal for the communication framework.

Mota and Reis (2007a) is the first sketch of the Setplay Framework, and was the basis of all development on this framework: it defined a first version of the Setplay language, but still lacked practical results and further developments such as the CBR-based Setplay selection algorithm. Mota and Reis (2008) was a development of the Setplay Framework, suggesting its inclusion in a wider model for robotic agent control.

In an expanded article (Mota et al., 2011), the application of the Framework to the 2D Simulation and Middle-size leagues was described in more detail, highlighting the differences of the experience in each. Mota et al. (2010a) and Mota et al. (2010b) described the application to the 2D Simulation team FCPortugal in more detail.

Lopes et al. (2010) describes the development of a graphical tool, which allowed the design of Setplays without actually editing the underlying text files.

While this research work was being conducted, the FCPortugal team kept participating in the international RoboCup championships, which lead to the publication of several Team Description Papers, where the different phases of the application of the Setplay Framework were described (Reis et al., 2007, 2008, 2009, 2010; Lau et al., 2011). Gimenes et al. (2007) describes the development of a Mixed Reality team, which participated in RoboCup 2007 in Atlanta, reaching the second place in this league's general rank. In this time frame, some other relevant results were reached in the 2D Simulation competition of the German Open: 2nd place in 2007 and 2011, and 3rd place in 2008 and 2009.

7.3 Future Work

The application of the Setplay Framework was taken to a competitive level, enhancing the underlying teams' performance. In spite of this, there is place for new developments, that open new research challenges.

Application to other teams. The Framework has been applied, with different levels of detail, to three different teams: CAMBADA in the middle-size league, and the FCPortugal teams in the 2D and 3D Simulation leagues. The Framework's generality could be shown even further if it was applied to still other teams and leagues. The application to other teams could take profit of these teams' strongnesses, and thus improve these teams' results.

Usage of optimization algorithms. Practical results of the Framework's application to the 2D Simulation league showed that the fine-tuning of Setplays' details could make a considerable difference in their performance. This kind of fine-tuning, when done through a human operator, is a difficult and error-prone task. Research should thus be done on the application of optimization and machine learning techniques, leading to improvements in the Setplay's performance.

Extraction of Setplays from past games. Regardless of the existence of formal intention, some collective moves by agents in any team might be a good source for new Setplays. As an example, a team might perform well when executing a corner kick. Past executions of such successful, implicit Setplays might be analysed, and treated as new Setplays.

Testing in the Middle-Size League - Real robots. After the full implementation in the simulated environment, that fully resembles the real-world setting, tests should be conducted with the real robots, to ascertain the quality of the application to this league. These tests, if successful, can lead to the integration of the Setplay framework in the CAMBADA team, in real competitions. Such an application would allow new, more advanced cooperation possibilities to the team.

Testing with Heterogeneous Real or Virtual robots. An interesting research topic is the integration of robots with different origins in the same team. With this purpose, the Setplay Framework could be used as a starting point to the creation of a mixed team, in any possible league. This is surely a research topic that has to be investigated in the future, as there is a trend to create this kind of teams. This idea is based on the fact that different robots, from different teams, might have complimentary strong and weak points. A promising strategy would be to use the strongest robots in each role or task. These heterogeneous robots would naturally need to cooperate, which could be done with the help of the Framework, concomitantly with other tools or frameworks.

Dissemination of results. Some of the results presented in this thesis are yet to be properly disseminated. Namely, the quantitative results of application to the 2D league, presented in chapter 6, were not subject to publication as a paper. Moreover, the CBR-based evaluation and selection algorithm presented in section 4.2 has also not been published with the necessary emphasis. These contributions to the RoboCup domain are certainly interesting, and are currently being prepared for submission.

7.4 Concluding remarks

The work presented in this thesis has plainly reached the goals originally drawn, with tangible results. Some aspects, like the CBR-based Setplay selection algorithm, were not considered from scratch, but were added to the Framework to maximize its utility, adding to its available set of tools, which are believed to be a consistent base for the application of Setplays in any RoboCup soccer team.

Appendix A

Setplay definitions used in tests and evaluation

```
(setplay :name ck-right-3p :id 1 :invertable true :version splanner_1.0
         :comment (Corner-kick from the right side with 3 participants)
         :players (list (playerRole :roleName Player4)
                        (playerRole :roleName Player8)
                        (playerRole :roleName Player9))
         :abortCond (or (bowner :players (player :team opp :number 0))
                  (and (not (playm play_on)) (not (playm ck_our))))
   :steps (seq
     (step :id 0 :waitTime 0 :abortTime 28
       :participants (list (at Player4 (pt :x 52.5 :y 34))
                           (at Player8 (pt :x 38.5 :y 22.5))
                           (at Player9 (pt :x 33 :y -5)))
       :condition (and (playm ck_our)
                       (bpos :region (regNamed :name right)))
       :leadPlayer Player4
       :transitions
         (nextStep :id 1
           :condition (canPassPl :from Player4 :to Player8))
           :directives (list
             (do :players Player8 :actions (intercept))
             (do :players Player9 :actions (pos :region (pt :x 33.5 :y 0.5)))
             (do :players Player4 :actions (bto :players Player8))))
     (step :id 1 :waitTime 0 :abortTime 38
       :participants (list (at Player8 (pt :x 38.5 :y 22.5))
                           (at Player9 (pt :x 33.5 :y 0.5)))
       :condition (and (playm play_on) (bowner :players Player8))
       :leadPlayer Player8
       :transitions
         (nextStep :id 2
           :directives (list
             (do :players Player8 :actions (dribble :region (pt :x 50 :y 22.5)))
             (do :players Player9 :actions (stop)))))
     (step :id 2 :waitTime 0 :abortTime 32
       :participants (list (at Player8 (pt :x 50 :y 22.5))
                           (at Player9 (pt :x 33.5 :y 0.5)))
       :condition (and (playm play_on) (bowner :players Player8))
       :leadPlayer Player8
       :transitions
         (nextStep :id 3
           :condition (canPassReg :from Player8 :to (pt :x 44 :y 5.5))
           :directives (list
             (do :players Player8 :actions (bto :region (pt :x 44 :y 5.5)))
             (do :players Player9 :actions (intercept)))))
    (step :id 3 :waitTime 0 :abortTime 24
      :participants (at Player9 (pt :x 44 :y 5.5))
      :condition (and (playm play_on) (bowner :players Player9))
      :leadPlayer (playerRole :roleName Player9)
      :transitions
        (nextStep :id 4 :condition (canShoot :players Player9)
          :directives (do :players Player9 :actions (shoot))))
    (step :id 4 :waitTime 0 :abortTime 0
      :participants (at Player4 (pt :x 52.5 :y 34))
      :condition (playm play_on) :leadPlayer Player4
      :transitions (finish))))
```

Figure A.1: Example Setplay definition of a corner-kick in SPlanner

```
(setplay :name throw-in :id 0
  :players (list (playerRole :roleName replacer)
    (playerRole :roleName receiver1) (playerRole :roleName receiver2))
  :parameters (parameter :name x_offset :type decimal)
  :steps (seq
    (step :id 0 :waitTime 1.5 :abortTime 15.0
      :participants (list (at replacer (pt ball))
        (at receiver1 (ptRel :x x_offset :y 0.5 :pt (pt ball)))
        (at receiver2 (ptRel :x 0 :y 1.5 :pt (pt ball))))
      :condition (playm ki_our) :leadPlayer (player :team our :number 6)
      :transitions (list
        (nextStep :id 1
          :condition (and (playm start)
            (canPassPl :from replacer :to receiver1)
            (ppos receiver1 (ptRel :x x_offset :y 0.5 :pt (pt ball)))
            (ppos receiver2 (ptRel :x 0 :y 1.5 :pt (pt ball))))
          :directives (list
            (do :players replacer :actions (bto :players receiver1))
            (do :players receiver1 :actions (receiveBall))))))
    (step :id 1 :waitTime 0.0 :abortTime 10.0
      :participants (list (at receiver1 (pt ball)))
      :condition (and (bowner :players receiver1) (playm play_on))
      :leadPlayer (player :team our :number 6)
      :transitions (list
        (nextStep :id 2
          :condition (canShoot :players receiver1)
          :directives (list (do :players receiver1 :actions (shoot))))
        (abort :condition (not (canShoot :players receiver1)))))
    (step :id 2 :waitTime 0 :abortTime 5
      :participants (list receiver1)
      :condition (playm ag_our) :leadPlayer (player :team our :number 6)
      :transitions (list (finish)))))
```

Figure A.2: Throw-in Setplay definition

```
(setplay :name playOnCambadaLeft :id 1
  :comment (Only works on left side because of the positioning
      of ball. Pass can only be done to shooter)
  :players (list (player :team our :number 6)
      (playerRole :roleName striker) (playerRole :roleName shooter))
  :steps (seq
    (step :id 0 :abortTime 5
      :participants (list (player :team our :number 6)
          (at striker (pt ball)) (at shooter (pt :x 0 :y 4.5)))
      :condition  (and (playm play_on)
                   (bpos :region (regNamed :name mid_left))
                   (bpos :region (regNamed :name their_back))
                   (bowner :players (player :team our :number 0)))
      :leadPlayer (player :team our :number 6)
      :transitions (list (nextStep :id 1
          :condition  (and (canPassPl :from striker :to shooter)
              (bowner :players striker))
          :directives (list (do :players striker
                   :actions (attentionTo :object shooter))))))
    (step :id 1 :abortTime 5
      :participants (list (player :team our :number 6)
          (at striker (pt ball)) (at shooter (pt :x 0 :y 4.5)))
      :condition  (and (playm play_on)
           (ppos :players shooter :region (arc :center (pt :x 0 :y 4.5)
             :radius_large 1)))
      :leadPlayer (player :team our :number 6)
      :transitions (list
        (nextStep :id 2 :condition (canPassPl :from striker :to shooter)
          :directives (list
            (do :players striker :actions (bto :players shooter))
            (do :players shooter :actions (receiveBall))))))
    (step :id 2 :abortTime 3
      :participants (list (player :team our :number 6) striker
          (at shooter (pt ball)))
      :condition (and (playm play_on) (bowner :players shooter))
      :leadPlayer (player :team our :number 6)
      :transitions (list (nextStep :id 3
          :condition (and (playm play_on) (canShoot :players shooter))
          :directives (list (do :players shooter :actions (shoot))))))
    (step :id 3 :abortTime 1
      :participants (list (player :team our :number 6) striker shooter )
      :condition (playm ag_our)
      :leadPlayer (player :team our :number 6)
      :transitions (list (finish)))))
```

Figure A.3: Middle-size league Setplay definition for play-on

```
(setplay :name goalieCatch_left_dynamic_goto_positions_fast :id 32 :invertable true
  :comment (The goalie kicks the ball in to a teammate on his left. This teammate,
    and to ones after him, keep passing the ball to the left and front)
  :players (list
    (playerRole :roleName Goalie)
    (playerRole :roleName LeftDefender)
    (playerRole :roleName LeftMidfielder)
    (playerRole :roleName LeftForward)
    (playerRole :roleName Runner))
  :abortCond (or (bowner :players (player :team opp :number 0))
    (and (not (playm gc_our)) (not (playm play_on))))
  :steps (seq
    (step :id 0 :waitTime 30 :abortTime 50
      :participants (list
        Goalie
        (at LeftDefender (pt :x -30 :y -20))
        (at LeftMidfielder (pt :x -18 :y -20))
        (at LeftForward (pt :x -0.5 :y -15))
        (at Runner (pt :x -0.5 :y -18)))
      :condition (and (playm gc_our)
                      (bpos :region (regNamed :name our_penalty_box)) )
      :leadPlayer Goalie
      :transitions (list
        (nextStep :id 1
          :directives (list
            (do :players Goalie
              :actions (pos :region (pt :x -37 :y -17)))))))
    (step :id 1 :waitTime 10 :abortTime 20
      :participants (list
        Goalie
        (at LeftDefender (pt :x -35 :y -28))
        (at LeftMidfielder (pt :x -18 :y -25))
        (at LeftForward (pt :x -0.5 :y -20))
        (at Runner (pt :x -0.5 :y -18)))
      :condition (and (playm gc_our)
                      (bpos :region (regNamed :name our_penalty_box)))
      :leadPlayer Goalie
      :transitions (list
        (nextStep :id 2
          :condition (canPassPl :from Goalie :to LeftDefender)
          :directives (list
            (do :players Goalie :actions (bto :players LeftDefender :type fast))
            (do :players LeftDefender :actions (receiveBall))
            (do :players LeftMidfielder
              :actions (pos :region (pt :x -18 :y -32)))))))))
```

Figure A.4: Setplay definition for Goalie Catch against Nemesis: four participating players, continues in A.5

```
(step :id 2 :waitTime 0 :abortTime 30
  :participants (list
    (at LeftDefender (pt :x -35 :y -28))
    (at LeftMidfielder (pt :x -18 :y -32))
    (at LeftForward (pt :x -0.5 :y -20))
    (at Runner (pt :x -0.5 :y -18)))
  :condition (and (playm play_on) (bowner :players LeftDefender))
  :leadPlayer LeftDefender
  :transitions (list
    (nextStep :id 3
      :condition (canPassPl :from LeftDefender :to LeftMidfielder)
      :directives (list
        (do :players LeftDefender
          :actions (bto :players LeftMidfielder :type fast))
        (do :players LeftMidfielder :actions (receiveBall))
        (do :players LeftForward :actions (pos :region (pt :x -0.5 :y -33)))))))
(step :id 3 :waitTime 5 :abortTime 30
  :participants (list
    (at LeftMidfielder (pt :x -18 :y -32))
    (at LeftForward (pt :x -0.5 :y -33))
    (at Runner (pt :x -0.5 :y -18)))
  :condition (and (playm play_on) (bowner :players LeftMidfielder))
  :leadPlayer LeftMidfielder
  :transitions (list
    (nextStep :id 4
      :condition (canPassPl :from LeftMidfielder :to LeftForward)
      :directives (list
        (do :players LeftMidfielder
          :actions (bto :players LeftForward :type fast))
        (do :players LeftForward :actions (receiveBall))))))
(step :id 4 :waitTime 0
  :participants (list LeftForward Runner)
  :condition (and (playm play_on) (bowner :players LeftForward))
  :leadPlayer LeftForward
  :transitions (list (finish)))))
```

Figure A.5: Setplay definition for Goalie Catch against Nemesis: four participating players, continued from Fig. A.4

```
(setplay :name goalieCatch_left_dynamic_forward_positions_6players_fast
  :id 31 :invertable true
  :comment (The goalie kicks the ball in to a teammate that backs to his left. This
    teammate, and the ones after him, keep passing the ball to the left and front)
  :players
    (list (playerRole :roleName Goalie)
      (playerRole :roleName LeftDefender)
      (playerRole :roleName LeftMidfielder)
      (playerRole :roleName LeftForward)
      (playerRole :roleName Runner)
      (playerRole :roleName Kicker))
  :abortCond (or (bowner :players (player :team opp :number 0))
    (and (not (playm gc_our)) (not (playm play_on))))
  :steps
    (seq
      (step :id 0 :waitTime 30 :abortTime 35
        :participants (list
          Goalie
          (at LeftDefender (pt :x -30 :y -20))
          (at LeftMidfielder (pt :x -28 :y -20))
          (at LeftForward (pt :x -15 :y -15))
          (at Kicker (pt :x -0.5 :y -18))
          (at Runner (pt :x -4 :y -15)))
        :condition
         (and (playm gc_our) (bpos :region (regNamed :name our_penalty_box)))
        :leadPlayer Goalie
        :transitions (list
          (nextStep :id 1
            :directives (list
              (do :players Goalie
                :actions (pos :region (pt :x -47 :y -17)))
              (do :players LeftDefender
                :actions (pos :region (pt :x -40 :y -24)))))))
      (step :id 1 :waitTime 10 :abortTime 20
        :participants (list
          Goalie
          (at LeftDefender (pt :x -40 :y -24))
          (at LeftMidfielder (pt :x -30 :y -30))
          (at LeftForward (pt :x -15 :y -25))
          (at Kicker (pt :x -0.5 :y -28))
          (at Runner (pt :x -1 :y -25)))
        :condition
          (and (playm gc_our) (bpos :region (regNamed :name our_penalty_box)))
        :leadPlayer Goalie
        :transitions (list
          (nextStep :id 2
            :directives (list
              (do :players Goalie
                :actions (bto :region (pt :x -40 :y -24) :type normal))
              (do :players LeftDefender :actions (intercept))
              (do :players LeftMidfielder
                :actions (pos :region (pt :x -30 :y -30)))))))
```

Figure A.6: Setplay definition for Goalie Catch against Nemesis: six participating players, continues in Fig. A.7

```
(step :id 2 :waitTime 0 :abortTime 30
  :participants (list
    (at LeftDefender (pt :x -40 :y -24))
    (at LeftMidfielder (pt :x -30 :y -30))
    (at LeftForward (pt :x -15 :y -30))
    (at Kicker (pt :x -0.5 :y -30))
    (at Runner (pt :x -1 :y -25)))
  :condition (and (playm play_on) (bowner :players LeftDefender))
  :leadPlayer LeftDefender
  :transitions (list
    (nextStep :id 3
      :directives (list
        (do :players LeftDefender
          :actions (bto :region (pt :x -30 :y -32) :type fast))
        (do :players LeftMidfielder :actions (intercept))
        (do :players LeftForward
          :actions (pos :region (pt :x -15 :y -32)))))))
(step :id 3 :waitTime 0 :abortTime 30
  :participants (list
    (at LeftMidfielder (pt :x -30 :y -30))
    (at LeftForward (pt :x -15 :y -32))
    (at Kicker (pt :x -0.5 :y -32))
    (at Runner (pt :x -1 :y -25)))
  :condition (and (playm play_on) (bowner :players LeftMidfielder))
  :leadPlayer LeftMidfielder
  :transitions (list
    (nextStep :id 4
      :directives (list
        (do :players LeftMidfielder
          :actions (bto :region (pt :x -15 :y -32) :type fast))
        (do :players LeftForward :actions (intercept))
        (do :players Kicker
          :actions (pos :region (pt :x -0.5 :y -32)))))))
(step :id 4 :waitTime 0 :abortTime 30
  :participants (list
    (at LeftForward (pt :x -15 :y -32))
    (at Kicker (pt :x -0.5 :y -31.5))
    (at Runner (pt :x -0.5 :y -25)))
  :condition (and (playm play_on) (bowner :players LeftForward))
  :leadPlayer LeftForward
  :transitions (list
    (nextStep :id 5
      :directives (list
        (do :players LeftForward
          :actions (bto :region (pt :x -0.5 :y -31) :type fast))
        (do :players Kicker :actions (receiveBall))))))
```

Figure A.7: Setplay definition for Goalie Catch against Nemesis: six participating players, continued from Fig. A.6, continues in Fig. A.8

```
(step :id 5 :waitTime 0 :abortTime 70
  :participants (list
    (at Kicker (pt :x -0.5 :y -31))
    (at Runner (pt :x -0.5 :y -25)))
  :condition (and (playm play_on) (bowner :players Kicker))
  :leadPlayer Kicker
  :transitions (list
    (nextStep :id 6
      :directives (list
        (do :players Kicker
          :actions (dribble :region (pt :x 50 :y -30.5)))
        (do :players Runner :actions (moveToOffSideLine :y -23))))))
(step :id 6 :waitTime 0 :abortTime 30
  :participants (list
    (at Kicker (pt :x 50 :y -31))
     Runner)
  :condition (and (playm play_on) (bowner :players Kicker)
    (or (nearOffsideLine :players Runner)
    (ppos :players (player :team opp :number 0)
      :region (arc :center (pt ball) :radius_large 3))))
  :leadPlayer Kicker
  :transitions (list
    (nextStep :id 7
      :condition
        (and (canPassReg :from Kicker
               :to (ptRel :x 3 :y 0 :pt (pt :player Runner)))
          (bpos :region (regNamed :name their_back)))
      :directives (list
        (do :players Kicker
          :actions (bto :region (ptRel :x 5 :y -2
                                  :pt (pt :player Runner))
                        :type slow))
        (do :players Runner :actions (intercept))))
   (nextStep :id 7
     :condition
       (and (canPassReg :from Kicker
      :to (ptRel :x 8 :y 0 :pt (pt :player Runner)))
          (not (bpos :region (regNamed :name their_back))))
     :directives (list
       (do :players Kicker
         :actions (bto :region (ptRel :x 12 :y -2
                                 :pt (pt :player Runner)) :type slow))
       (do :players Runner :actions (intercept))))
   (finish
     :condition
       (not (canPassReg :from Kicker
              :to (ptRel :x 3 :y 0 :pt (pt :player Runner)))))))
(step :id 7 :waitTime 0
  :participants (list Runner)
  :condition (and (playm play_on) (bowner :players Runner))
  :leadPlayer Runner
  :transitions (list (finish))))
```

Figure A.8: Setplay definition for Goalie Catch against Nemesis: six participating players, continued from Fig. A.7

```
(setplay :name cornerKickParaBahiaLMmotaFastLinhaSemCondShoot
  :id 1 :invertable true
  :version splanner_1.0
  :players (list
    (playerRole :roleName Player6)
    (playerRole :roleName Player8)
    (playerRole :roleName Player9)
    (playerRole :roleName Player10)
    (playerRole :roleName Player11))
  :abortCond (or (bowner :players  (player :team opp :number 0))
    (and (not (playm ag_our)) (not (playm play_on)) (not (playm ck_our))))
   :steps (seq
    (step :id 0 :waitTime 10 :abortTime 35
      :participants (list
        (at (playerRole :roleName Player6) (pt :x 52.5 :y -34))
        (at (playerRole :roleName Player8) (pt :x 36 :y 0.5))
        (at (playerRole :roleName Player9) (pt :x 36.5 :y -9))
        (at (playerRole :roleName Player10) (pt :x 36 :y -27.5))
        (at (playerRole :roleName Player11) (pt :x 49.5 :y -21.5)))
      :condition (and (playm ck_our) (bpos :region (regNamed :name left)))
      :leadPlayer (playerRole :roleName Player6)
      :transitions (list
        (nextStep :id 1
          :condition (canPassPl :from  (list (playerRole :roleName Player6))
             :to (list (playerRole :roleName Player11)))
          :directives (list
            (do :players (list (playerRole :roleName Player6))
               :actions (list (bto :players (playerRole :roleName Player11)
                 :type fast)))
           (do :players (list (playerRole :roleName Player11))
              :actions (list (receiveBall)))))))
    (step :id 1 :waitTime 1 :abortTime 26
       :participants (list
         (at (playerRole :roleName Player10) (pt :x 36 :y -27.5))
         (at (playerRole :roleName Player11) (pt :x 49.5 :y -21.5))
         (at (playerRole :roleName Player8) (pt :x 36 :y 0.5))
         (at (playerRole :roleName Player9) (pt :x 36 :y -9)))
       :condition (and (playm play_on)
         (bowner :players (playerRole :roleName Player11)))
       :leadPlayer (playerRole :roleName Player11)
       :transitions (list
         (nextStep :id 2
           :condition (canPassPl :from  (list (playerRole :roleName Player11))
              :to (list (playerRole :roleName Player10)))
           :directives (list
             (do :players (list (playerRole :roleName Player10))
                :actions (list (receiveBall) ))
             (do :players (list (playerRole :roleName Player11))
               :actions (bto :players (playerRole :roleName Player10)
                  :type fast))))))
```

Figure A.9: Setplay definition for corner kick against Bahia, continued in Fig. A.10

```
(step :id 2 :waitTime 1 :abortTime 28
  :participants (list
    (at (playerRole :roleName Player9) (pt :x 36 :y -9))
    (at (playerRole :roleName Player10) (pt :x 36 :y -27.5))
    (at (playerRole :roleName Player11) (pt :x 49.5 :y -21.5))
    (at (playerRole :roleName Player8) (pt :x 36 :y 0.5)))
  :condition (and (playm play_on)
    (bowner :players (playerRole :roleName Player10)))
  :leadPlayer (playerRole :roleName Player10)
  :transitions (list
     (nextStep :id 3
       :condition (canPassPl :from  (list (playerRole :roleName Player10))
         :to (list (playerRole :roleName Player9) ))
       :directives (list
         (do :players (playerRole :roleName Player9) :actions (receiveBall))
         (do :players (list (playerRole :roleName Player10))
            :actions (bto :players (playerRole :roleName Player9) :type fast))
         (do :players (list (playerRole :roleName Player11))
            :actions (list (moveToOffSideLine :y -15)))))))
(step :id 3 :waitTime 0 :abortTime 28
   :participants (list
     (at (playerRole :roleName Player9) (pt :x 36 :y -9))
     (at (playerRole :roleName Player10) (pt :x 36 :y -27.5))
     (at (playerRole :roleName Player11) (pt :x 49.5 :y -21.5))
     (at (playerRole :roleName Player8) (pt :x 36 :y 0.5)))
   :condition (and (playm play_on)
     (bowner :players  (list (playerRole :roleName Player9))))
   :leadPlayer (playerRole :roleName Player9)
   :transitions (list
     (nextStep :id 4
       :condition (canPassPl :from  (list (playerRole :roleName Player9))
         :to (list (playerRole :roleName Player8)))
       :directives (list
         (do :players (playerRole :roleName Player8) :actions (receiveBall))
         (do :players (playerRole :roleName Player9)
            :actions (bto :players (playerRole :roleName Player8) :type fast))))))
(step :id 4 :waitTime 0 :abortTime 15
  :participants (list Player8)
  :condition (and (playm play_on)
    (bowner :players (list (playerRole :roleName Player8))))
  :leadPlayer (playerRole :roleName Player8)
  :transitions (list
    (nextStep :id 5
     :directives (list
        (do :players (playerRole :roleName Player8) :actions (shoot))))))
(step :id 5 :waitTime 0 :abortTime 0
  :participants (list
     (playerRole :roleName Player8))
  :condition (playm ag_our)
  :leadPlayer (playerRole :roleName Player8)
  :transitions (list  (finish :directives (list))))))
```

Figure A.10: Setplay definition for corner kick against Bahia, continued from Fig. A.9

```
(setplay :name cornerKickMuitoSimples2 :id 2 :invertable true
  :version splanner_1.0
  :players (list
    (playerRole :roleName Player6)
    (playerRole :roleName Player8))
  :abortCond (or (bowner :players  (player :team opp :number 0))
    (and (not (playm play_on)) (not (playm ck_our)) (not (playm ag_our))))
  :steps (seq
    (step :id 0 :waitTime 3 :abortTime 30
      :participants (list
        (at (playerRole :roleName Player6) (pt :x 52.5 :y -34))
        (at (playerRole :roleName Player8) (pt :x 46 :y 0)))
      :condition (and (playm ck_our) (bpos :region (regNamed :name left)))
      :leadPlayer (playerRole :roleName Player6)
      :transitions (list
        (nextStep :id 1
          :directives (list
           (do :players (list (playerRole :roleName Player6) )
              :actions (list (bto :region (pt :x 46 :y 0) :type normal)))
           (do :players (list (playerRole :roleName Player8))
             :actions (list (receiveBall)))))))
    (step :id 1 :waitTime 1 :abortTime 23
      :participants (list
         (at (playerRole :roleName Player8) (pt :x 46 :y 0)))
      :condition (and (playm play_on)
         (bowner :players  (list (playerRole :roleName Player8))))
      :leadPlayer (playerRole :roleName Player8)
      :transitions (list
        (nextStep :id 2
          :condition (canShoot :players  (list (playerRole :roleName Player8) ))
          :directives (list
           (do :players (list (playerRole :roleName Player8)) :actions
(list (shoot)))))))
    (step :id 2 :waitTime 0 :abortTime 0
      :participants (list
         (at (playerRole :roleName Player8) (pt :x 46 :y 0)))
      :condition (playm ag_our)
      :leadPlayer (playerRole :roleName Player8)
      :transitions (list (finish :directives (list))))))
```

Figure A.11: Setplay definition for simple corner kick against Bahia

```
(setplay :name ck2aLMota :id 52 :invertable true
  :comment (Receiver and StrikerLeft cross start running
     in different directions from near the penalty box corner.
     Receiver receives the ball next to the nearest goal post
     and makes a straight pass to the middle of the penalty
     box to StrikerLeft, StrikerCenter or StrikerBack to score.)
  :players (list
    (playerRole :roleName Taker)
    (playerRole :roleName Receiver)
    (playerRole :roleName BackReceiver)
    (playerRole :roleName StrikerLeft)
    (playerRole :roleName StrikerCenter)
    (playerRole :roleName StrikerBack))
  :abortCond (or (bowner :players (player :team opp :number 0))
    (and (not (playm ck_our)) (not (playm play_on))
      (not (playm ag_our))))
  :steps (seq
    (step :id 0 :waitTime 16 :abortTime 35
      :participants (list Taker
        (at BackReceiver (pt :x 42 :y -20))
         (at Receiver (pt :x 41 :y -20))
         (at StrikerLeft (pt :x 38 :y -20))
         (at StrikerCenter (pt :x 37 :y 0))
         (at StrikerBack (pt :x 34 :y 0)))
      :condition
        (and (playm ck_our) (bpos :region (regNamed :name our_left)))
      :leadPlayer Taker
      :transitions (list
         (nextStep :id 1
          :directives (list (do :players Taker :actions (intercept))))))
    (step :id 1 :waitTime 13 :abortTime 25
      :participants (list Taker
         (at BackReceiver (pt :x 46 :y -24))
         (at Receiver (pt :x 51 :y -12))
         (at StrikerLeft (pt :x 46 :y -7))
         (at StrikerCenter (pt :x 43 :y -2))
         (at StrikerBack (pt :x 42:y 0)))
      :condition (and (or (playm ck_our) (playm ki_our)))
      :leadPlayer Taker
      :transitions (list
         (nextStep :id 2
          :directives (list
           (do :players Taker :actions (bto :region (pt :x 51 :y -12)
              :type fast))
           (do :players Receiver :actions (receiveBall))))))
```

Figure A.12: Setplay definition for Corner against Nemesis: six participating players, continues in Fig. A.13

```
(step :id 2 :waitTime 0 :abortTime 10
  :participants (list Receiver
     (at StrikerLeft (pt :x 46 :y -7))
     (at StrikerCenter (pt :x 47.5 :y -5))
     (at StrikerBack (pt :x 47.5 :y 0)))
  :condition (and (bowner :players Receiver) (playm play_on))
  :leadPlayer Receiver
  :transitions (list
     (nextStep :id 3
      :directives (list
       (do :players Receiver :actions (bto :region (pt :x 46 :y -7) :type fast))
       (do :players StrikerLeft :actions (receiveBall))))))
(step :id 3 :waitTime 0 :abortTime 8
  :participants (list StrikerLeft (at StrikerCenter (pt :x 50 :y -5))
     (at StrikerBack (pt :x 50 :y 0)))
  :condition (and (playm play_on)
     (bowner :players (list StrikerLeft StrikerCenter StrikerBack)))
  :leadPlayer StrikerLeft
  :transitions (list
     (nextStep :id 4
      :condition (canShoot :players (list StrikerLeft))
      :directives (list
       (do :players StrikerLeft :actions (shoot))))))
(step :id 4 :waitTime 0 :abortTime 0
  :participants (list StrikerLeft StrikerBack StrikerCenter)
  :condition (or (playm ag_our)
    (and (playm play_on) (bowner :players (list StrikerCenter StrikerBack))))
  :leadPlayer StrikerLeft
  :transitions (list (finish))))
```

Figure A.13: Setplay definition for Corner against Nemesis: six participating players, continued from A.12

```
(setplay :name kickInCambadaLeft :id 1 :invertable true
  :comment (Only works on left side because of the positioning of players.
     At this point without aligning with the goal...)
  :players (list (playerRole :roleName replacer) (playerRole :roleName receiver)
    (playerRole :roleName supporter))
  :abortCond (or (bowner :players (player :team opp :number 0))
    (and (not (playm ki_our)) (not (playm play_on)) (not (playm ag_our))))
  :steps (seq
    (step :id 0
      :participants (list
        (at replacer (ptRel :x 0 :y 0.6 :pt (pt ball)))
        (at receiver (ptRel :x 0 :y -1 :pt (pt ball)))
        (at supporter (ptRel :x 1 :y 0 :pt (pt ball))) )
      :condition  (playm ki_our)
      :leadPlayer replacer
      :transitions (list
         (nextStep :id 1
          :condition (and (playm ki_our) (canPassPl :from replacer :to receiver))
          :directives (list
             (do :players replacer :actions (bto :players receiver))
             (do :players receiver :actions (receiveBall))))))
    (step :id 1
      :participants (list
         (at replacer (ptRel :x 0 :y 4 :pt (pt ball)))
         (at receiver (pt ball))
         (at supporter (ptRel :x 1 :y 0 :pt (pt ball))))
      :condition (and (playm play_on) (bowner :players receiver))
      :leadPlayer receiver
      :transitions (list
         (nextStep :id 2
          :condition (playm play_on)
          :directives (list (do :players receiver :actions (shoot))))))
    (step :id 2
      :participants (list receiver)
      :condition (playm ag_our)
      :leadPlayer receiver
      :transitions (list (finish))))
)
```

Figure A.14: Setplay definition for kick-in from CAMBADA

References

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, March 1994.

Wouter H. T. M. Aangenent, Jeroen J. T. H. de Best, B. H. M. Bukkems, F. M. W. Kanters, K. J. Meessen, J. J. P. A. Willems, R. J. E. Merry, and M. J. G. v.d. Molengraft. TechUnited Eindhoven Team Description 2009. In Baltes et al. (2009).

Hidehisa Akiyama and Itsuki Noda. Multi-Agent Positioning Mechanism in the Dynamic Environment. In Visser et al. (2008), pages 377–384.

Minoru Asada and Hiroaki Kitano, editors. *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Artificial Intelligence*, 1999. Springer.

John Atkinson and Dario Rojas. Generating dynamic formation strategies based on human experience and game conditions. In Visser et al. (2008), pages 159–170.

Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, and Saeed Shiry, editors. *RoboCup 2009: Robot Soccer World Cup XIII, CD proceedings*, 2009.

Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, and Saeed Shiry, editors. *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *Lecture Notes in Computer Science*, 2010. Springer.

Julien Beaudry, Julian Choquette, Pierre-Marc Fournier, Louis-Alain Larouche, and François Savard. Robofoot ÉPM team description – RoboCup2006 MiddleSize League. In Lakemeyer et al. (2006).

Sven Behnke and Marcell Missura. Nimbro teensize 2011 team description. In Röfer et al. (2011b).

Ralf Berger and Gregor Lämmel. Exploiting past experience – case-based decision support for soccer agents. In *KI 2007: Advances in Artificial Intelligence*, 2007.

Ralf Berger, Michael Gollin, and Hans-Dieter Burkhard. AT Humboldt 2004 & AT Humboldt 3D team description. In Nardi et al. (2004).

Ralf Berger, Daniel Hein, and Hans-Dieter Burkhard. AT Humboldt & AT Humboldt 3d team description 2006. In Lakemeyer et al. (2006).

Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors. *RoboCup-2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, 2002. Springer.

Joschka Boedecker and Minoru Asada. Simspark - concepts and application in the RoboCup 3D Soccer Simulation League. In *SIMPAR-2008 Workshop on The Universe of RoboCup Simulators*, Venice, Italy, 2008.

Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 376–383, 2004.

Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors. *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Artificial Intelligence*, 2006. Springer.

Brett Browning, James Bruce, Michael Bowling, and Manuela Veloso. Stp: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering*, 219:2005, 2004.

Sean Buttinger, Marco Diedrich, Leo Hennig, Angelika Hoenemann, Philipp Huegelmeyer, Andreas Nie, Andres Pegam, Collin Rogowski, Claus Rollinger, Timo Steffens, and Wilfried Teiken. The Dirty Dozen Team and Coach Description. In Birk et al. (2002), pages 43–52.

Claudio Castelpietra, Luca Iocchi, Daniele Nardi, Maurizio Piaggio, A. Scalzo, and Antonio Sgorbissa. Communication and coordination among heterogeneous mid-size players: Art99. In Stone et al. (2001).

Claudio Castelpietra, A. Guidotti, Luca Iocchi, Daniele Nardi, and Riccardo Rosati. Design and implementation of cognitive soccer robots. In Birk et al. (2002), pages 86–95.

Mao Chen, Ehsan Foroughi, Fredrik Heintz, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, Yi Wang, and Xiang Yin. *Users manual: RoboCup soccer server manual for soccer server version 7.07 and later*, 2003. URL `http://sourceforge.net/projects/sserver/`.

Philip R. Cohen and Hector Levesque. Teamwork. Technical report, Center for the Study of Language and Information SRI International, 1991.

André Scolari Conceição, A. Paulo Moreira, Luís Paulo Reis, and Paulo J. Costa. Architecture of cooperation for multi-robot systems. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *First IFAC Workshop on Multivehicle Systems (MVS'06)*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 458–469. Springer, Berlin, Heidelberg, New York, 2006.

João Guilherme Bettencourt Cravo. SPlanner: Uma aplicação gráfica de definição flexível de jogadas estudadas no RoboCup. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2011.

Rodrigo da Silva Guerra, Joschka Boedecker, Norbert Mayer, Shinzo Yanagimachi, Yasuji Hirosawa, Kazuhiko Yoshikawa, Masaaki Namekawa, and Minoru Asada. Introducing physical visualization sub-league introducing physical visualization sub-league. In Visser et al. (2008), pages 496–503.

HesamAddin Torabi Dashti, Nima Aghaeepour, Sahar Asadi, Meysam Bastani, Zahra Delafkar, Fatemeh Miri Disfani, Serveh Mam Ghaderi, Shahin Kamali, Sepideh Pashami, and Alireza Fotuhi Siahpirani. Dynamic Positioning based on Voronoi Cells (DPVC). In Bredenfeld et al. (2006), pages 219–229.

John Davin, Patrick Riley, and Manuela Veloso. CommLang: Communication for coachabe agents. In Nardi et al. (2005), pages 46–59.

Jeroen de Best, René van de Molengraft, and Maarten Steinbuch. A novel ball handling mechanism for the robocup middle size league. *Mechatronics*, 21(2):469–478, 2011.

Joel de Guzman and Dan Nuffer. The spirit library: Inline parsing in c++. *C/C++ Users Journal*, 21(9):22, September 2003.

Javier Ruiz del Solar, Eric Chown, and Paul Ploeger, editors. *RoboCup 2010: Robot Soccer World Cup XIV, CD proceedings*, 2010.

Javier Ruiz del Solar, Eric Chown, and Paul Ploeger, editors. *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, 2011. Springer.

Alessandro Farinelli, Giorgio Grisetti, and Luca Iocchi. SPQR-RDK: a modular framework for programming mobile robots. In Nardi et al. (2005), pages 653–660.

Alexander Ferrein, Gerhard Lakemeyer, and Stefan Schiffer. Allemaniacs 2006 team description. In Lakemeyer et al. (2006).

M. Friedmann, J. Kuhn, S. Kohlbrecher, K. Petersen, D. Scholz, D. Thomas, J. Wojtusch, and O. von Stryk. Darmstadt dribblers team description for humanoid kidsize league of robocup 2011. In Röfer et al. (2011b).

Ulrich Furbach, Jan Murray, Falk Schmidsberger, and Frieder Stolzenburg. Model checking hybrid multiagent-systems for the RoboCup. In Visser et al. (2008), pages 262–269.

Thomas Gabel and Martin Riedmiller. On progress in robocup: The simulation league showcase. In del Solar et al. (2011).

Brian P. Gerkey and Maja J. Mataric. On role allocation in RoboCup. In Polani et al. (2004), pages 43–53.

Ricardo Gimenes, Luís Mota, Luís Paulo Reis, Nuno Lau, and João Certo. Simulation meets reality: A cooperative approach to robocup's physical visualization soccer league. In José Maia Neves, Manuel Filipe Santos, and José Manuel Machado, editors, *13th Portuguese Conference on Artificial Intelligence, EPIA 2007*, 2007.

Roland Hafner and Martin Riedmiller. Reinforcement learning on an omnidirectional mobile robot. In *IEEE/RSJ IROS 2003 Workshop on Robotics for Nanosciences and Nanotechnology*, 2003.

Patrick Heinemann, Juergen Haase, and Andreas Zell. A novel approach to efficient monte-carlo localization in robocup. In Lakemeyer et al. (2007), pages 322–329.

Patrick Heinemann, Frank Sehnke, Felix Streichert, and Andreas Zell. Towards a calibration-free robot: The act algorithm for automatic online color training. In Lakemeyer et al. (2007), pages 322–329.

Luca Iocchi and Daniele Nardi. Self-localization in the robocup environment. In Veloso et al. (2000), pages 227–242.

Luca Iocchi, L. Marchetti, Daniele Nardi, Pedro Lima, Marco Barbosa, Hugo Pereira, and Nuno Lopes. SPQR + ISocRob RoboCup 2007 qualification report. Technical report, Sapienza Universitá di Roma, Italy; Instituto Superior Técnico, 2007.

Luca Iocchi, Hitoshi Matsubara, Alfredo Weitzenfeld, and Changjiu Zhou, editors. *RoboCup 2008: Robot Soccer World Cup XII, CD proceedings*, 2008.

Luca Iocchi, Hitoshi Matsubara, Alfredo Weitzenfeld, and Changjiu Zhou, editors. *RoboCup 2008: Robot Soccer World Cup XII*, volume 5399 of *Lecture Notes in Artificial Intelligence*, 2009. Springer.

Gal Kaminka, Pedro Lima, and Raúl Rojas, editors. *RoboCup-2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Artificial Intelligence*, 2003. Springer.

Alankar Karol, Bernhard Nebel, Christopher Stanton, and Mary-Anne Williams. Case based game play in the RoboCup four-legged league part i the theoretical model. In Polani et al. (2004), pages 739–747.

Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *Lecture Notes in Artificial Intelligence*, 1998. Springer.

Alexander Kleiner and Thorsten Buchheim. A plugin-based architecture for simulation in the F2000 league. In Polani et al. (2004).

Marco Kögler and Oliver Obst. Simulation league: The next generation. In Polani et al. (2004).

Jelle R. Kok and Nikos Vlassis. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *Journal of Machine Learning Research (JMLR)*, 7:1789–1828, 2006.

Jelle R. Kok, Matthijs Spaan, and Nikos Vlassis. Multi-robot decision making using coordination graphs. In *11th International Conference on Advanced Robotics (ICAR 2003)*, pages 1124–1129, 2003.

Jelle R. Kok, Matthijs T. J. Spaan, and Nikos Vlassis. Non-Communicative Multi-Robot Coordination in Dynamic Environments. *Robotics and Autonomous Systems*, 50(2-3): 99–114, 2005.

Hatice Kose, Kemal Kaplan, Cetin Mericliand Utku Tatlidede, and Levent Akin. Market-driven multi-agent collaboration in robot soccer domain. In *Cutting Edge Robotics*, pages 407–416. pIV pro literatur Verlag, 2005.

Vadim Kyrylov and Eddie Hou. While the Ball in the Digital Soccer is Rolling, Where the Non-Player Characters Should go in a Defensive Situation? In Bill Kapralos, Mike Katchabaw, and Jay Rajnovich, editors, *Proceedings of the Conference on Future Play*, pages 90–96, Toronto, Canada, 2007. ACM.

Vadim Kyrylov and Eddie Hou. Pareto-Optimal Collaborative Defensive Player Positioning in Simulated Soccer. In Baltes et al. (2010).

Vadim Kyrylov and Serguei Razykov. Pareto-Optimal Offensive Player Positioning in Simulated Soccer. In Visser et al. (2008), pages 228–237.

R. Lafrenz, O. Zweigle, U.-P. Käppeler, H. Rajaie, A. Tamke, T. Ruhr, M. Oubbati, M. Schanz, F. Schreiber, and P. Lev. Major scientific achievements 2006 - CoPS Stuttgart registering for world championships in Bremen. Technical report, University of Stuttgart, 2006.

Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors. *RoboCup-2006: Robot Soccer World Cup X, CD proceedings*, 2006.

Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors. *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, 2007. Springer.

Sascha Lange and Martin Riedmiller. Evolution of computer vision subsystems in robot navigation and image classification tasks. In Nardi et al. (2005), pages 184–195.

Sascha Lange, Christian Müller, and Stefan Welker. *Tribots: Soccer Strategy*. RoboCup Workshop Kassel 2008, http://www.ni.uos.de/fileadmin/user_upload/tribots/Research/Kooperation.pdf, 2008. Last visited: 21 of July 2009.

Valerio Lattarulo and Sander G. van Dijk. Application of the "alliance algorithm" to energy constrained gait optimization. In Röfer et al. (2011b).

Andreas D. Lattner, Andrea Miene, Ubbo Visser, and Otthein Herzog. Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In Bredenfeld et al. (2006), pages 118–129.

Nuno Lau and Luís Paulo Reis. FC Portugal 2001 team description: Configurable strategy and flexible teamwork. In Birk et al. (2002), pages 1–10.

Nuno Lau and Luís Paulo Reis. Coordination methodologies developed for FC Portugal 3D 2006 team. In Lakemeyer et al. (2006).

Nuno Lau and Luis Paulo Reis. *FC Portugal - High-level Coordination Methodologies in Soccer Robotics*, pages 167–192. Itech Education and Publishing, Vienna, Austria, 2007.

Nuno Lau, Luis Seabra Lopes, and Gustavo Corrente. CAMBADA: Information sharing and team coordination. In *Eighth Conference on Autonomous Robot Systems and Competitions*, pages 27–32, Aveiro, Portugal, 2008. Universidade de Aveiro.

Nuno Lau, Luis Seabra Lopes, Gustavo Corrente, and Nelson Filipe. Multi-robot team coordination through roles, positionings and coordinated procedures. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems - IROS 2009,*, St. Louis, USA, October 2009.

Nuno Lau, Luis Seabra Lopes, Gustavo Corrente, Nelson Filipe, and Ricardo Sequeira. Robot team coordination using dynamic role and positioning assignment and role based setplays. *Mechatronics*, 21(2):445–454, 2010.

Nuno Lau, Luís Paulo Reis, Luís Mota, and Fernando Almeida. FC Portugal 2D Simulation: Team Description Paper. In Röfer et al. (2011b).

Martin Lauer, Sascha Lange, and Martin Riedmiller. Calculating the perfect match: An efficient and accurate approach for robot self-localisation. In Bredenfeld et al. (2006), pages 142–153.

Rui Lopes, Luís Mota, Luís Paulo Reis, and Nuno Lau. Playmaker: Graphical definition of formations and setplays. In *Workshop em Sistemas Inteligentes e Aplicações - 5. Conferência Ibérica de Sistemas e Tecnologias de Informação (CISTI'2010)*, 2010.

Martin Lötzsch, Joscha Bach, Hans-Dieter Burkhard, and Matthias Juengel. Designing agent behavior with the extensible agent behavior specification language XABSL. In Polani et al. (2004), pages 114–124.

Nathan Lovell and Vladimir Estivill-Castro. A descriptive language for flexible and robust object recognition. In Nardi et al. (2005), pages 540–547.

Jie Ma and Stephen Cameron. Combining policy search with planning in multi-agent cooperation. In Iocchi et al. (2009), pages 532–543.

Carlos Marques and Pedro Lima. A localization method for a soccer robot using a vision-based omni-directional sensor. In Stone et al. (2001), pages 96–107.

Colin McMillen and Manuela Veloso. Distributed, play-based role assignment for robot teams in dynamic environments. In *8th International Symposium on Distributed Autonomous Robotic Systems (DARS 2006)*, Minnesota, USA, 2006.

António Paulo Moreira, Paulo Costa, André Scolari, Armando Sousa, and Paulo Marques. 5dpo-2000 team description for RoboCup 2006. Technical report, University of Porto (UP), 2006.

Luís Mota and Luís Paulo Reis. Setplays: Achieving coordination by the appropriate use of arbitrary pre-defined flexible plans and inter-robot communication. In Alan F. T. Winfield and Jason Redi, editors, *First International Conference on Robot Communication and Coordination (ROBOCOMM 2007)*, volume 318 of *ACM International Conference Proceeding Series*, page 13. IEEE, 2007a.

Luís Mota and Luís Paulo Reis. An elementary communication framework for open cooperative RoboCup soccer teams. In *The Third International Workshop on Multi-Agent Robotic Systems (MARS 2007), at the 4th International Conference on Informatics in Control, Automation and Robotics - ICINCO 2007*, Angers, France, 2007b.

Luís Mota and Luís Paulo Reis. A Common Framework for co-operative robotics: an open, fault tolerant architecture for multi-league RoboCup teams. In Stefano Carpin, Itsuki Noda, Enrico Pagello, Monica Reggiani, and Oskar von Stryk, editors, *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*, volume 5325 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2008.

Luís Mota, Luís Paulo Reis, and Hans-Dieter Burkhard. Communication challenges raised by open co-operative teams in RoboCup. In *Encontro Científico do Festival Nacional de Robótica*, Guimarães, Portugal, 2006.

Luís Mota, Nuno Lau, and Luís Paulo Reis. Multi-robot coordination using setplays in the simulation league. In *Encontro Científico do Festival Nacional de Robótica*, Leiria, Portugal, 2010a.

Luís Mota, Luís Paulo Reis, and Nuno Lau. Co-ordination in RoboCup's 2D Simulation League: Setplays as flexible, multi-robot plans. In *IEEE International Conference on Robotics, Automation and Mechatronics (RAM 2010)*, Singapore, 2010b.

Luís Mota, Luís Paulo Reis, and Nuno Lau. Multi-robot coordination using setplays in the middle-size and simulation leagues. *Mechatronics*, 21(2):434–444, March 2011.

Jan Murray. Specifying agent behaviors with UML statecharts and StatEdit. In Polani et al. (2004).

Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors. *RoboCup-2004: Robot Soccer World Cup VIII, CD proceedings*, 2004.

Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors. *RoboCup-2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*, 2005. Springer.

Peter Naur, John W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and

M. Woodger. Report on the algorithmic language algol 60. *Commun. ACM*, 3(5): 299–314, June 1960.

António J. R. Neves, Bernardo Cunha, Armando J. Pinho, and Ivo Pinheiro. Autonomous configuration of parameters in robotic digital cameras. In *4th Iberian Conference on Pattern Recognition and Image Analysis (ibPRIA 2009)*, pages 80–87, Póvoa de Varzim, Portugal, June 2009.

Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2–3):233–250, 1998.

Nokia. Qt - A cross-platform application and UI framework, 07 2011. URL http://qt.nokia.com/products/.

Mehrab Norouzitallab, Amin Javari, Alireza Noroozi, S.M.A. Salehizadeh, and Kourosh Meshgi. Nemesis team description 2010. Technical report, Amir Kabir University of Technology, Tehran, Iran, 2010. URL http://julia.ist.tugraz.at/robocup2010/tdps/2D_TDP_Nemesis.pdf.

Pier Francesco Palamara, Vittorio A. Ziparo, Luca Iocchi, Daniele Nardi, and Pedro Lima. Teamwork design based on Petri Net plans. In Iocchi et al. (2009), pages 200–211.

Lynne E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors. *RoboCup-2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence*, 2004. Springer.

Ali Ajdari Rad, Navid Qaragozlou, and Maryam Zaheri. Scenario-based teamworking, how to learn, create, and teach complex plans? In Polani et al. (2004).

Nelson Ramos, Marco Barbosa, and Pedro Lima. Multi-robot systems middleware applied to soccer robots. In *Encontro Científico do Festival Nacional de Robótica*, Paderne, Portugal, 2007.

Luís Paulo Reis and Nuno Lau. FC Portugal team description: Robocup 2000 simulation league champion. In Stone et al. (2001), pages 29–40.

Luís Paulo Reis and Nuno Lau. Coach unilang - a standard language for coaching a (robo) soccer team. In Birk et al. (2002), pages 251–265.

Luis Paulo Reis, Nuno Lau, and Eugénio Oliveira. Situation based strategic positioning for coordinating a simulated robosoccer team. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *Balancing React. and Social Deliberation in MAS*, volume 2103 of *Lecture Notes in Artificial Intelligence*, pages 175–197. Springer, 2001.

Luís Paulo Reis, Nuno Lau, and Luís Mota. FC Portugal 2007 – 2D Simulation Team Description Paper. In Visser et al. (2007).

Luís Paulo Reis, Nuno Lau, Luís Mota, Artur Pereira, Bernardo Cunha, and João Certo. Mixed Reality Competition: FC Portugal Team Description Paper. In Iocchi et al. (2008).

Luís Paulo Reis, Nuno Lau, and Luís Mota. FC Portugal 2009 - 2D Simulation Team Description Paper. In Baltes et al. (2009).

Luís Paulo Reis, Nuno Lau, and Luís Mota. FC Portugal 2D Simulation: Team Description Paper. In del Solar et al. (2010).

Martin Riedmiller and Artur Merke. Using machine learning techniques in complex multi-agent domains. In I. Stamatescu, W. Menzel, M. Richter, and U. Ratsch, editors, *Adaptivity and Learning*, pages 311–328. Springer, 2003.

Martin Riedmiller, Artur Merke, David Meier, Alex Sinner, Ortwin Thate, and R. Ehrmann. Karlsruhe Brainstormers - a reinforcement learning way to robotic soccer. In Stone et al. (2001), pages 367–372.

Max Risler and Oskar von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in XABSL. In *AAMAS08-Workshop on Formal Models and Methods for Multi-Robot Systems*, page 7, Estoril, Portugal, 2008.

Ronald Rivest. *S-Expressions Internet-Draft*. MIT Laboratory for Computer Science, Room 324, 545 Technology Square Cambridge, MA 02139, May 1997. URL `http://people.csail.mit.edu/rivest/Sexp.txt`.

Thomas Röfer. An architecture for a national robocup team. In Kaminka et al. (2003), pages 417–425.

Thomas Röfer, Joerg Brose, Eike Carls, Jan Carstens, Daniel Goehring, Matthias Juengel, Tim Laue, Tobias Oberlies, Sven Oesau, Max Risler, Michael Spranger, Christian Werner, and Joerg Zimmer. Germanteam 2006 the german national robocup team. Technical report, Deutsches Forschungszentrum fuer Kuenstliche Intelligenz, Universitaet Darmstadt, Universitaet Bremen and Humboldt-Universitaet zu Berlin, 2006.

Thomas Röfer, Tim Laue, Colin Graf, Tobias Kastner, Alexander Fabisch, and Christian Thedieck. B-human team description for robocup 2010. In del Solar et al. (2010).

Thomas Röfer, Norbert Michael Mayer, Jesus Savage, and Uluç Saranlı, editors. *RoboCup 2011: Robot Soccer World Cup XV, CD proceedings*, 2011b.

Raquel Ros and Manuela Veloso. Executing multi-robot cases through a single coordinator. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (AAMAS 07)*, pages 215:1– 215:3. ACM, 2007.

Raquel Ros, Manuela Veloso, Ramon López de Màntaras, Carles Sierra, and Josep Lluís Arcos. *Advances in Case-Based Reasoning*, volume 4106 of *Lecture Notes in Computer Science*, chapter Retrieving and Reusing Game Plays for Robot Soccer, pages 47–61. Springer, 2006.

Raquel Ros, Ramon López de Màntaras, Josep Lluís Arcos, and Manuela Veloso. *Case-Based Reasoning Research and Development*, volume 4626 of *Lecture Notes in Computer Science*, chapter Team Playing Behavior in Robot Soccer: A Case-Based Reasoning Approach, pages 46–60. Springer, 2007.

Myriam Arias Ruiz and Jorge Ramirez Uresti. Team Agent Behavior Architecture in Robot Soccer. In *Proceedings of the Latin American Robotic Symposium*, pages 20–25, 2008.

Frederico Santos, Luís Almeida, Luís Seabra Lopes, José Luís Azevedo, and M. Bernardo Cunha. Communicating among robots in the robocup middle-size league. In Baltes et al. (2010), pages 320–331.

Nima Shafii, Luís Paulo Reis, and Nuno Lau. Biped walking using coronal and sagittal movements based on truncated fourier series. In del Solar et al. (2011), pages 324–335.

Shuhei Shiota, Yasuyuki Yamazaki, Shinichi Takahashi, Yosuke Taniguchi, and Ikuo Takeuchi. Yowai2006 team description. In *Robocup 2006 Symposium*. University of Tokyo, Japan, 2006.

Marco A. C. Simões, Josemar R. de Souza, Fagner de A. M. Pimentel, and Diego Frias. MR-Simulator: A simulator for the Mixed Reality competition of RoboCup. In del Solar et al. (2011), pages 82–96.

Hendrik Skubch, Michael Wagner, Roland Reichle, and Kurt Geihs. A modelling language for cooperative plans in highly dynamic domains. *Mechatronics*, 21(2):423 – 433, 2011. Special Issue on Advances in intelligent robot design for the Robocup Middle Size League.

Russell Smith. *Open Dynamics Engine v0.5 User Guide, http://www.ode.org/*, 2004.

Christopher Stanton and Mary-Anne Williams. Grounding robot sensory and symbolic information using the semantic web. In Polani et al. (2004), pages 757–764.

Peter Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110 (2):241–273, 1999.

Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors. *RoboCup-2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*, 2001. Springer.

Kai Stoye and Carsten Elfers. Intuitive plan construction and adaptive plan selection. In Visser et al. (2008), pages 278–285.

Freek Stulp, Michael Isik, and Michael Beetz. Implicit coordination in robotic teams using learned prediction models. In *2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006.

Freek Stulp, Hans Utz, Michael Isik, and G. Mayer. Implicit Coordination with Shared Belief: A Heterogeneous Robot Soccer Team Case Study. *Advanced Robotics*, 24(7): 1017–1036, 2010.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

Hans Utz, Freek Stulp, and Arndt Muehlenfeld. Sharing belief in teams of heterogenous robots. In Nardi et al. (2005).

José Luis Vega, Ma. de los Ángeles Junco, and Jorge Ramírez. Major behavior definition of football agents through XML. In Tamio Arai, Rolf Pfeifer, Tucker R. Balch, and Hiroshi Yokoi, editors, *Proceedings of the 9th International Conference on Intelligent Autonomous Systems*, pages 668–675, University of Tokyo, Tokyo, Japan, 2006. IOS Press. ISBN 1-58603-595-9.

Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors. *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Artificial Intelligence*, 2000. Springer.

Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors. *RoboCup-2007: Robot Soccer World Cup XI, CD proceedings*, 2007.

Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors. *RoboCup-2007: Robot Soccer World Cup XI*, volume 5001 of *Lecture Notes in Artificial Intelligence*, 2008. Springer.

Andrew Walker. *An Introduction to Boost*. The Code Project, Jul 2003. URL `http://www.codeproject.com/KB/stl/boostintro.aspx`.

Thilo Weigel, Willi Auerbach, Markus Dietl, Burkhard Dumler, Jens-Steffen Gutmann, Kornel Marko, Klaus Müller, Bernhard Nebel, Boris Szerbakowski, and Maximilian Thiel. Cs freiburg: Doing the right thing in a group. In Stone et al. (2001), pages 52–63.

Jan Wendler and Joscha Bach. Recognizing and predicting agent behavior with case based reasoning. In Polani et al. (2004), pages 729–738.

Henry Work, Eric Chown, Tucker Hermans, Jesse Buttereld, and Mark McGranaghan. Player positioning in the four-legged league. In Iocchi et al. (2009), pages 391–402.

K. Yokota, K. Ozaki, N. Watanabe, A. Matsumoto, D. Koyama, T. Ishikawa, Kuniaki Kawabata, Hayato Kaetsu, and Hajime Asama. Uttori united: Cooperative team play based on communication. In Asada and Kitano (1999), pages 479 – 484.

Stefan Zickler, Tim Laue, Oliver Birbach, Mahisorn Wongphati, and Manuela Veloso. SSL-Vision: The shared vision system for the RoboCup Small Size League. In Baltes et al. (2010), pages 425–436.

Vittorio A. Ziparo, Luca Iocchi, Daniele Nardi, Pier Francesco Palamara, and Hugo Costelha. Petri net plans: a formal model for representation and execution of multi-robot plans. In *Proceedings of the 7th International Joint Conference on Autonomous Agents & Multiagent Systems*, volume 1 of *AAMAS'08*, pages 79–86, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-0-9.

Oliver Zweigle, Reinhard Lafrenz, Thorsten Buchheim, Uwe-Philipp Käppeler, Hamid Rajaie, Frank Schreiber, and Paul Levi. Cooperative agent behavior based on special interaction nets. In *Intelligent Autonomous Systems 9: IAS-9*, 2006.

Oliver Zweigle, U.-P. Käppeler, H. Rajaie, K. Hüssermann, R. Lafrenz, A. Tamke, F. Schreiber, M. Höferlin, M. Schanz, and P. Levi. CoPS Stuttgart team description 2008. In Iocchi et al. (2008).