Users Manual(日本語版)

RoboCup Soccer Server

for Soccer Server Version 7.07 and later

Mao Chen†, Klaus Dorer,
Ehsan Foroughi, Fredrik Heintz,
ZhanXiang Huang†, Spiros Kapetanakis,
Kostas Kostiadis, Johan Kummeneje,
Jan Murray, Itsuki Noda,
Oliver Obst, Pat Riley,
Timo Steffens, Yi Wang† and
Xiang Yin†

日本語訳
秋山 英久 <akiyama@ntt.dis.titech.ac.jp>

November 21, 2004

† <ustc9811@sina.com>
<klaus.dorer@living-systems.de>
<foroughi@ce.sharif.edu>
<frehe@ida.liu.se>
<spiros@cs.york.ac.uk>
<kkosti@essex.ac.uk>
<johan.kummeneje@generalwireless.se>
<murray@uni-koblenz.de>
<noda@etl.go.jp>
<fruit@uni-koblenz.de>
<pfr+@cs.cmu.edu>
<timosteffens@gmx.de>

Acknowledgements

We are very grateful for the work of the authors from the previous versions of the manual that could not help out on this version:

- *David Andre* at Berkeley, University of California, USA.
- *Pascal Gugenberger* at Humboldt University, Berlin, Germany.
- *Marius Joldos* at Technical University of Cluj-Napoca, Romania.
- *Paul Arthur Navratil* at University of Texas at Austin, USA.
- *Peter Stone* at University of Texas at Austin, USA.
- *Tomoichi Takahashi* at Chubu University, Japan.
- *Tralvex Yeap* at KRDL, Singapore.
- *Emiel Corten* at University of Amsterdam, Netherlands.
- *Helmut Myritz* at Humboldt University, Germany.
- *Jukka Riekki* at Oulu University, Finland.

Besides the authors, we would also like to thank Stefan Sablatnög from the University of Ulm, Germany, and Mike Hewett from University of Texas at Austin, USA, for a thorough proofreading of the soccermanual 4.00. We have also received a lot of good suggestions from Erik Mattsson at the University of Karlskrona/Ronneby, Sweden.

We would not have been able to do this manual without the above mentioned people[1].

This product includes software developed by the University of California, Berkeley and its contributors – namely flex.

[1] The persons listed on the title page are the persons responsible for the different sections of the manual.

Contents

1 イントロダクション

We are in the early days of RoboCup [7], with half a century to go before we can "...build a team of robot soccer players, which can beat a human world cup champion team" ([6], p. v). The challenge posed by the goal is enormous and inspires hundreds of researchers yearly throughout the world to engage themselves and their students in RoboCup. RoboCup has been used as a research challenge in parallel with a usage for educational purposes, and to stimulate the interest of the public for robotics and artificial intelligence (AI). Each year since 1997, researchers from different countries have gathered to play the world cup. The event has drawn an increasing amount of interest from the public, as robotics is still not commonplace.

The intention of this manual[1] is to guide the developers of simulated league teams in the beginning steps, and also serve as a reference manual for the experienced users.

1.1 背景

Mackworth [11] introduced the idea of using soccer-playing robots in research. Unfortunately, the idea did not get the proper response until the idea was further developed and adapted by Kitano, Asada, and Kuniyoshi, when proposing a Japanese research programme, called Robot J-League[2]. During the autumn of 1993, several American researchers took interest in the Robot J-League, and it thereafter changed name to the Robot World Cup Initiative or *RoboCup* for short. RoboCup is sometimes referred to as the RoboCup challenge or the RoboCup domain.

In 1995, Kitano et al. [7] proposed the first Robot World Cup Soccer Games and Conferences to take place in 1997. The aim of RoboCup was to present a new standard problem for AI and robotics, somewhat jokingly described as the life of AI after Deep Blue[3]. RoboCup differs from previous research in AI by focusing on a distributed solution instead of a centralised solution, and by challenging researchers from not only traditionally AI-related fields, but also researchers in the areas of robotics, sociology, real-time mission critical systems, etc.

To co-ordinate the efforts of all researchers, the RoboCup Federation was formed. The goal of RoboCup Federation is to promote RoboCup, for example by annually arranging the world cup tournament. Members of the RoboCup Federation are all active researchers in the field, and represent a number of universities and major companies. As the body of researchers is quite large and widespread, local committees are formed to promote RoboCup-related events in their geographical area.

[1] Parts of this chapter is taken directly from [18]
[2] The J-League is the professional soccer league in Japan.
[3] In reference to Deep Blue and its games with Kasparov, see `http://www.chess.ibm.com`.

1.2 RoboCupの目標

The RoboCup Federation has set goals and a timetable for the research. Setting goals and a timetable are means of pushing the state-of-the-art further, in conjunction with formalised test-beds. In resemblance with John F. Kennedy's national goal of "landing a man on the moon and returning him safely to earth" ([4], p. 8276), the main accomplishment was not to land a man on the moon and returning him safely, but the overall technological advancement. Therefore, the most important goal of RoboCup is to advance the overall technological level of society, and as a more pragmatic goal to achieve the following:

> 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[4], against the winner of the most recent World Cup [15].

There will be several technological advancements, even if the goal of the robotic soccer team is not reached, starting with Team-Partitioned, Opaque-Transition Reinforcement Learning (TPOT-RL) [19] which has found application in the domain of packet routing in computer networks. TPOT-RL is a distributed learning method in domains where "agents have limited information about environmental state transitions" ([19], p. 22).

In most RoboCup leagues, the teams consist of either robots or programs that co-operate in order to defeat the opponent team. RoboCup Rescue and the commentator exhibition diverge from the other RoboCup leagues. The goal of defeating an opponent would raise ethical issues in RoboCup Rescue, since we cannot assign comparable utilities to human lives and buildings. Hence, the focus in RoboCup Rescue is on the co-operative efforts between heterogeneous agents. In the commentator exhibition, the goal is to observe and comment.

Besides the commentator exhibition and RoboCup Rescue, the main body of the RoboCup challenge consists of several leagues for soccer playing. However, as this manual is about the simulated league we will only focus on it.

1.2.1 シミュレーションリーグ

The RoboCup simulator league is based on the RoboCup simulator called the soccer server [13], a physical soccer simulation system. All games are visualised by displaying the field of the simulator by the soccer monitor on a computer screen. The soccer server is written to support competition among multiple virtual soccer players in an uncertain multi-agent environment, with real-time demands as well as semi-structured conditions. One of the advantages of the soccer server is the abstraction made, which relieves the researchers from having to handle robot problems such as object recognition [9], communications, and hardware issues, e.g., how to make a robot move. The abstraction enables researchers to focus on higher level concepts such as co-operation and learning.

Since the soccer server provides a challenging environment, i.e., the intentions of the players cannot mechanically be deduced, there is a need for a referee when playing

[4] Fédération Internationale de Football Association (FIFA) defines the rules of soccer [27].

a match. The included artificial referee is only partially implemented and can detect trivial situations, e.g., when a team scores. However, there are several hard-to-detect situations in the soccer server, e.g., deadlocks, which brings the need for a human referee. All participating teams are also obliged to play according to a gentlemen's agreement, e.g., not to use loopholes.

Since the first version of the soccer server was completed in 1995, there have been four world cups and one pre-world cup event, not to mention all other RoboCup-related events. The 1996 pre-RoboCup event [5] was held in Osaka, with only seven entrants in the competition which ended with a Japanese victory by the team Ogalets from Tokyo University. In Nagoya the following year, the first formal competition was held in conjunction with the IJCAI'97 conference. The competition had 29 teams participating, and the winner was AT Humboldt [2]. The RoboCup world cup of 1998 was played in conjunction with the human world cup in Paris, and the winning team was CMUnited98 [29]. During the world cup, media was heavily covering the event, as it was public in a museum in the suburbs of Paris. The year after, the world cup was held in conjunction with IJCAI'99 in Stockholm, and the winners (once again) were CMUnited99 [30]. An unchanged version of the champion team must participate, as a benchmark, in the next world cup. The benchmarking teams have always been able to win their group, but only in 2000 did the benchmark team advance further than the first game after group play.

1.2.2 サッカーサーバとは

Soccer Server is a system that enables autonomous agents consisting of programs written in various languages to play a match of soccer (association football) against each other.

A match is carried out in a client/server style: A server, soccerserver, provides a virtual field and simulates all movements of a ball and players. Each client controls movements of one player. Communication between the server and each client is done via UDP/IP sockets. Therefore users can use any kind of programing systems that have UDP/IP facilities.

The soccerserver consists of 2 programs, `soccerserver` and `soccermonitor`. `Soccer Server` is a server program that simulates movements of a ball and players, communicates with clients, and controls a game according to rules. `Soccermonitor` is a program that displays the virtual field from the soccerserver on the monitor using the X window system. A number of `soccermonitor` programs can connect with one `soccerserver`, so we can display field-windows on multiple displays.

A client connects with `soccerserver` by an UDP socket. Using the socket, the client sends commands to control a player of the client and receives information from sensors of the player. In other words, a client program is a brain of the player: The client receives visual and auditory sensor information from the server, and sends control-commands to the server.

Each client can control only one player[5][6]. So a team consists of the same number of

[5] Technically, it is easy to cheat the server. Therefore this is a gentleman's agreement.

[6] In order to test various kinds of systems, we may permit a client to control multiple players if the different control modules of players are separated logically from each other in the client.

clients as players. Communications between the clients must be done via `soccerserver` using `say` and `hear` protocols. (See section 4.2.1.) One of the purposes of soccerserver is evaluation of multi-agent systems, in which efficiency of communication between agents is one of the criteria. Users must realize control of multiple clients by such restricted communication.

1.3 歴史

In this section we will first describe the history of the soccerserver and thereafter the history of the RoboCup Simulation League. To end the section we will also describe the history of the manual effort.

1.3.1 Soccer Server の歴史

The first, preliminary, original system of soccerserver was written in September of 1993 by Itsuki Noda, ETL. This system was built as a library module for demonstration of a programming language called MWP, a kind of Prolog system that has multi-threads and high level program manipulation. The module was a closed system and displayed a field on a character display, that is VT100.

The first version (version 0) of the client-server style server was written in July of 1994 on a LISP system. The server shows the field on an X window system, but each player was shown in an alphabet character. It used the TCP/IP protocol for connections with clients. This LISP version of soccerserver became the original style of the current soccerserver. Therefore, the current soccerserver uses S-expressions for the protocol between clients and the server.

The LISP version of soccerserver was re-written in C++ in August of 1995 (version 1). This version was announced at the IJCAI workshop on Entertainment and AI/Alife held in Montreal, Canada, August 1995.

The development of version 2 started January of 1996 in order to provide the official server of preRoboCup-96 held at Osaka, Japan, November 1996. From this version, the system is divided into two modules, soccerserver and soccerdisplay (currently, soccermonitor). Moreover, the feature of coach mode was introduced into the system. These two features enabled researchers on machine learning to execute games automatically. Peter Stone at Carnegie Mellon University joined the decision-making process for the development of the soccerserver at this stage. For example, he created the configuration files that were used at preRoboCup-96.

After preRoboCup-96, the development of the official server for the first RoboCup, RoboCup-97 held at Nagoya, Japan, August 1997, started immediately, and the version 3 was announced in February of 1997. Simon Ch'ng at RMIT joined decisions of regulations of soccerserver from this stage. The following features were added into the new version:

- logplayer
- information about movement of seen objects in visual information

- capacity of hearing messages

The development of version 4 started after RoboCup-97, and announced November 1997. From this version, the regulations are discussed on the mailing list organized by Gal Kaminka. As a result, many contributers joined the development. Version 4 had the following new features:

- more realistic stamina model
- goalie
- handling offside rule
- disabling players for evaluation
- facing direction of players in visual information
- sense_body command

Version 4 was used in Japan Open 98, RoboCup-98 and Pacific Rim Series 98.

Version 5 was used in Japan Open 99, and will also be used in RoboCup-99 in Stockholm during the summer of 1999.

In Melbourne 2000, version 6 was used, and for the world cup in 2001 version 7 will be used.

1.3.2 RoboCup シミュレーションリーグの歴史

The RoboCup simulation league has had five main official events: preRoboCup-96, RoboCup-97, RoboCup-98, RoboCup-99, and RoboCup 2000. Research results have been reported extensively in the proceedings of the workshops and conferences associated with these competitions. In this section, we focus mainly on the competitions themselves.

preRoboCup-96

preRoboCup-96 was the first robotic soccer competition of any sort. It was held on November 5–7, 1996 in Osaka, Japan [5]. In conjunction with the IROS-96 conference, preRoboCup-96 was meant as an informal, small-scale competition to test the RoboCup soccerserver in preparation for RoboCup-97. 5 of the 7 entrants were from the Tokyo region. The other 2 were from Ch'ng at RMIT and Stone and Veloso from CMU.

The winning teams were entered by:

1. Ogawara (Tokyo University)
2. Sekine (Tokyo Institute of Technology)
3. Inoue (Waseda University)
4. Stone and Veloso (Carnegie Mellon University)

In this tournament, team strategies were generally quite straightforward. Most of the teams kept players in fixed locations, only moving them towards the ball when it was nearby.

RoboCup-97

The RoboCup-97 simulator competition was the first formal simulated robotic soccer competition. It was held on August 23–29, 1997 in Nagoya, Japan in conjunction with the IJCAI-97 conference [6]. With 29 teams entering from all around the world, it was a very successful tournament.

The winning teams were entered by:

1. Burkhard et al. (Humboldt University)
2. Andou (Tokyo Institute of Technology)
3. Tambe et al. (ISI/University of Southern California)
4. Stone and Veloso (Carnegie Mellon University)

In this competition, the champion team exhibited clearly superior low-level skills. One of its main advantages in this regard was its ability to kick the ball harder than any other team. Its players did so by kicking the ball around themselves, continually increasing its velocity so that it ended up moving towards the goal faster than was imagined possible. Since the soccerserver did not (at that time) enforce a maximum ball speed, a property that was changed immediately after the competition, the ball could move arbitrarily fast, making it almost impossible to stop. With this advantage at the low-level behavior level, no team, regardless of how strategically sophisticated, was able to defeat the eventual champion.

At RoboCup-97, the RoboCup scientific challenge award was introduced. Its purpose is to recognize scientific research results regardless of performance in the competitions. The 1997 award went to Sean Luke [10] of the University of Maryland "for demonstrating the utility of evolutionary approach by co-evolving soccer teams in the simulator league."

RoboCup-98

The second international RoboCup championship, RoboCup-98, was held on July 2–9, 1998 in Paris, France [1]. It was held in conjunction with the ICMAS-98 conference.

The winning teams were entered by:

1. Stone et al. (Carnegie Mellon University)
2. Burkhard et al. (Humboldt University)
3. Corten and Rondema (University of Amsterdam)
4. Tambe et al. (ISI/University of Southern California)

Unlike in the previous year's competition, there was no team that exhibited a clear superiority in terms of low-level agent skills. Games among the top three teams were all quite closely contested with the differences being most noticeable at the strategic, team levels.

One interesting result at this competition was that the previous year's champion team competed with minimal modifications and finished roughly in the middle of the final standings. Thus, there was evidence that as a whole, the field of entries was much stronger than during the previous year: roughly half the teams could beat the previous champion.

The 1998 scientific challenge award was shared by Electro Technical Laboratory (ETL), Sony Computer Science Laboratories, Inc., and German Research Center for Artificial Intelligence GmbH (DFKI) for "development of fully automatic commentator systems for RoboCup simulator league."

To encourage the transfer of results from RoboCup to the scientific community at large, RoboCup-98 was the first to host the Multi-Agent Scientific Evaluation Session. 13 different teams participated in the session, in which their adaptability to loss of team-members was evaluated comparatively. Each team was played against the same fixed opponent (the 1997 winner, AT Humboldt'97) four half-games under official RoboCup rules. The first half-game (phase A) served as a base-line. In the other three half-games (phases B-D), 3 players were disabled incrementally: A randomly chosen player, a player chosen by the representative of the fixed opponent to maximize "damage" to the evaluated team, and the goalie. The idea is that a more adaptive team would be able to respond better to these.

Very early on, even during the session itself, it became clear that while in fact most participants agreed intuitively with the evaluation protocol, it wasn't clear how to quantitatively, or even qualitatively, analyse the data. The most obvious measure of the goal-difference at the end of each half may not be sufficient: some teams seem to do better with less players, some do worse. Performance, as measured by the goal-difference, really varied not only from team to team, but also for the same team between phases. The evaluation methodology itself and analysis of the results became open research problems in themselves. To facilitate this line of research, the data from the evaluation was made public at: `http://www.isi.edu/~galk/Eval/`

RoboCup-99

The third international RoboCup championship, RoboCup-99, was held in late July and early August, 1999 in Stockholm, Sweden [3]. It was held in conjunction with the IJCAI-99 conference.

RoboCup 2000

The fourth international RoboCup championship, RoboCup 2000, was held in early September, 2000 in Melbourne, Australia [16]. It was held in conjunction with the PRICAI-2000 conference.

1.3.3 マニュアル作成の歴史

The first versions of the manual were written by Itsuki Noda, while developing the soccerserver, and around version 3.00 there were several requests on an updated manual, to better correspond to the server as well as enable newcomers to more easily participate in the RoboCup World Cup Initiative. In the fall of 1998 Peter Stone initiated the Soccer Manual Effort, over which Johan Kummeneje took responsibility to organize and as a result the Soccer Server Manual version 4.0 was released on the 1st of November 1998.

In 1999, the manual for the soccerserver version 5.0 was released. Unfortunately the manual lost part of its pace, and there was no release of the manual for soccerserver version 6.0.

Since 1999, the soccerserver has changed major version to 7 and is continuously developed. Therefore the Soccer Manual Effort has developed a new version, which you are currently reading.

1.4 このマニュアルについて

This manual is the joint effort of the authors from a diverse range of universities and organizations, which build upon the original work of *Itsuki Noda*.
If there are errors, inconsistencies, or oddities, please notify johank@dsv.su.se or fruit@uni-koblenz.de with the location of the error and a suggestion of how it should be corrected.

We are always looking for anyone who has an idea on how to improve the manual, as well as proofread or (re)write a section of the manual. If you have any ideas, or feel that you can contribute with anything to the SoccerServer Manual Effort please mail johank@dsv.su.se or fruit@uni-koblenz.de.

The latest manual can be downloaded at `http://www.dsv.su.se/~johank/RoboCup/manual`.

1.5 マニュアルガイド

The thesis is written for a wide range of readers, and therefore the chapters are not equally important to all readers. We shortly describe the remaining chapters to give an overview of the thesis.

Chapter 2 introduces the concepts of the simulated league and will help the newcomer to get to terms with the different parts.

Chapter 3 helps the beginners to start compiling and running the software.

Chapter 4 describes the soccerserver.

Chapter 5 describes the soccermonitor.

Chapter 6 describes the soccerclient and how to create one.

Chapter 7 describes the coachclient.

Chapter 8 suggests some further reading.

2 概要

2.1 Getting Started

この節では，RoboCup サッカーシミュレータの主な構成要素の手短な紹介を行う．各構成要素に関する詳細な情報 (設定パラメータやランタイムオプションなど) は，このマニュアル内の別の場所で記述されるだろう．

2.1.1 サーバ

サーバは様々なチームが作家の試合を行うことを可能にするシステムである．試合はクライアント・サーバ方式で実行されるため，チーム開発手法に関する制限は存在しない．クライアント-サーバ間の UDP/IP 通信さえサポートしていれば，チーム開発に用いるツールに何を用いても良い．これは，サーバと各クライアントとの間の通信が全て UDP/IP ソケットによって行われるためである．各クライアントは個別のプロセスであり，指定のポートを通してサーバと接続する．プレイヤがサーバへ接続した後は，全てのメッセージがこのポートを通して運ばれる．チームは最大 12 個のクライアント、すなわち 11 個のプレイヤ (フィールダ 10 体 + キーパ 1 体) とコーチを持つことができる．プレイヤは，実行したい行動 (ボールキック，ターン，ダッシュなど) に関するリクエストをサーバへと送信する．サーバはそれらのメッセージを受信し，リクエストを処理し，それに応じて環境を更新する．更に，サーバは全てのプレイヤにセンサ情報 (フィールド上の物体の位置に関する視覚データ，スタミナやスピードのようなプレイヤのリソースに関するデータなど) を与える．サーバは離散時間間隔 (またはサイクル) で稼働する実時間システムである，という点を言及しておかなければならないだろう．各サイクルは指定の持続期間を持つ．与えられたサイクル内で実行する必要のある行動は，正規の時間間隔のうちにサーバへ届いていなければならない．それゆえ，プレイヤの実行効率が低ければ行動が失われる機会が増え，チーム全体としてのパフォーマンスにも重大な影響を与える．サーバに関する詳細な説明は 4 章に書かれている．

2.1.2 モニタ

Soccer Monitor を用いることで，試合中にサーバ内で起こっていることを人間が確認することが可能になる．現在，モニタとして `rcssmonitor` と `rcssmonitor_classic` の二種類が用意されている．モニタに表示される情報には，得点，チーム名，全てのプレイヤとボールの位置などが含まれている．モニタはサーバへのシンプルなインターフェイスも提供している．例えば，モニタ上の"Kick-Off" ボタンは，両チームの接続後，人間の審判が試合が開始するために使用される．`rcssmonitor` は Artur Merke によって開発された `frameview` を基にしている．`rcssmonitor` はクラシックモニタを拡張したものであり，いくつかの機能が追加されている．

- I フィールドを拡大表示できる．この機能はデバッグ目的に特に有益である．
- 全てのプレイヤとボールの現在の位置と速度をいつでもコンソールに表示できる．
- 様々な情報をモニタ上に表示できる．例えば，プレイヤの視界コーン，スタミナやプレイヤタイプなどである．
- プレイヤとボールをマウスによって移動させることができる．

サーバ上で試合を実行するだけであれば，モニタは必要とされない．しかし，必要とされれば，複数のモニタを同時にサーバへと接続することができる (例えば，異なる端末で同じ試合を観たい場合など)．モニタに関する詳細は 5 章を参照してもらいたい．

2.1.3 ログプレイヤ

ログプレイヤは試合再生のために使用されるツールであり，ビデオプレイヤのようなものである．サーバを実行する際に特定のオプションを設定すると，サーバは試合に関するデータをハードドライブへと保存する (ビデオの録画ボタンを押すようなものである)．それから，`rcsslogplayer` プログラムをモニタと組み合わせて使用することによって，何度でも試合を再生できる．このことはチーム解析やチームの長所/短所の発見のために有益である．ビデオプレイヤと同様に，ログプレイヤは play, stop, fast forward, rewind ボタンを備えている．更に，ログプレイヤは試合の特定のサイクルへジャンプすることも可能にする (例えば，ゴールのみを観たい場合)．最後に，ログプレイヤは記録内容の編集も可能にする．例えば，(複数の) 試合から興味深いシーンを抜き出し，別のログファイルへと保存することが可能である．これによって，プレゼンテーションの作成が容易になる．

ログプレイヤは簡単な GUI，またはコマンドラインインターフェイスによって制御できる．コマンドはファイルから読み込むことも可能である．このファイルによって，制限されたスクリプティング能力がログプレイヤへ追加される．

2.1.4 デモクライアント

RoboCup Soccer Simulator に同梱された `rcssclient` というプログラムである．非常に原始的なテキストベースのクライアントが実装されている．このプログラムの目的は，クライアント開発のための最初のアイデアを開発者へと与えることである．

`rcssclient` が起動すると，プログラムはサーバに接続し，シンプルな ncurses ベースインターフェイスが現れる．このインターフェイスによって，サーバで実行されるコマンドを入力することができる．クライアントが受信した情報は，情報のタイプ (*visual*, *sense body*, *other* ...) に応じてスクリーンの異なる区画に表示される．コマンドを入力し，起こったことを観察することによって，シミュレーションにおいてなすべき仕事のアイデアを得ることができるだろう．あなたが既に新規参加者ではないとしても，このプログラムはテストのために便利のよいものだろう．例えば，シミュレーションに新しいコマンドを追加するためのテストに利用できるだろう．

2.2 試合ルール

試合中，サーバに組み込まれた自動審判，または人間の審判によって多くのルールが執行される．この節の目的は，これらのルールがどのようにして働くか，そして，それらがどのように試合に影響するか、を解説することである．

2.2.1 自動審判が判断するルール

キックオフ

キックオフ直前 (ハーフの開始前かゴール後のいずれか)，全てのプレイヤは自陣内にいなければならない．この移動を行うために，得点が入った後，審判は試合を 5 秒間停止させる．この間に，プレイヤは **move** コマンドを用いることによって自陣内へ瞬間移動することができる．このため，自陣内へ走って移動する必要は無く，無駄なスタミナを消費しなくて済む．5 秒経過後にプレイヤが敵陣内に残っていれば，プレイヤは審判によって自陣内のランダムな位置へと移動させられる．

ゴール

チームが得点を挙げたとき，審判は多くのタスクを実行する．まず，全てのプレイヤゴールのメッセージを配信して通知する．更に，審判はスコアを更新し，(前節で述べたように) 自陣内へ移動する余裕をプレイヤに与えるために試合を 5 秒間停止させる．最後に，ボールをフィールド中央へ移動させ，プレイモードを kick_off_x (x は l か r のいずれか) に変更する．

フィールドアウト

ボールがフィールドの外に出たとき，審判はボールを適切な位置 (タッチライン，コーナー，ゴールエリアなど) へ移動させ，プレイモードを kick_in, corner_kick, または goal_kick に変更する．コーナーキックの場合では，審判はフィールドコーナーの内側 (1m, 1m) の位置にボールを置く．

プレイヤ除去

プレイモードが kick_off, free_kick, kick_in, または corner_kick のとき，ボールから一定距離以内にいる守備側の全てのプレイヤが審判によってボールから遠ざけられる．この距離はサーバパラメータで設定されている (通常，9.15 メートル)．遠ざけられたプレイヤはこの円の円周上に移動させられる．プレイモードが offside のとき，攻撃側のプレイヤは非オフサイドの位置まで引き戻される．該当するのは，オフサイドエリアにいる全てのプレイヤと，ボールから 9.15 メートル以内にいる全てのプレイヤである．プレイモードが goal_kick のとき，攻撃側のプレイヤはペナルティエリアの外へ移動させられる．ゴールキックが行われている間，攻撃側プレイヤがペナルティエリア内へ再侵入することはできない．ボールがペナルティエリアの外へ出ると，直ちにプレイモードが play_on に変更される．

プレイモード制御

プレイモードが kick_off, free_kick, kick_in, または corner_kick のとき，プレイヤの **kick** コマンドによってボールが動き始めると，審判はすぐにプレイモードを play_on に変更する．

オフサイド

次の条件を満たすプレイヤはオフサイドを取られる:
– 敵陣内にいる．
– 少なくとも二人の敵プレイヤよりも敵ゴールラインに近い．
– ボールよりも敵ゴールラインに近い．
– ボールとの距離が 2.5m 以内である．(この値はサーバパラメータ *offside_active_area_size* によって変更できる)

バックパス

実際のサッカーと同様に，見方からパスされたボールをキーパがキャッチすることは禁止されている．この違反が発生すると，審判は *back_pass_l* または *back_pass_r* をコールし，相手チームへフリーキックを与える．バックパスはペナルティエリア内でのみ発生するので，ボールはペナルティエリアのコーナーに置かれる．キーパがキャッチを行った位置に近い方のコーナーが選択される．キーパがボールをキャッチしようとしなければ，キーパへのパスで違反を取られることはない．

フリーキックフォルト

フリーキック，コーナーキック，キャッチ後のキーパのキック，またはキックインが実行されるとき，プレイヤが自分自身へパスすることは禁止されている．プレイヤがこれらのフリーキック実行後に再びボールをキックすれば，審判は *free_kick_fault_l* または *free_kick_fault_r* をコールし，相手チームへフリーキックを与える．

ボールを目標速度まで加速するために，プレイヤは二回以上キックを実行しなければならないかもしれない．そのため，次の条件を満たした場合にのみフリーキックフォルトが取られる:

1. フリーキックを行ったプレイヤが再び最初にボールをキックした．
2. キックの間にそのプレイヤが移動した (dash を行った)．

よって，**kick–kick–dash** または **kick–turn–kick** というコマンド列の実行は全く違反にならない．一方，**kick–dash–kick** というコマンド列はフリーキックフォルトを取られる．

ハーフタイムとタイムアップ

前半または後半が終了したとき，審判は試合を中断する．各ハーフのデフォルトの長さは 3000 シミュレーションサイクル (約 5 分) である．後半終了時にスコアが引き分けであれば，試合が延長される．延長戦では得点が入った時点で試合が終了し，得点を入れたチームに勝利が与えられる．“ゴールデンゴール” 方式または “サドンデス” としても知られているルールである．

2.2.2 人間の審判が判断するルール

プレイヤの意図に関係するため，“オブストラクション”(妨害) のようなファウルを自動的に判断することは困難である．このような状況を解決するために，サーバは人間の介入を可能にするインターフェイスを提供する．これによって，人間の審判が試合を中断し，いずれ可のチームにフリーキックを与えることが可能となっている．下記は，RoboCup 2000 の競技会に先立って採択されたガイドラインであり，その後使用され続けているものである．

- ボールを取り囲む．
- あまりに多数のプレイヤでゴールをブロックする．
- 与えられたサイクル内でボールをプレイに戻さない (キックインなどを行わない)．
  現在では，このルールは自動審判によって充分に扱われる．もし *drop_ball_time* サイクル経過してもボールをプレイに戻せなければ，審判によって drop_ball が発行される．しかしながら，ボールをプレイに戻すのを何度も失敗するようであれば，人間の審判が先にボールをドロップしてもよい．
- 他のプレイヤの動きを意図的にブロックする．
- キーパが **catch** コマンドを繰り返す (ペナルティエリア内で安全にボールを移動させることが可能となってしまうため，キーパがキックとキャッチを繰り返してはならない)．
- メッセージでサーバを溢れさせる．
  プレイヤは，1 サイクル中に 3 また 4 以上のコマンドをサーバへ送信するべきではない．サーバに以上が認められれば，または試合終了後に要求があれば，コマンドの乱用がチェックされるかもしれない．
- 不適切な振る舞い
  プレイヤが不適切な手段で試合を妨害しているのが確認されれば，人間の審判は試合を中断して相手チームへフリーキックを与えることができる．

3 Getting Started

この節は，RoboCup Soccer Simulator のインストールに必要なソースファイル入手のための全ての情報を含む．このマニュアルを読んでいるということは，あなたは既に，RoboCup Soccer Simulator に関するソフトウェアとドキュメントを見つけていることだろう．しかし，用心のために伝えることにする． :-)

3.1 ホームページ

RoboCup Soccer Simulator の項式ホームページは，`http://sserver.sourceforge.net/` で見付けられる．このページは，一般的な RoboCup や RoboCup Soccer Simulator に関する有益な情報やリンクを含む．

ページ内のコンテンツの簡単な説明を行う:

Home: RoboCup Soccer Simulator ホームページ．

SF Project Page: SourceForge.net のプロジェクト管理ページ．

Downloads: ダウンロード可能なパッケージについての簡単な説明を行い，実際のダウンロードページへのリンクを張っている．パッケージに関する，より詳細な情報を得るために，このページを参照しようとするかもしれない．しかし，手早く始めるためには，3.3 節の導入方法を参照する方が良いだろう．

Mailing Lists: 公式メーリングリストへの案内．

Links: RoboCup 主導者に関するサイトへのリンク集．

About us: RoboCup Federation の一般的な組織構成を示す．RoboCup シミュレーションリーグと RoboCup Soccer Simulator メンテナンス委員会に関するより詳細な情報を得られる．

Events: これまでに行われた，または予定されている，RoboCup イベントのリスト．ローカルイベントと国際イベントのいずれも含まれる．

Competition Results: 競技会の結果のデータベース．

Teams: チーム情報のデータベース．ユーザによる登録が可能である．

SSIL: *Simulated Soccer Internet League* は，Koblenz 大学でホストされている，連続的に実行されるトーナメントである．参加チームは，わずかに設定変更された (実効速度を落とされた) サーバを用いてトーナメントを行う．トーナメントが終了すると，次のラウンドが開始される．

SSIL は，継続的なフィードバックをチーム開発者へ提供し，サッカーエージェントの能力改善のための価値ある助けをもたらす．従って，このイベントにおいては，トーナメントに勝つことだけでなく，アイデアをテストしデータを集めることも重要な点である．

あなたが自身のチームを持っているなら，このリーグへの参加を望むかもしれない．その場合は，SSIL ページの New User Request のリンク先から登録の申込を行うだけで良い．

RoboCup: RoboCup Federation の公式ホームページ．このページから，他のリーグ，目的や歴史など，RoboCup に関する全ての情報を得られる．

Clients: RoboCup 2002 以降，チームバイナリのリリースが参加者に義務づけられた．それ以前にも，多くのチームはバイナリを公開し，いくつかのチームはソースコード (の一部) も公開していた．クライアントリポジトリはそれらを集め，ダウンロード可能にしている．あなたのチームをテストするための対戦チームが必要であれば，このページを見るべきである．

Libraries: このページでは，シミュレータと併せて用いられる，いくつかの有益なライブラリを公開している．通常，それらのライブラリは，チームの低レベル部分を扱うためのものである．例えば，サーバとの接続 と 同期，ワールドモデル，そして ドリブル や パスのような基本スキルである．

SIGs: RoboCup *Special Interest Groups* のリストが得られる．RobCup は科学的な背景を持っているため，例年の競技会に加えて，科学的な下部組織構造が存在する．RoboCup special interest groups では，特定の研究観点に焦点が合わせられる．

ページのその他の部分は，リリースに関する最新ニュースや，SourceForge によって提供されるプロジェクト管理ツールへのショートカットなどである．

最後に，もし RoboCup Soccer Simulator メンテナンスグループへ寄付を贈ってもらえるなら，メニュー下部の PayPal ボタンをクリックしてください．

3.2 サーバの入手とインストール

ここで示す手順は，SuSE7.3-GNU/Linux 2.4.10-4GB が動作する PC(`uname -sr` によって，あなたの環境を確認できる)，gcc 2.95.3 と gcc 3.2(`gcc -v` によって，使用する gcc のバージョンを確認できる)，で実行したものである．よりバージョンの新しい環境であれば，上手くインストールできるはずである．以下では，`->`はコマンドラインプロンプトを意味する．

ソースファイルか RPM をシミュレータのプロジェクトサイトから入手する:

- `http://sourceforge.net/projects/sserver`

Simulator Official Release リンクをクリックすれば，tar.gz ファイル，または RPM ファイルをリストしたページが表示される．

この文書の執筆時点 (Jan-22-2003) における最新バージョンは，9.2.2 である．以下の例はこのバージョンに基づく．9.2.2 を最新バージョンの置き換えて読んでください．

シミュレータ全体は rcsoccersim-9.2.2.tar.gz ファイルとしてダウンロード可能である．モジュール毎に別々にダウンロードすることも可能である．

- `rcssbase` は，他のモジュールによって使用されるベースコードである
- `rcssserver` は，実際のシミュレーションを実行する
- `rcsslogplayer` は，rcssserver によって生成されたログ (*.rcg ファイル) の再生を可能にする
- `rcssmonitor` と `rcssmonitorclassic` は，侵攻中の試合やログプレイヤが再生した試合の観戦を可能にする

rcsoccersim-*.tar.gz をダウンロードした場合，最初に，以下のコマンドでソースファイルへの展開を行う:

```
-> tar zxvf rcsoccersim-9.2.2.tar.gz
```

`rcsoccersim-*` ディレクトリが作成される．次に，作業ディレクトリを `rcsoccersim-*` へ変更する．このディレクトリは，以下のファイル郡を含む:

```
-> cd rcsoccersim-9.2.2
-> ls -a
.               Makefile.in    install-sh                rcsslogplayer
..              NEWS           mergeCL.awk               rcssmonitor
AUTHORS         NEWS.base      missing                   rcssmonitor_classic
BUGS            README         mkinstalldirs             rcssserver
COPYING         aclocal.m4     rcsoccersim               rel.awk
ChangeLog       configure      rcsoccersim-9.2.2.spec    spec.tmpl
ChangeLog.base  configure.in   rcssbase
Makefile.am     date.awk       rcsslogplay
```

常に `README` を最初に読むこと:

```
-> more README
```

`COPYING` ファイルは，ソフトウェアを使用または変更する場合のライセンスに関する詳細を含む．必ず目を通しておくこと．

```
-> more COPYING
```

3.3 クイックスタート

rcsoccersim-* ディレクトリで以下のコマンドを実行する:

```
-> ./configure
-> make
```

これにより，必要な実行バイナリのビルドが行われる．

`rcsoccersim-*/rcssserver/src/rcssserver` は，シミュレータサーバのバイナリである．シミュレータサーバは，実際のシミュレーションとクライアントプログラムとの通信を管理する．サンプルクライアントが `rcsoccersim-*/rcssserver/src/rcssclient` に用意されている．

シミュレータで起こっていることを確認するために，シミュレータモニタを起動する必要がある．モニタバイナリは，`rcsoccersim-*/rcssmonitor/src/rcssmonitor` である．

記録した試合を再生するためには，ログプレイヤを起動する必要がある．ログプレイヤバイナリは，`rcsoccersim-*/rcsslogplayer/src/rcsslogplayer` である．ログプレイヤは，試合のどの部分を見るかを制御する．しかし，再生の様子を実際に見るには，モニタ (rcssmonitor など) を起動する必要がある．

3.4 フルインストール

3.4.1 Configuring

RoboCup Soccer Simulator をビルドする前に，配布パッケージディレクトリ直下に置かれている `configure` スクリプトを実行する必要がある．

デフォルトの設定では，シミュレータは以下の場所へインストールされる:

/usr/local/bin	実行バイナリ
/usr/local/include	ヘッダファイル
/usr/local/lib	ライブラリ

シミュレータをデフォルトの場所へインストールするには，管理者の権限が必要かもしれない．インストール場所は，configure スクリプト の`--prefix=DIR` オプション (とそれに関連するオプション) によって変更可能である．利用可能なオプションの概要については，`configure --help` の実行結果を参照．

このパッケージに特有の多くの feature がある．これらのいくつかは，デフォルトで有効となっている．feature を有効にしたい場合は，オプション`--enable-`*`feature`* [=yes] を使用する．

feature を無効にするには，`--disable-`*`feature`* か`--enable-`*`feature`* [=no] を使用すれば良い．

`--enable-rcssclient`

シンプルなテストクライアントである `rcssclient` (cf. 2.1.4) を，`rcssserver` と一緒にビルドするかどうかを決定するオプション．この feature は，デフォルトで **on** である．

`--enable-client2dview`

このオプションは，`client2dview` のビルドを制御する．デフォルトで有効である．`rcssmonitor` は，サッカーモニタとしてだけでなく，他のデータの視覚化のためにも同様に使用することができる．`client2dview` は，モニタが表示可能なものを示すシンプルなプログラムである．詳細は，モニタの FAQ を読んでもらいたい．FAQ は，パッケージ内の `rcssmonitor` サブディレクトリ内に置かれている．

`--enable-fast_scanner`
このオプションを有効にすると，高速なビルドが実行されるが，非常に強大なスキャナが要求される．このオプションは，デフォルトで無効である．

3.4.2 Building

シミュレータの configure が成功すれば，後は `make` を実行するだけでソースをビルドできる．

3.4.3 インストール

シミュレータを完全にビルドできれば，`make install` を実行することで，各コンポーネントをインストールできるようになる．インストール場所は，configure で指定した場所か，指定していなければデフォルトの場所である．インストールする場所によって，特別な権限が必要になる場合がある．

3.4.4 アンインストール

パッケージの配布ディレクトリ内で `make uninstall` を実行することで，シミュレータを簡単に削除することができる．この作業は，インストールされた全てのファイルを削除するが，インストール過程で作成されたディレクトリは削除されない．

3.5 シミュレータの実行

サーバを起動するには，実行バイナリを含むディレクトリで

`./rcssserver`

とタイプするか，PATH が通った場所に実行バイナリがインストールされていれば，

`rcssserver`

とタイプすれば良い．

rcssserver はユーザのホームディレクトリ内に設定ファイルが存在するか同化を探す:

```
.rcssserver-server.conf
.rcssserver-player.conf
.rcssserver-landmark.xml
```

これらのファイルが存在しなければ，rcssserver によってファイルが作成され，デフォルトの値が書き込まれる．

`rcssserver` の起動オプションに `-sfile` と `-pfile` を使用することで，別の場所にある設定ファイルを指定することができる．指定したファイルが存在しない場合，それらは rcssserver によって作成され，デフォルトの値が書き込まれる．

シミュレータ内で起こっていることを確認するには，rcssserver と同様に，

`./rcssmonitor` または `rcssmonitor`

とタイプし，サッカーモニタを実行すればよい．

PATH の通った場所に実行バイナリがインストールされていれば，`rcsoccersim` スクリプトによって，サーバとモニタの両方を起動できる．この場合，`rcsoccersim` は PATH の

通った場所 (rcssserver と同じ場所) にインストールされている．このスクリプトは，サーバとモニタを起動し，ユーザがモニタを閉じれば自動的にサーバを停止させる．

サッカーサーバにおける試合を実際に開始するために，ユーザは各クライアントをサーバへ接続しなければならない (各チーム最大 11 人のプレイヤ+コーチ)．これらのクライアントの準備が整えば，サッカーモニタの `Kick Off` を押せば試合が始まる．あなたはまだ自分のクライアントを作成しないかもしれない．そのような場合，??節を読めば，他の RoboCupper が寄贈したチームを用いて試合をセットアップする方法の手順を知ることができる．

また，シミュレータの配布パッケージにはサンプルクライアント `rcssclient` が含まれている．`rcssclient` は，ncurses インターフェイス，または ncurses が利用できなければコマンドラインのインターフェイスを持っている．`-nogui` オプションを用いることで，明示的にコマンドラインインターフェイスを指定することも可能である．

`rcssclient` を実行すると，デフォルトパラメータ (host=localhost, port=6000) でサーバへの接続を試みる．もちろん，これらのパラメータはプログラム引数で変更可能である．クライアントが起動すると，ユーザはサーバへの接続を初期化しなければならない．これは，手作業で init コマンドをタイプし，エンターを押すことでなされる．よって，接続を初期化するには以下のようにタイプする:

```
(init MyTeam (version 9))
```

一方のチームが，“MyTeam” という名前になり，一体のプレイヤがサイドラインの側に現れることに気づくだろう．このプレイヤは，あなたが初期化したクライアントと一致する．更に，クライアントがターミナルに情報を出力していることにも気づくだろう．これは，クライアントがサーバから受信したメッセージである．

以下のテキスト (分かり易さのために，適宜改行を加えている) では，最初の 11 行は初期化に関するメッセージ[1]である．その他のデータはサーバがクライアントへ送信したセンサ情報である:

```
(init MyTeam (version 9))
(init l 2 before_kick_off)
(server_param (catch_ban_cycle 5)(clang_advice_win 1)
    (clang_define_win 1)(clang_del_win 1)(clang_info_win 1)
    (clang_mess_delay 50)(clang_mess_per_cycle 1)
    (clang_meta_win 1)(clang_rule_win 1)(clang_win_size 300)
    (coach_port 6001)(connect_wait 300)(drop_ball_time 0)
    (freeform_send_period 20)(freeform_wait_period 600)
    (game_log_compression 0)(game_log_version 3)
    (game_over_wait 100)(goalie_max_moves 2)(half_time -10)
    (hear_decay 1)(hear_inc 1)(hear_max 1)(keepaway_start -1)
    (kick_off_wait 100)(max_goal_kicks 3)(olcoach_port 6002)
    (point_to_ban 5)(point_to_duration 20)(port 6000)
    (recv_step 10)(say_coach_cnt_max 128)
    (say_coach_msg_size 128)(say_msg_size 10)
    (send_step 150)(send_vi_step 100)(sense_body_step 100)
```

[1] サーバからの応答は，クライアントが左サイドでプレイし，背番号が 1，プレイモードが before_kick_off であることを意味している．他の行は，現在のサーバパラメータとプレイヤタイププパラメータに対応している．

```
    (simulator_step 100)(slow_down_factor 1)(start_goal_l 0)
    (start_goal_r 0)(synch_micro_sleep 1)(synch_offset 60)
    (tackle_cycles 10)(text_log_compression 0)
    (game_log_dir "/home/thoward/data")
    (game_log_fixed_name "rcssserver")keepaway_log_dir "./")
    (keepaway_log_fixed_name "rcssserver")
    (landmark_file "~/.rcssserver-landmark.xml")
    (log_date_format "%Y%m%d%H%M-")(team_l_start "")
    (team_r_start "")(text_log_dir "/home/thoward/data")
    (text_log_fixed_name "rcssserver")(coach 0)
    (coach_w_referee 1)(old_coach_hear 0)(wind_none 0)
    (wind_random 0)(auto_mode 0)(back_passes 1)
    (forbid_kick_off_offside 1)(free_kick_faults 1)
    (fullstate_l 0)(fullstate_r 0)(game_log_dated 1)
    (game_log_fixed 1)(game_logging 1)(keepaway 0)
    (keepaway_log_dated 1)(keepaway_log_fixed 0)
    (keepaway_logging 1)(log_times 0)(profile 0)
    (proper_goal_kicks 0)(record_messages 0)(send_comms 0)
    (synch_mode 0)(team_actuator_noise 0)(text_log_dated 1)
    (text_log_fixed 1)(text_logging 1)(use_offside 1)
    (verbose 0)(audio_cut_dist 50)(ball_accel_max 2.7)
    (ball_decay 0.94)(ball_rand 0.05)(ball_size 0.085)
    (ball_speed_max 2.7)(ball_weight 0.2)(catch_probability 1)
    (catchable_area_l 2)(catchable_area_w 1)(ckick_margin 1)
    (control_radius 2)(dash_power_rate 0.006)(effort_dec 0.005)
    (effort_dec_thr 0.3)(effort_inc 0.01)(effort_inc_thr 0.6)
    (effort_init 0)(effort_min 0.6)(goal_width 14.02)
    (inertia_moment 5)(keepaway_length 20)(keepaway_width 20)
    (kick_power_rate 0.027)(kick_rand 0)(kick_rand_factor_l 1)
    (kick_rand_factor_r 1)(kickable_margin 0.7)(maxmoment 180)
    (maxneckang 90)(maxneckmoment 180)(maxpower 100)
    (minmoment -180)(minneckang -90)(minneckmoment -180)
    (minpower -100)(offside_active_area_size 2.5)
    (offside_kick_margin 9.15)(player_accel_max 1)
    (player_decay 0.4)(player_rand 0.1)(player_size 0.3)
    (player_speed_max 1)(player_weight 60)(prand_factor_l 1)
    (prand_factor_r 1)(quantize_step 0.1)(quantize_step_l 0.01)
    (recover_dec 0.002)(recover_dec_thr 0.3)(recover_min 0.5)
    (slowness_on_top_for_left_team 1)
    (slowness_on_top_for_right_team 1)(stamina_inc_max 45)
    (stamina_max 4000)(stopped_ball_vel 0.01)
    (tackle_back_dist 0.5)(tackle_dist 2.5)(tackle_exponent 6)
    (tackle_power_rate 0.027)(tackle_width 1.25)
    (visible_angle 90)(visible_distance 3)(wind_ang 0)
    (wind_dir 0)(wind_force 0)(wind_rand 0))
(player_param (player_types 7)(pt_max 3)(random_seed -1)
    (subs_max 3)(dash_power_rate_delta_max 0)
    (dash_power_rate_delta_min 0)
    (effort_max_delta_factor -0.002)
```

```
    (effort_min_delta_factor -0.002)
    (extra_stamina_delta_max 100)
    (extra_stamina_delta_min 0)
    (inertia_moment_delta_factor 25)
    (kick_rand_delta_factor 0.5)
    (kickable_margin_delta_max 0.2)
    (kickable_margin_delta_min 0)
    (new_dash_power_rate_delta_max 0.002)
    (new_dash_power_rate_delta_min 0)
    (new_stamina_inc_max_delta_factor -10000)
    (player_decay_delta_max 0.2)
    (player_decay_delta_min 0)
    (player_size_delta_factor -100)
    (player_speed_max_delta_max 0.2)
    (player_speed_max_delta_min 0)
    (stamina_inc_max_delta_factor 0))
(player_type (id 0)(player_speed_max 1)(stamina_inc_max 45)
    (player_decay 0.4)(inertia_moment 5)(dash_power_rate 0.006)
    (player_size 0.3)(kickable_margin 0.7)(kick_rand 0)
    (extra_stamina 0)(effort_max 1)(effort_min 0.6))
(player_type (id 1)(player_speed_max 1.1956)(stamina_inc_max 30.06)
    (player_decay 0.4554)(inertia_moment 6.385)(dash_power_rate 0.007494)
    (player_size 0.3)(kickable_margin 0.829)(kick_rand 0.0645)
    (extra_stamina 9.4)(effort_max 0.9812)(effort_min 0.5812))
(player_type (id 2)(player_speed_max 1.135)(stamina_inc_max 33.4)
    (player_decay 0.4292)(inertia_moment 5.73)(dash_power_rate 0.00716)
    (player_size 0.3)(kickable_margin 0.8198)(kick_rand 0.0599)
    (extra_stamina 31.3)(effort_max 0.9374)(effort_min 0.5374))
(player_type (id 3)(player_speed_max 1.1964)(stamina_inc_max 31.24)
    (player_decay 0.4664)(inertia_moment 6.66)(dash_power_rate 0.007376)
    (player_size 0.3)(kickable_margin 0.88)(kick_rand 0.09)
    (extra_stamina 47.1)(effort_max 0.9058)(effort_min 0.5058))
(player_type (id 4)(player_speed_max 1.151)(stamina_inc_max 37.8)
    (player_decay 0.45)(inertia_moment 6.25)(dash_power_rate 0.00672)
    (player_size 0.3)(kickable_margin 0.8838)(kick_rand 0.0919)
    (extra_stamina 44.1)(effort_max 0.9118)(effort_min 0.5118))
(player_type (id 5)(player_speed_max 1.1544)(stamina_inc_max 34.68)
    (player_decay 0.4352)(inertia_moment 5.88)(dash_power_rate 0.007032)
    (player_size 0.3)(kickable_margin 0.8052)(kick_rand 0.0526)
    (extra_stamina 47.1)(effort_max 0.9058)(effort_min 0.5058))
(player_type (id 6)(player_speed_max 1.193)(stamina_inc_max 36.7)
    (player_decay 0.4738)(inertia_moment 6.845)(dash_power_rate 0.00683)
    (player_size 0.3)(kickable_margin 0.885)(kick_rand 0.0925)
    (extra_stamina 92)(effort_max 0.816)(effort_min 0.416))
(sense_body 0 (view_mode high normal) (stamina 4000 1) (speed 0 0)
    (head_angle 0) (kick 0) (dash 0) (turn 0) (say 0) (turn_neck 0)
    (catch 0) (move 0) (change_view 0) (arm (movable 0) (expires 0)
    (target 0 0) (count 0)) (focus (target none) (count 0)) (tackle
    (expires 0) (count 0)))
```

```
(see 0 ((f c t) 6.7 27 0 0) ((f r t) 58.6 3) ((f g r b) 73 37)
    ((g r) 69.4 32) ((f g r t) 66 27) ((f p r c) 55.7 41)
    ((f p r t) 45.2 22) ((f t 0) 6.3 -18 0 0)
    ((f t r 10) 16.1 -7 0 0) ((f t r 20) 26 -4 0 0)
    ((f t r 30) 36.2 -3) ((f t r 40) 46.1 -2)
    ((f t r 50) 56.3 -2) ((f r 0) 73.7 30) ((f r t 10) 68.7 23)
    ((f r t 20) 66 15) ((f r t 30) 64.1 6) ((f r b 10) 79 37)
    ((f r b 20) 85.6 42))
(sense_body 0 (view_mode high normal) (stamina 4000 1) (speed 0 0)
    (head_angle 0) (kick 0) (dash 0) (turn 0) (say 0) (turn_neck 0)
    (catch 0) (move 0) (change_view 0) (arm (movable 0) (expires 0)
    (target 0 0) (count 0)) (focus (target none) (count 0)) (tackle
    (expires 0) (count 0)))
(see 0 ((f c t) 6.7 27 0 0) ((f r t) 58.6 3) ((f g r b) 73 37)
    ((g r) 69.4 32) ((f g r t) 66 27) ((f p r c) 55.7 41)
    ((f p r t) 45.2 22) ((f t 0) 6.3 -18 0 0)
    ((f t r 10) 16.1 -7 0 0) ((f t r 20) 26 -4 0 0)
    ((f t r 30) 36.2 -3) ((f t r 40) 46.1 -2)
    ((f t r 50) 56.3 -2) ((f r 0) 73.7 30) ((f r t 10) 68.7 23)
    ((f r t 20) 66 15) ((f r t 30) 64.1 6) ((f r b 10) 79 37)
    ((f r b 20) 85.6 42))
...
```

コマンド ((move 0 0) や (turn 45) など) をタイプすることで，プレイヤがサーバへ送信するコマンドを指示できる．これらのコマンドの結果は，サッカーモニタで確認することができるだろう．

3.6 サーバの停止方法

サーバを停止させる正しい手順は:

1. 全てのクライアント (プレイヤ) を停止させる
2. Quit ボタンをクリックして，全てのサッカーモニタを停止させる
3. サーバを実行したターミナルウインドウで CTRL-C を押してサーバを止める

この手順に従う場合，全ての可視の実行プロセスを止めるだけでなく，バックグラウンドで走っているかもしれない全ての関連プロセス (rcssserver など) の停止も確認できる．シミュレータを適切にシャットダウンしない場合に発生する問題は，異なるパラメータで起動しない限り，別プロセスでサーバを起動できないかもしれないことである．

更に，ctrl-c でシミュレータを停止しなければ，ログファイルが適切に閉じられず，リネームが正しく行われないだろう (ただし，これは圧縮ログファイルを使用している場合にのみ重要)．

注意: プロセスをその名前で停止することが有用かつ便利であることがある．kill コマンドの使用するには，停止させたいプロセス番号を ps コマンドで調べなければならない．より簡単に特定の名前のプロセスを全て停止させるには，killall コマンドを使用すれば良い．例えば，rcssserver という名前の全てのプロセスを停止させるには，“killall rcssserver” を実行するだけで充分である．

3.7 サポートされるプラットフォーム

Soccer Server はかなり多くの UNIX スタイルプラットフォームをサポートする．しかし，実際に動作するプラットフォームのリストは作成されていない．シミュレータ (バージョン番号でグループ分け済み) は，以下で示すプラットフォームで動作することが確認されている[2]:

- 9.2.2
  - SuSE 7.3 with gcc 2.95.3 or 3.2 (Tom Howard)
  - Windows 2000 with Cygwin with gcc 2.95.3 (Tom Howard)
  - SuSE 8.1 with gcc 3.2 (Jan Murray)
  - Debian 3.0 (woody) with gcc 2.95.4 (Jan Murray)
  - SuSE 7.0 Linux with gcc 2.95.2 (Kernel 2.4.16) (Goetz Schwandtner)
- 9.1.5
  - SuSE 8.1 with gcc 3.2 (Jan Murray)
  - Debian 3.0 (woody) with gcc 2.95.4 (Jan Murray)
  - SuSE 7.3 with gcc 2.95.3 or 3.2 (Tom Howard)
  - Windows 2000 with Cygwin with gcc 2.95.3 (Tom Howard)

あなたが上記リストに含まれないプラットフォームを持っており，特定のバージョンのシミュレータをその環境で動作させることに成功したならば，<`sserver-admin@lists.sf.net`> まで知らせてください．

3.8 トラブルシューティング

この節では，既知の問題をリストし，解決法もしくは少なくともあなたを正しい方向へ導くための情報の提供を試みます．

シミュレータを configure，ビルド，または実行する際に何らかの問題に遭遇し，その問題がここでレポートされていないものであれば，特に，問題を解決するためのパッチやヒントを提供してもらえるならば，RoboCup Soccer Simulator Web サイト，`http://sserver.sf.net/` からバグレポートを行ってもらいたい．

3.8.1 Libtool と Sed

この問題は `rcssserver` バージョン **9.3.2** で解決された．よって，この問題の解決には更新されたサーバを用いる方法が推奨される．

いくつかのバージョンの `libtool` は壊れており，ビルドにエラーを発生させる．この問題を修正するには，ストリームエディタ sed の位置を示す環境変数 `SED` を，手作業で設定すれば良い．現在，sed の位置は configure スクリプトによってチェックされるように

[2]リストされている名前はプラットフォームを検証した人物の名前である．

なっている．もし `SED` 変数がセットされていなければ，configure は以下のエラーを出力して終了する:

```
creating libtool

*************** ERROR *****************
The SED environment variable is not set.
Please set it to the sed excutable on your system.
```

あなたが使用しているシェルのタイプによっては，このエラーを修正するために，以下の作業を行わなければならない: **(t)csh** を使用しているなら，configure を実行する前に `setenv` によって変数をセットする．

```
-> setenv SED sed
```

bash を使用しているなら，以下のように変数をセットする:

```
-> export SED=sed
```

この後，再び configure を実行する．

```
-> ./configure
```

`configure` を以下のように実行してはいけない．

```
-> ./configure SED=sed
```

変数を `configure` へ渡すことはテストの回避にはなるが，問題の解決にはならないからである．

それでもエラーが発生する場合は，恐らく `sed` バイナリの位置が PATH に含まれていないだろう．この場合，上記手順内の `sed` を あなたの環境の `sed` バイナリへの絶対パス (通常 `/bin/sed`) に置き換えればよい．

SED 変数を常時セットしておくには，上記の変数設定コマンドを (t)csh の場合は `.cshrc` へ，bash の場合は `.bashrc` へ追加すればよい．

4 サッカーサーバ

4.1 物体

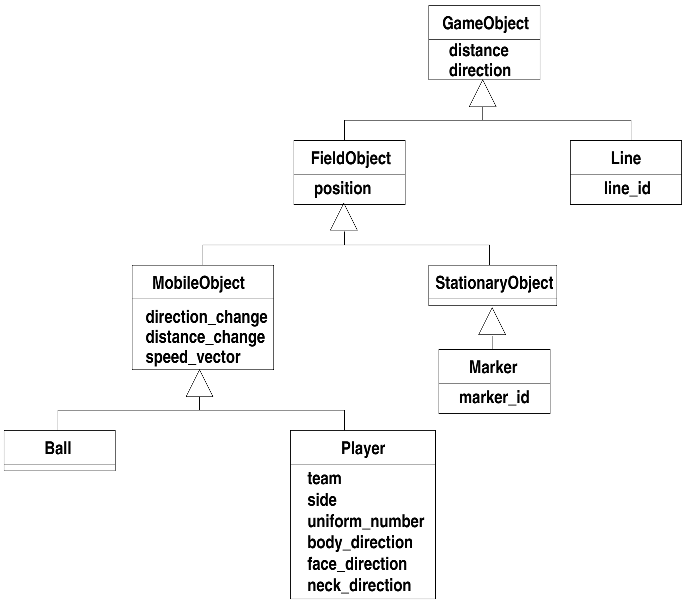

Figure 4.1: シミュレータ内の物体の UML ダイアグラム

4.2 プロトコル

4.2.1 コマンドプロトコル

接続，再接続，切断

クライアントからサーバへ	サーバからクライアントへ
(init *TeamName* [(version *VerNum*)] [(goalie)]) $TeamName ::= (-\|\_\|a-z\|A-Z\|0-9)^+$ $VerNum$ ::= the protocol version (e.g. 7.0)	**(init *Side Unum PlayMode*)** $Side$ ::= l | r $Unum$ ::= **1** ∼ 11 $PlayMode$ ::= プレイモードの一つ **(error no_more_team_or_player_or_goalie)**
(reconnect *TeamName Unum*) $TeamName ::= (-\|\_\|a-z\|A-Z\|0-9)^+$	**(reconnect *Side PlayMode*)** $Side$ ::= l | r $Unum$ ::= **1** ∼ **11** $PlayMode$ ::= プレイモードの一つ **(error no_more_team_or_player)** **(error reconnect)**
(bye)	

サーバとクライアントとの接続，または再接続が成功し，プロトコルバージョンが 7 以上であった場合，サーバは，server_param メッセージ，player_param メッセージ，player_type メッセージをクライアントへ送信する．バージョン 7.x の場合のフォーマットを以下に記す (バージョン 8 以降は，パラメータ名と値のペアをリストした状態で送信されるようになった)．最後に，プレイヤは交替されたプレイヤ情報のメッセージを受けとる (4.6 節を参照)．

- (server_param ***gwidth inertia_moment psize pdecay prand pweight pspeed_max paccel_max stamina_max stamina_inc recover_init recover_dthr recover_min recover_dec effort_init effort_dthr effort_min effort_dec effort_ithr effort_inc kick_rand team_actuator_noise prand_factor_l prand_factor_r kick_rand_factor_l kick_rand_factor_r bsize bdecay brand bweight bspeed_max baccel_max dprate kprate kmargin ctlradius ctlradius_width maxp minp maxm minm maxnm minnm maxn minn visangle visdist windir winforce winang winrand kickable_area catch_area_l catch_area_w catch_prob goalie_max_moves ckmargin offside_area win_no win_random say_cnt_max SayCoachMsgSize clang_win_size clang_define_win clang_meta_win clang_advice_win clang_info_win clang_mess_delay clang_mess_per_cycle half_time sim_st send_st recv_st sb_step lcm_st SayMsgSize hear_max hear_inc hear_decay cban_cycle slow_down_factor useoffside kickoffoffside offside_kick_margin audio_dist dist_qstep land_qstep dir_qstep dist_qstep_l dist_qstep_r land_qstep_l land_qstep_r dir_qstep_l dir_qstep_r CoachMode CwRMode old_hear sv_st start_goal_l start_goal_r fullstate_l fullstate_r drop_time***)
- (player_param ***player_types subs_max pt_max player_speed_max_delta_min player_speed_max_delta_max stamina_inc_max_delta_factor player_decay_delta_min***

player_decay_delta_max inertia_moment_delta_factor dash_power_rate_delta_min dash_power_rate_delta_max player_size_delta_factor kickable_margin_delta_min kickable_margin_delta_max kick_rand_delta_factor extra_stamina_delta_min extra_stamina_delta_max effort_max_delta_factor effort_min_delta_factor)

- プレイヤタイプごとに以下のメッセージが得られる
  (**player_type** ***id player_speed_max stamina_inc_max player_decay - inertia_moment dash_power_rate player_size kickable_margin kick_rand extra_stamina effort_max effort_min***)

クライアント制御

クライアントからサーバへ	1 サイクルにつき 1 回のみ
(catch *Direction*) *Direction* ::= ***minmoment*** ~ ***maxmoment*** 度	Yes
(change_view *Width Quality*) *Width* ::= narrow | normal | wide *Quality* ::= high | low	No
(dash *Power*) *Power* ::= ***minpower*** ~ ***maxpower*** 注: 後向きのダッシュ(負のパワー) は 2 倍のスタミナを消費する．	Yes
(kick *Power Direction*) *Power* ::= ***minpower*** ~ ***maxpower*** *Direction* ::= ***minmoment*** ~ ***maxmoment*** 度	Yes
(move *X Y*) *X* ::= **-52.5** ~ **52.5** *Y* ::= **-34** ~ **34**	Yes
(say *Message*) *Message* ::= メッセージ	No
(sense_body) サーバからの応答 (**sense_body** *Time* **(view_mode {high | low} {narrow | normal | wide})** **(stamina *Stamina Effort*)** **(speed *AmountOfSpeed DirectionOfSpeed*)** **(head_angle *HeadAngle*)** **(kick *KickCount*)** **(dash *DashCount*)** **(turn *TurnCount*)** **(say *SayCount*)** **(turn_neck *TurnNeckCount*)** **(catch *CatchCount*)** **(move *MoveCount*)** **(change_view *ChangeViewCount*)**)	No
(score) サーバからの応答 (**score** *Time OurScore TheirScore*)	No
(turn *Moment*) *Moment* ::= ***minmoment*** ~ ***maxmoment*** 度	Yes
(turn_neck *Angle*) *Angle* ::= ***minneckmoment*** ~ ***maxneckmoment*** 度 turn_neck は体の方向に相対である． turn, dash, kick などと同じサイクルで実行可能である．	Yes

サーバは上記のコマンドに対してエラーを返すかも知れない:

(error unknown_command)

(error illegal_command_form)

4.2.2 センサプロトコル

サーバからクライアントへ

(**hear** *Time Sender* ”*Message*”) (**hear** *Time Online_Coach Coach_Language_Message*)

Time ::= simulation cycle of the **soccerserver**
Sender ::= **online_coach_left** | **online_coach_right** | **coach** | **referee** | **self** | *Direction*
Direction ::= **-180** ∼**180 度**
Message ::= 文字列
Online_Coach ::= **online_coach_left** | **online_coach_right**
Coach_Language_Message ::= 標準コーチ言語の節を参照

(**see** *Time ObjInfo*$^{+}$)

Time ::= **soccerserver** のサイクル
ObjInfo ::= (***ObjName Distance Direction DistChange DirChange BodyFacingDir HeadFacingDir***)
| (***ObjName Distance Direction DistChange DirChange***
| (***ObjName Distance Direction***)
| (***ObjName Direction***)
ObjName ::= (**p** [”*Teamname*” [*UniformNumber* [**goalie**]]])
| (**b**)
| (**g** [**l**|**r**])
| (**f c**)
| (**f** [**l**|**c**|**r**] [**t**|**b**])
| (**f p** [**l**|**r**] [**t**|**c**|**b**])
| (**f g** [**l**|**r**] [**t**|**b**])
| (**f** [**l**|**r**|**t**|**b**] **0**)
| (**f** [**t**|**b**] [**l**|**r**] [**10**|**20**|**30**|**40**|**50**])
| (**f** [**l**|**r**] [**t**|**b**] [**10**|**20**|**30**])
| (**l** [**l**|**r**|**t**|**b**])
| (**B**)
| (**F**)
| (**G**)
| (**P**)
Distance ::= 正の実数
Direction ::= **-180** ∼**180 度**
DistChange ::= 実数
DirChange ::= 実数
HeadFaceDir ::= **-180** ∼**180 度**
BodyFaceDir ::= **-180** ∼**180 度**
Teamname ::= 文字列
UniformNumber ::= 1 ∼11

(**sense_body** *Time*
(**view_mode** {**high** | **low**} {**narrow** | **normal** | **wide**})
(**stamina *Stamina Effort***)
(**speed *AmountOfSpeed DirectionOfSpeed***)
(**head_angle *HeadAngle***)
(**kick *KickCount***)
(**dash *DashCount***)
(**turn *TurnCount***)
(**say *SayCount***)
(**turn_neck *TurnNeckCount***)
(**catch *CatchCount***)
(**move *MoveCount***)
(**change_view *ChangeViewCount***))

Time ::= **soccerserver** のサイクル
Stamina ::= 正の実数
Effort ::= 正の実数
AmountOfSpeed ::= 正の実数
DirectionOfSpeed ::= **-180** ∼**180 度**
HeadAngle ::= **-180** ∼**180 度**
**Count* ::= 正の整数

4.3 センサモデル

RoboCup サッカーシミュレータのプレイヤエージェントは，三つの異なるセンサを持つ．聴覚センサは，審判，コーチ，そして他のプレイヤからのメッセージを検出する．視覚センサは，プレイヤの現在の視界内にある物体への距離と方向のような，フィールドに関する情報を検出する．視覚センサは，プレイヤの近傍だが背後に存在する物体を“見る”ことによって，接近センサとしても働く．Body センサは，プレイヤの現在の“物理的な”状態 (スタミナ，スピード，首の角度など) を検出する．センサ情報によって，エージェントは環境の状況を得ることができる．

4.3.1 聴覚センサモデル

聴覚センサメッセージは，プレイヤかコーチが say コマンドを実行した時に送信される．審判からのメッセージも聴覚メッセージとして送信される．すべてのメッセージは個別に送信される．

サッカーサーバから送られてくる聴覚センサメッセージのフォーマットは以下のようになる:

(hear *Time* *Sender* "*Message*")

Time 現在のサイクルを示す．

Sender 送信者が他のプレイヤの場合，その相対方向．それ以外の場合，以下のうちの一つ:

- **self**: 送信者は自分自身．
- **referee**: 送信者は審判．
- **online_coach_left** or **online_coach_right**: 送信者はオンラインコーチ．

Message メッセージ本体．最大長は ***say_msg_size*** バイト．審判からのメッセージについての詳細は 4.7.1 節に記述されている．

聴覚センサに影響するサーバパラメータは表 4.1 に記述されている．

`server.conf` 内のパラメータ	値
audio_cut_dist	50.0
hear_max	1
hear_inc	1
hear_decay	1
say_msg_size	10

Table 4.1: 聴覚センサに関するパラメータ

聴覚センサの許容量

プレイヤの hear capacity が ***hear_decay*** 以上である場合のみ，プレイヤはメッセージを聞くことができる．プレイヤがメッセージを聞いた時，プレイヤの hear capacity が ***hear_decay*** だけ減少させせられるからである．hear capacity は毎サイクル ***hear_inc*** だけ回復し，その最大値は ***hear_max*** である．コミュニケーションチャネルに負荷をかけて他チームのコミュニケーションを妨害することを避けるために，プレイヤはチーム毎に個別の hear capacity を持つ．バージョン 7 以前では，2 サイクルに一回，各チームから 1 メッセージずつ聞ける設定であった．バージョン 8 以降の server.conf では，1 サイクルにつき各チームから 1 メッセージずつ聞ける設定になっている．

プレイヤの許容量を越えるメッセージが同時に届いた場合，プレイヤが実際に聞けるメッセージがどれになるかは定義されていない (過去の実装では，最も早くサーバに受信されたメッセージが選ばれていた．バージョン 9.4.5 では同サイクル内に届いたメッセージ内からランダムに選択されるようになっている)．このルールは審判と自分自身からのメッセージには適用されない．言い替えれば，プレイヤがメッセージを say することと，他のプレイヤからのメッセージを聞くことを，同じサイクルで行なうことができる．

コミュニケーションの範囲

プレイヤが発したメッセージは，そのプレイヤから ***audio_cut_dist*** メートル以内にいるプレイヤにのみ伝えられる．例えば，自陣ゴール近くにいるディフェンダーは，ゴールキーパからのメッセージを聞くことはできるが，敵ゴール近くにいるストライカーからのメッセージを聞くことはできない．審判からのメッセージは，常に全てのプレイヤが聞くことができる．

4.3.2 視覚センサモデル

現在，プレイヤに見えている物体の情報は視覚センサによって伝えられる．情報は ***sense_step***(現在，150) ミリ秒ごとにプレイヤへと自動的に送られる．

サーバから送られてくる視覚情報は，次のような基本フォーマットを持つ:

(**see** *ObjName Distance Direction DistChng DirChng BodyDir HeadDir*)

各要素のフォーマットは，

$$
\begin{array}{lcl}
\mathit{ObjName} & ::= & (\textbf{p}\ \text{"}\mathit{Teamname}\text{"}\ \mathit{UniformNumber}\ \textbf{goalie}) \\
& & \mid \textbf{(g [l|r])} \\
& & \mid \textbf{(b)} \\
& & \mid \textbf{(f c)} \\
& & \mid \textbf{(f [l|c|r] [t|b])} \\
& & \mid \textbf{(f p [l|r] [t|c|b])} \\
& & \mid \textbf{(f g [l|r] [t|b])} \\
& & \mid \textbf{(f [l|r|t|b] 0)} \\
& & \mid \textbf{(f [t|b] [l|r] [10|20|30|40|50])} \\
& & \mid \textbf{(f [l|r] [t|b] [10|20|30])} \\
& & \mid \textbf{(l [l|r|t|b])}
\end{array}
$$

Distance, *Direction*, *DistChng* そして *DirChng* は，次の方法で計算される:

$$
\begin{aligned}
p_{rx} &= p_{xt} - p_{xo} && (4.1)\\
p_{ry} &= p_{yt} - p_{yo} && (4.2)\\
v_{rx} &= v_{xt} - v_{xo} && (4.3)\\
v_{ry} &= v_{yt} - v_{yo} && (4.4)\\
\mathit{Distance} &= \sqrt{p_{rx}^2 + p_{ry}^2} && (4.5)\\
\mathit{Direction} &= \arctan(p_{ry}/p_{rx}) - a_o && (4.6)\\
e_{rx} &= p_{rx}/\mathit{Distance} && (4.7)\\
e_{ry} &= p_{ry}/\mathit{Distance} && (4.8)\\
\mathit{DistChng} &= (v_{rx} * e_{rx}) + (v_{ry} * e_{ry}) && (4.9)\\
\mathit{DirChng} &= [(-(v_{rx} * e_{ry}) + (v_{ry} * e_{rx}))/\mathit{Distance}] * (180/\pi) && (4.10)\\
\mathit{BodyDir} &= \mathit{PlayerBodyDir} - \mathit{AgentBodyDir} - \mathit{AgentHeadDir} && (4.11)\\
\mathit{HeadDir} &= \mathit{PlayerHeadDir} - \mathit{AgentBodyDir} - \mathit{AgentHeadDir} && (4.12)
\end{aligned}
$$

(p_{xt}, p_{yt}) は対象物体の絶対位置座標，(p_{xo}, p_{yo}) は観測者プレイヤの絶対位置座標，(v_{xt}, v_{yt}) は対象物体の絶対速度，(v_{xo}, v_{yo}) は観測者プレイヤの絶対速度，そして，a_o は観測者プレイヤの視界方向である．プレイヤの視界の絶対方向は，そのプレイヤの *BodyDir* と *HeadDir* の和である．これに加え，(p_{rx}, p_{ry}) は対象の相対位置座標，(v_{rx}, v_{ry}) は対象の相対速度，(e_{rx}, e_{ry}) は相対位置ベクトルに並行な単位ベクトルである．*BodyDir* と *HeadDir* は，対象オブジェクトがプレイヤである場合にのみ含まれ，それぞれ，対象プレイヤの体と首の方向である．これらの方向は，観測者プレイヤの視界方向に対して相対な値で得られる．例えば，対象プレイヤと観測者プレイヤが同じ方向に視界を向けていれば，*HeadDir* の値は 0 になる．

物体 **(goal r)** は，右サイドのゴールラインの中心として解釈される．**(f c)** は，フィールド中央にあるフラッグである．**(f l b)** はフィールド左下隅，**(f p l b)** は左サイドのペナルティエリアボックス右下隅に存在するフラッグである．**(f g l b)** は，左サイドのゴールの右ポストに位置するフラッグである．残りのタイプのフラッグは全て，フィールドの 5 メートル外側に存在する．例えば，**(f t l 20)** は，上側のサイドラインから 5 メートル，

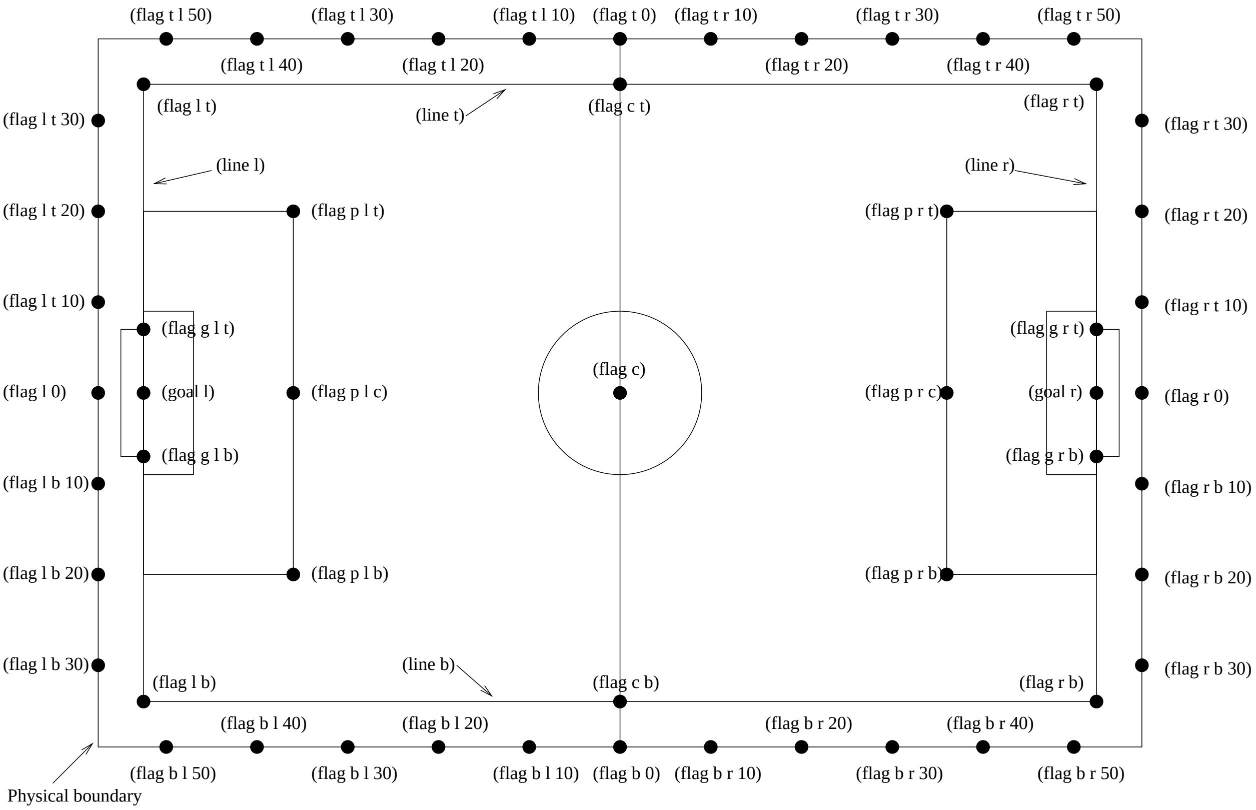

Figure 4.2: 仮想フィールド上のフラッグとライン

センターラインから左に 20 メートルの位置に存在する．同様に，**(f r b 10)** は，右のゴールラインから 5 メートル，右ゴールの中央から下へ 10 メートルの位置に存在する．更に，**(f b 0)** は，下側のサイドラインの中央から下に 5 メートルの位置に存在する．

(l ...) というメッセージの場合，$Distance$ はプレイヤの視界方向と対象ラインとの交点までの距離である．$Direction$ は，対象ラインの方向である．

視界の範囲

プレイヤが視認可能な領域は，いくつかの要素に従って決定される．まず第一に，シミュレータでは，視覚情報の基本送信間隔を決定する ***sense_step*** パラメータと，プレイヤの通常の視界コーン角度の大きさを決定する ***visible_angle*** パラメータが設定されている．現在のデフォルト値はそれぞれ，150 ミリ秒と 90 度である．

プレイヤは，$ViewWidth$ と $ViewQuality$ を変えることによって，視覚情報の頻度と品質を変化させることができる．

現在の ***view_frequency*** と ***view_angle*** の計算には，式 4.13 と式 4.14 が用いられる．

$$\textit{\textbf{view\_frequency}} = \textit{\textbf{sense\_step}} * \textit{\textbf{view\_quality\_factor}} * \textit{\textbf{view\_width\_factor}} \quad (4.13)$$

view_quality_factor は，$ViewQuality$ が ***high*** のときに 1，***low*** のときに 0.5 となる．***view_width_factor*** は，$ViewWidth$ が ***narrow*** のときに 2，***normal*** のときに 1，***wide*** のときに 0.5 となる．

$$\boldsymbol{view\_angle} = \boldsymbol{visible\_angle} * \boldsymbol{view\_width\_factor} \quad (4.14)$$

view_width_factor は，$ViewWidth$ が ***narrow*** のときに 0.5, ***normal*** のときに 1, ***wide*** のときに 2 となる．

プレイヤは，物体との距離が ***visible_distance*** メートル以内に存在するときにも，その物体を “見る” ことができる．物体がこの距離内に存在するが視界コーン内には存在しない場合，観測者プレイヤはその物体のタイプ (ボール，プレイヤ，ゴール，フラッグ) しか知ることができず，物体の正確な名前を知ることはできない．この場合，物体の名前として，“**B**”, “**P**”, “**G**”, “**F**” といった大文字の名前が用いられる．(視界コーン内であれば，物体の名前は，“**b**”, “**p**”, “**g**”, “**f**” という小文字になる)

次の例と図 4.3 は，[19] から採用された．

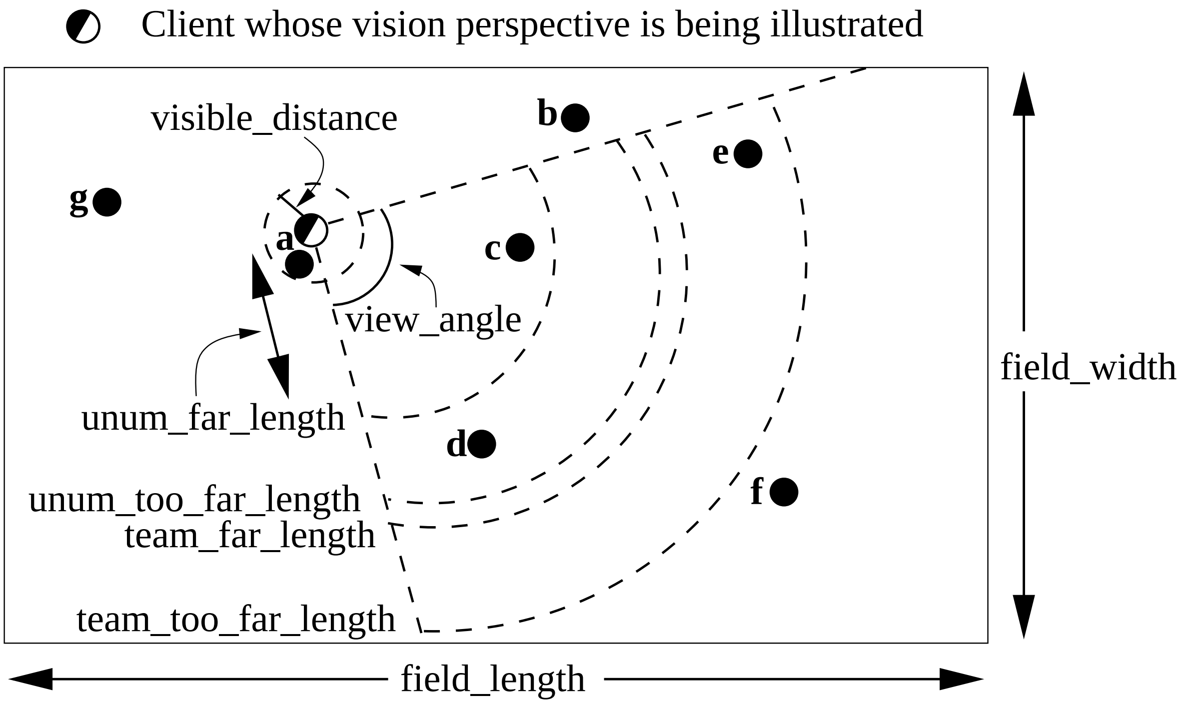

Figure 4.3: サッカーサーバ内での個々のエージェントの視界の範囲．観測者エージェントは白黒の円で表されている．白の半円がその前方である．黒の円は観測者エージェント以外の物体を表す．観測者エージェントは，視界方向から ***view_angle***°/2 以内，または，***visible_distance*** 以内の距離に存在する物体のみを見ることができる．***unum_far_length***, ***unum_too_far_length***, ***team_far_length***, ***team_too_far_length*** は，プレイヤの識別精度に影響を与える．[19] より引用．

view_angle パラメータの意味を，図 4.3 に示す．この図では，観測者エージェントは

白黒の円で表されている．白の半円がその前方である．黒の円は観測者エージェント以外の物体を表す．観測者エージェントは，視界方向から $\boldsymbol{view\_angle}°/2$ 以内，または，$\boldsymbol{visible\_distance}$ 以内の距離に存在する物体のみを見ることができる．この図の場合，物体 b と g を見ることはできない．しかし，他の物体は全て見ることができる．

物体 f は，観測者エージェントの直接前方にいる．その方向は，0 度として報告されるだろう．物体 e の方向は，おおよそ-40° として，物体 d は，おおよそ 20° として報告されるだろう.

図 4.3 にも示されているように，プレイヤに関する情報の精度は，対象プレイヤとの距離によって変化する．充分近いプレイヤに関しては，対象プレイヤのチームと背番号のいずれもが報告される．しかしながら，距離が増加するとまず背番号が見え難くなり，更に遠く離れるとチーム名さえも見え難くなる．サーバ内部では，$\boldsymbol{unum\_far\_length} \leq \boldsymbol{unum\_too\_far\_length} \leq \boldsymbol{team\_far\_length} \leq \boldsymbol{team\_too\_far\_length}$ と想定されている．対象プレイヤとの距離が $dist$ であるとすると:

- $dist \leq \boldsymbol{unum\_far\_length}$ であれば，背番号とチーム名の両方を見ることができる．
- $\boldsymbol{unum\_far\_length} < dist < \boldsymbol{unum\_too\_far\_length}$ であれば，チーム名は常に見ることができる．しかし，$dist$ が増加するに従って，背番号が見える確率は 1 から 0 へ線形に減少する．
- $dist \geq \boldsymbol{unum\_too\_far\_length}$ であれば，背番号は見えない．
- $dist \leq \boldsymbol{team\_far\_length}$ であれば，チーム名は見える．
- $\boldsymbol{team\_far\_length} < dist < \boldsymbol{team\_too\_far\_length}$ であれば，$dist$ が増加するに従って，チーム名が見える確率は 1 から 0 へ線形に減少する．
- $dist \geq \boldsymbol{team\_too\_far\_length}$ であれば，チーム名は見えない．

例えば，図 4.3 において，ラベルづけされた円を全てプレイヤとする．このとき，プレイヤ c は，チーム名と背番号の両方を識別されるだろう．プレイヤ d は，チーム名は確実に識別されるが，背番号に関しては 50%の識別確率となるだろう．プレイヤ e は，チーム名の識別確率が 25%であるだけで，背番号は完全に不明となる．プレイヤ f は，単に無名のプレイヤとして認識されるのみだろう．

視覚センサのノイズモデル

視覚センサデータにノイズを導入するために，サーバから送られる値は離散化される．例えば，物体の距離の値は，その物体がボールかプレイヤ（移動物体）の場合，以下の方法で離散化される:

$$d' = \mathrm{Quantize}(\exp(\mathrm{Quantize}(\log(d), \boldsymbol{quantize\_step})), 0.1) \tag{4.15}$$

d は正確な距離であり，d' は離散化された距離の値である．そして，

$$\mathrm{Quantize}(V, Q) = \mathrm{ceiling}(V/Q) \cdot Q \tag{4.16}$$

server.conf 内のパラメータ	値
sense_step	150
visible_angle	90.0
visible_distance	3.0
unum_far_length[a]	20.0
unum_too_far_length[a]	40.0
team_far_length[a]	40.0
team_too_far_length[a]	60.0
quantize_step	0.1
quantize_step_l	0.01

[a] server.conf には無いが，サーバに組み込まれている

Table 4.2: 視覚センサに関するパラメータ

この式は，非常に遠く離れた物体の正確な位置をプレイヤが知ることができない，ということを意味する．例えば，距離が約 100.0m であればノイズの最大値は約 10.0m になり，距離が 10.0m 以下であればノイズは 1.0.m 以下である．

フラッグ，ゴールとライン（静止物体）の場合，距離の離散化は以下の方法で行われる．

$$d' = \mathrm{Quantize}(\exp(\mathrm{Quantize}(\log(d), \boldsymbol{quantize\_step\_l})), 0.1) \tag{4.17}$$

4.3.3 ボディセンサモデル

Body センサは，プレイヤの現在の "物理的な" 状態を報告する．Body センサ情報は，***sense_body_step***(現在，100) ミリ秒毎に，自動的にプレイヤへと送られる．

Body センサ情報のフォーマットは:

(**sense_body** *Time*
(view_mode ViewQuality ViewWidth)
(stamina Stamina Effort)
(speed AmountOfSpeed DirectionOfSpeed)
(head_angle HeadDirection)
(kick KickCount)
(dash DashCount)
(turn TurnCount)
(say SayCount)
(turn_neck TurnNeckCount)
(catch CatchCount)
(move MoveCount)
(change_view ChangeViewCount))

ViewQuality は，***high*** と ***low*** のいずれか．

ViewWidth は，**narrow**, **normal**, **wide** のいずれか．

AmountOfSpeed は，プレイヤのスピードの概算．

DirectionOfSpeed は，プレイヤの速度の方向の概算．

HeadDirection は，プレイヤの首の相対的な方向．

The *Count* 変数は，各コマンドがサーバによって実行された総数．例えば，$DashCount = 134$ は，プレイヤが **dash** コマンドをこれまでに 134 回実行したことを意味する．

各パラメータの意味は，実際に使用される場所で解説されている．例えば，*ViewQuality* と *ViewWidth* パラメータは， 4.3.2 節で解説されている．

Body センサに影響するサーバパラメータは，表 4.3 に記述されている．

`server.conf` 内のパラメータ	値
sense_body_step	100

Table 4.3: Body センサに関するパラメータ

4.4 移動モデル

各シミュレーションサイクルにおいて，物体の移動の計算は以下のように行なわれる:

$$\begin{aligned}(u_x^{t+1}, u_y^{t+1}) &= (v_x^t, v_y^t) + (a_x^t, a_y^t):\ \text{加速} \\ (p_x^{t+1}, p_y^{t+1}) &= (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}):\ \text{移動} \\ (v_x^{t+1}, v_y^{t+1}) &= \mathrm{decay} \times (u_x^{t+1}, u_y^{t+1}):\ \text{速度減衰} \\ (a_x^{t+1}, a_y^{t+1}) &= (0,0):\ \text{加速度リセット}\end{aligned} \tag{4.18}$$

(p_x^t, p_y^t) と (v_x^t, v_y^t) は，それぞれ，サイクル t における物体の位置と速度である．decay は，**ball_decay** や **player_decay** で指定される，速度減衰のパラメータである．(a_x^t, a_y^t) は物体の加速度で，**dash** コマンド (プレイヤの場合) や **kick** コマンド (ボールの場合) の *Power* パラメータから，以下のように求められる:

$$(a_x^t, a_y^t) = Power \times power\_rate \times (\cos(\theta^t), \sin(\theta^t))$$

power_rate は，**dash_power_rate** か，または 4.5.3 節で述べられている **kick_power_rate** から計算される値である．θ^t は，サイクル t における物体の方向である．プレイヤの場合，これは単にプレイヤの体の方向である．ボールの場合，その方向は以下のように求められる:

$$\theta^t_{\mathrm{ball}} = \theta^t_{\mathrm{kicker}} + Direction$$

θ^t_{ball} はボールの方向，$\theta^t_{\mathrm{kicker}}$ はキックしたプレイヤの体の方向である．*Direction* は **kick** コマンドの第 2 パラメータである．

4.4.1 移動ノイズモデル

現実世界のように物体の動きに予期不能なものにするために，物体の移動とコマンドのパラメータにノイズが加えられる．

物体の移動に伴って，式. 4.18 へ以下のようにノイズが加えられる．

$$(u_x^{t+1}, u_y^{t+1}) \quad = \quad (v_x^t, v_y^t) + (a_x^t, a_y^t) + (\tilde{r}\mathrm{rmax}, \tilde{r}\mathrm{rmax})$$

$\tilde{r}_{\mathrm{max}}$ は，分散が $[-\mathrm{rmax}, \mathrm{rmax}]$ の範囲で一様な乱数である．rmax は，物体の速度の大きさに依存して，以下のように求められるパラメータである:

$$\mathrm{rmax} \quad = \quad \mathrm{rand} \cdot |(v_x^t, v_y^t)|$$

rand は，**player_rand** または **ball_rand** によって指定されるパラメータである．

コマンド引数の *Power* や *Moment* へのノイズは，以下のように加えられる:

$$argument \quad = \quad (1 + \tilde{r}_{\mathrm{rand}}) \cdot argument$$

4.4.2 衝突モデル

シミュレーションサイクル終了時に二つの物体が重なっていれば，物体は互いに重ならない位置まで移動させられる．そして，速度が-0.1 倍される．シミュレーションサイクル終了時に物体が重ならない限り，衝突は起こらず，すり抜けることができる点に注意．

4.5 アクションモデル

4.5.1 Catch モデル

キーパはボールをキャッチする能力を持つ唯一のプレイヤである．プレイモードが ‘`play_on`’ の時，キーパとボールが自陣ペナルティエリア内にあり，ボールがキャッチャブルエリア内にあれば，キーパは任意の方向にあるボールをキャッチできる．キーパが φ の方向へキャッチを行なった場合，φ 方向に長さ ***catchable_area_l***，幅 ***catchable_area_w*** の矩形の領域 (図 4.4 参照) がキャッチャブルエリアとなる．ボールがその領域内に入っていれば，***catch_probability*** の確率でキャッチされる．catch コマンドに関するパラメータについては，表 4.4 を参照．

catch コマンドが失敗した場合，***catch_ban_cycle*** サイクル経過するまでは次の catch コマンドを実行できない．この間に実行された catch コマンドは，何の効果も起こさない．キーパがボールキャッチに成功した場合，プレイモードはまず ‘`goalie_catch_ball_[l|r]`’ へと変更され，次に同じサイクル内で ‘`free_kick_[l|r]`’ へと変更される．キーパがボールをキャッチすると，**move** コマンドによってボールを持ったままペナルティエリア内を移動できるようになる．ただし，move 後にボールをペナルティキックエリアの外に出してしまうと，‘`catch_fault_[l|r]`’ が取られ，相手チームへフリーキックが与えられる．ボールをキックするまでに使用可能な **move** コマンドの回数は，***goalie_max_moves*** 回である．それ以上 **move** コマンドを実行しようとしても何の効果も得られず，サーバから (`error too_many_moves`) という応答が得られる．短いキックを行なって再びボールをキャッチし，移動を繰り返すといったプレイは非紳士的なプレイとみなされることに注意．

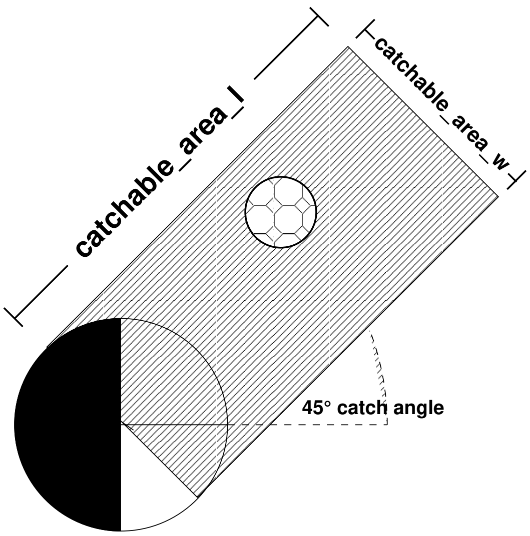

Figure 4.4: (catch -45) を実行した場合のキャッチャブルエリア

`server.conf` 内のパラメータ	値
catchable_area_l	2.0
catchable_area_w	1.0
catch_probability	1.0
catch_ban_cycle	5
goalie_max_moves	2

Table 4.4: catch コマンドに関するパラメータ

4.5.2 Dash モデル (スタミナモデル含む)

Dash モデル

dash コマンドは，プレイヤをその体の方向へ加速するために使用される．**dash** は加速度のパラメータとして *power* を取る．*power* の有効範囲は `server.conf` で設定され，***min-power*** と ***maxpower*** によって指定される．ダッシュモデルの現在のパラメータ値については，表 4.5 を参照．

各プレイヤは一定量のスタミナを持っており，**dash** コマンドによってスタミナは消費される．各ハーフの最初に，プレイヤのスタミナ値は ***stamina_max*** にセットされる．プレイヤは前方へ加速した場合 ($power > 0$)，スタミナの減少量は *power* である．後方へ加速した場合 ($power < 0$)，プレイヤの支払う代価は大きくなり，スタミナの減少量は $-2 \cdot power$ となる．プレイヤのスタミナが **dash** に必要とされる量よりも少ない場合，残りスタミナで可能な数値まで *power* が減らされる．ヘテロジニアスプレイヤは extra stamina を持っており，必要とされるスタミナが不足している場合には， extra stamina 分のスタミナを追加で使うことができる．追加スタミナ量の大きさは，プレイヤタイプとパラメータ sparamextra_stamina_delta_min，***extra_stamina_delta_max*** に依存している．

スタミナ減少後，サーバは **dash** コマンドに対する有効なダッシュパワー*: effectivedash power* を計算する．有効なダッシュパワー *edp* は，***dash_power_rate*** とプレイヤの現在の effort によって決定される．プレイヤの effort の範囲は ***effort_min*** と ***effort_max*** の間となり，プレイヤのスタミナ管理によって変化する．

$$\mathrm{edp} = \mathrm{effort} \cdot \mathrm{dash\_power\_rate} \cdot \mathrm{power} \tag{4.19}$$

edp とプレイヤの現在の体の向きがベクトルに変換され，プレイヤの現在の加速度ベクトル $\vec{a}_n$ へと加算される．(通常，プレイヤは 1 サイクルに一回しかダッシュできず，ダッシュ以外の方法で加速を得ることは無いため，$\vec{a}_n$ は事前に 0 にセットされている[1])．

シミュレーションステップ n から $n+1$ への移動において，加速度 $\vec{a}_n$ が適用される:

1. $\vec{a}_n$ の長さの最大値は ***player_accel_max*** へと正規化される．
2. $\vec{a}_n$ が現在のプレイヤのスピード $\vec{v}_n$ へと加算される. $\vec{v}_n$ の長さの最大値は ***player_speed_max*** へと正規化される．
3. ノイズ $\vec{n}$ と風 $\vec{w}$ が $\vec{v}_n$ に加算される．ノイズと風に関するパラメータは `server.conf` によって変更可能である．風に関するパラメータは ***wind_force***， ***wind_dir***，***wind_rand*** である．現在の設定では，フィールド上に風は全く存在しない．ノイズに関連するパラメータは ***player_rand*** である．ノイズベクトルの XY 成分は，$[-|\vec{v}_n| \cdot \mathrm{player\_rand} \ldots |\vec{v}_n| \cdot \mathrm{player\_rand}]$ の範囲の値を取り得る．
4. プレイヤの新しい位置 $\vec{p}_{n+1}$ は，古い位置 $\vec{p}_n$ に速度ベクトル $\vec{v}_n$ を加算した値になる．
5. プレイヤの速度へ ***player_decay*** が適用される: $\vec{v}_{n+1} = \vec{v}_n \cdot \mathrm{player\_decay}$. 加速度 $\vec{a}_{n+1}$ が 0 にセットされる．

[1] is that so?

基本パラメータ server.conf		ヘテロジニアスプレイヤのパラメータ player.conf		
名前	値	名前	値	**Range**
minpower	-100			
maxpower	100			
stamina_max	4000			
stamina_inc_max	45	***new_stamina_inc_max_delta_factor*** ***new_dash_power_rate_delta_min*** ***new_dash_power_rate_delta_max***	-10000.0 0.0 0.002	25 — 45
extra_stamina[a]	0.0	***extra_stamina_delta_min*** ***extra_stamina_delta_max***	0.0 100.0	0.0 — 100.0
dash_power_rate	0.006	***new_dash_power_rate_delta_min*** ***new_dash_power_rate_delta_max***	0.0 0.002	0.006 — 0.008
effort_min	0.6	***effort_min_delta_factor*** ***extra_stamina_delta_min*** ***extra_stamina_delta_max***	-0.002 0.0 100.0	0.4 — 0.6
effort_max[a]	1.0	***effort_max_delta_factor*** ***extra_stamina_delta_min*** ***extra_stamina_delta_max***	-0.002 0.0 100.0	0.8 — 1.0
effort_dec_thr	0.3			
effort_dec	0.005			
effort_inc_thr	0.6			
effort_inc	0.01			
recover_dec_thr	0.3			
recover_dec	0.002			
recover_min	0.5			
player_accel_max	1.2			
player_speed_max	1.2	***player_speed_max_delta_min*** ***player_speed_max_delta_max***	0.0 0.0	1.2 — 1.2
player_rand	0.1			
wind_force	0.0			
wind_dir	0.0			
wind_rand	0.0			
player_decay	0.4	***player_decay_delta_min*** ***player_decay_delta_max***	0.0 0.2	0.4 — 0.6

[a] `server.conf` には無いが，サーバには組み込まれている．

Table 4.5: Dash と Stamina に関するパラメータ (バージョン 9)

Stamina モデル

プレイヤのスタミナに関する重要な変数として，*stamina*，*recovery*，*effort* の 3 つがある．*stamina* はダッシュを行なうと減少し，毎サイクル少しずつ回復する．*recovery* は *stamina* の回復量に関係し，*effort* はダッシュの効果に影響を及ぼす (前節の式 4.19 を参照)．スタミナモデルに関する重要なパラメータは `server.conf` と `player.conf` で変更可能である．表 4.5 も参照．基本的に，図 4.5 で述べられているアルゴリズム次のようになる: *stamina* がある閾値を下回っていれば，*effort* と *recovery* は減少する (ただし，最小値以下にはならない)．*stamina* がある閾値を上回っていれば，*effort* は増加する (ただし，最大値以上にはならない)．The *recovery* は各ハーフ開始時に 1.0 にリセットされるのみで，試合中には回復しない．

{ スタミナが recovery 減少の閾値位かであれば，recovery が減少する }
if stamina $\leq$ recover_dec_thr $\cdot$ stamina_max **then**
  if recovery $>$ recover_min **then**
    recovery $\leftarrow$ recovery $-$ recover_dec
  end if
end if

{ スタミナが effort 減少の閾値以下であれば，effort が減少する }
if stamina $\leq$ effort_dec_thr $\cdot$ stamina_max **then**
  if effort $>$ effort_min **then**
    effort $\leftarrow$ effort $-$ effort_dec
  end if
  effort $\leftarrow$ max(effort, effort_min)
end if

{ スタミナが effort 増加の閾値以上であれば，effor が増加する }
if stamina $\geq$ effort_inc_thr $\cdot$ stamina_max **then**
  if effort $<$ effort_max **then**
    effort $\leftarrow$ effort $+$ effort_inc
    effort $\leftarrow$ min(effort, effort_max)
  end if
end if

{ スタミナが小量回復する }
stamina $\leftarrow$ stamina $+$ recovery $\cdot$ stamina_inc_max
stamina $\leftarrow$ min(stamina, stamina_max)

Figure 4.5: スタミナモデルのアルゴリズム

4.5.3 Kick モデル

バージョン 6 から 7 にかけて，キックモデルには基本的な変更は加えられなかった．よって，過去のキックスキルの実装は動作するだろう．しかし，サーバパラメータの変更のために，マルチキックの必要が無くなることがあるだろう．

kick コマンドは 2 つのパラメータ *kick power* と *angle* を取る．*kick power* はボールを蹴る力で，***minpower*** から ***maxpower*** の範囲の値を使える．*angle* はボールを蹴る方向で，値は度数で与える．値の範囲は ***minmoment*** から ***maxmoment*** である．(現在のパラメータ値については，表 4.6 を参照)．

kick コマンドがサーバに受理されると，そのプレイヤがボールをキック可能な状態であれば，キックが実行される．ボールとプレイヤの間の距離が 0 以上 ***kickable_margin*** 以下であれば，プレイヤはボールをキック可能である．ヘテロジニアスプレイヤはことなるキッカブルマージンを持つ．この節における距離の計算では，プレイヤとボールとの距離とは，プレイヤとボールそれぞれの外周円の間の距離を意味する．よって，この節における距離とは，両物体の中心位置の距離からボールの半径とプレイヤの半径を引いた値になる．

キックに関する計算では，まず最初に有効キックパワー ep を計算する:

$$\mathrm{ep} = \mathrm{kick\ power} \cdot \mathrm{kick\_power\_rate} \tag{4.20}$$

有効キックパワーは，ボールがプレイヤの体の正面に密着している場合に最大になり，ボールとプレイヤの位置関係 (角度と距離) に応じて減少する．

プレイヤの体の方向に対するボールの相対角度が $0°$ — 例えば，ボールがプレイヤの正面にある — であれば，有効キックパワーは変化することなくそのまま維持されるだろう．角度が大きくなるほど，有効キックパワーは減少する．最悪のケースは，ボールがプレイヤの真後ろ (角度 $180°$) となる場合で，このとき，有効キックパワーは 25%減少する．

有効キックパワーに関する次に重要な変数は，ボールとプレイヤとの距離である．キックが実行されるとき，ボールとプレイヤとの距離は 0 と ***kickable_margin*** の間であることは明らかである．距離が 0 であれば，やはり有効キックパワーは全く減少しない．ボールがプレイヤから離れるほど，有効キックパワーは減少する．距離が ***kickable_margin*** であれば，有効キックパワーは 25%減少する．

ボールがプレイヤの真後ろで，距離が ***kickable_area*** の時に最悪のケースとなり，*kick power* は 50%しか効果を表さない．有効キックパワーについては，公式 4.21 が得られる．(dir_diff は，ボールの絶対方向とプレイヤの体の絶対方向との角度差の絶対値を意味する．dist_diff は，ボールとプレイヤとの距離の絶対値を意味する．)
$0 \leq \mathrm{dir\_diff} \leq 180° \quad \wedge \quad 0 \leq \mathrm{dist\_diff} \leq \mathrm{kickable\_margin}$:

$$\mathrm{ep} = \mathrm{ep} \cdot \left(1 - 0.25 \cdot \frac{\mathrm{dir\_diff}}{180°} - 0.25 \cdot \frac{\mathrm{dist\_ball}}{\mathrm{kickable\_margin}}\right) \tag{4.21}$$

有効キックパワーは，$\vec{a}_{n_i}$ の計算に用いられる．この加速度ベクトルは，サイクル n におけるボールの絶対加速度 $\vec{a}_n$ に加算される (ボールの近くにいる全てのプレイヤが同時にボールをキック可能であることを覚えておくこと)．

サーバパラメータ ***kick_rand*** が，ボール加速時のノイズ生成に関するパラメータとして用いられる．デフォルトプレイヤに関しては，***kick_rand*** は 0 であり，ノイズは発生しない．ヘテロジニアスプレイヤに関しては，***kick_rand*** の値は `player.conf` 内の

kick_rand_delta_factor パラメータと実際のキッカブルマージンに依存する．RoboCup 2000 においては，通常のプレイヤにも ***kick_rand*** を設定することで，evaluation session を実施した．

シミュレーションステップが n から $n+1$ に移る時，加速度 $\vec{a}_n$ が適用される:

1. $\vec{a}_n$ は，大きさが ***baccel_max*** 以下になるように正規化される．現在 (サーババージョン 9)，最大加速度は有効キックパワーの最大値と等しい．
2. $\vec{a}_n$ が，現在のボール速度 $\vec{v}_n$ に加算される．$\vec{v}_n$ は，大きさが ***ball_speed_max*** 以下になるように正規化される．
3. ノイズ $\vec{n}$ と風の影響 $\vec{w}$ が，$\vec{v}_n$ に加算される．ノイズと風の影響はいずれも，`server.conf` による設定変更が可能である．風に関するパラメータは，***wind_force***，***wind_dir***，***wind_rand***，ノイズに関するパラメータは，***ball_rand*** である．ノイズベクトルの XY 成分の大きさは，いずれも $[-|\vec{v}_n| \cdot \mathrm{ball\_rand} \ldots |\vec{v}_n| \cdot \mathrm{ball\_rand}]$ の間の値である．
4. ボールの新しい位置 $\vec{p}_{n+1}$ は，古い位置 $\vec{p}_n$ に速度ベクトル $\vec{v}_n$ を加算した値である．(すなわち，連続したシミュレーションステップ間でのボールの最大移動距離は，***ball_speed_max*** である).
5. ***ball_decay*** がボールの速度ベクトルに適用される: $\vec{v}_{n+1} = \vec{v}_n \cdot \mathrm{ball\_decay}$. 加速度 $\vec{a}_{n+1}$ の大きさは 0 にセットされる．

現在の設定では，ボールが最大速度で蹴り出された場合，ボールの最大移動距離は 45m である．この場合，キック後 53 サイクル経過した時点で，ボールの移動距離は約 43m となり，ボールのスピードは 0.1 以下になる．キック後 15 サイクル経過した時点では，ボールの移動距離は 27m から 28m となり，ボールのスピードは 1.0 よりわずかに大きい値を保っている．

現在のキックモデルとパラメータ設定では，連続したキックによってボール制御を行なうと効果的である．例えば，ボールを止め，キッカブルエリア内のより有利な位置へボールを運び，望ましい方向へとキックを行うといったことが可能である．連続したキックを行なうことで，相対位置 (0,0°) にボールを置くこと無く，ボールを最大スピードへと加速することも可能になる．

4.5.4 Move モデル

move コマンドは，プレイヤをフィールド上の指定位置へと直接移動させるために用いられる．**move** はチームのセットアップのために存在し，通常の試合中には使用されない．各ハーフの最初 (プレイモードが ‘`before_kick_off`’ の時) と，ゴールが決まった後 (プレイモードが ‘`goal_r_`*n*’ または ‘`goal_l_`*n*’ の時) の場合に利用可能である．これらの状況では，プレイヤは自陣ハーフ内 ($X < 0$) の任意の位置へ移動可能で，プレイモードが変更されない限り，**move** コマンドの実行回数に制限は無い．プレイヤが敵陣ハーフ内へ移動した場合，サーバによって自陣内のランダムな位置へと移動される．

move コマンドの第二の目的は，キーパーがボールをキャッチした後に，ペナルティエリア内でキーパーを移動させることである (4.5.1 節も参照)．キーパーがボールをキャ

基本パラメータ server.conf		ヘテロジニアスプレイヤのパラメータ player.conf		
名前	値	名前	値	範囲
minpower	-100			
maxpower	100			
minmoment	-180			
maxmoment	180			
kickable_margin	0.7	***kickable_margin_delta_min*** ***kickable_margin_delta_max***	0.0 0.2	0.7 — 0.9
kick_power_rate	0.027			
kick_rand	0.0	***kick_rand_delta_factor*** ***kickable_margin_delta_min*** ***kickable_margin_delta_max***	0.5 0.0 0.2	0.0 — 0.1
ball_size	0.085			
ball_decay	0.94			
ball_rand	0.05			
ball_speed_max	2.7			
ball_accel_max	2.7			
wind_force	0.0			
wind_dir	0.0			
wind_rand	0.0			

Table 4.6: ボールとキックモデルに関するパラメータ

ッチすると，キーパーはボールと共にペナルティエリア内を移動できる．キーパーには，ボールをキックするまでに ***goalie_max_moves*** 回までの **move** コマンドの実行が許可されている．それ以上 **move** コマンドを実行しても何の効果も得られず，サーバからは `(error too_many_moves)` という応答が得られる．

`server.conf` 内のパラメータ	値
goalie_max_moves	2

Table 4.7: move コマンドに関するパラメータ

4.5.5 Say モデル

プレイヤは，say コマンドを使用することによって，他のプレイヤへとメッセージを配信することができる．メッセージ長は ***say_msg_size*** 文字まで許されており，say メッセージとして有効な文字セットは [`-0-9a-zA-Z ().+*/?<>_`] である (カギ括弧は除く)．メッセージを発したプレイヤからの距離が ***audio_cut_dist*** 以内の位置にいれば，その他のプレイヤは (どちらのチームの所属であっても) メッセージを聞くことができる (4.3.1 節も参照)．サーババージョン 7 までは，say コマンドがサーバへ送られると，メッセージはすぐに配信されていた．バージョン 8 以降，メッセージの配信は，say コマンドが受理された次のサイクルの最初に行なわれように変更された．say コマンドに関する制限は，メッセージを聞くプレイヤの聴覚能力の制限によってのみ行なわれる．

`server.conf` 内のパラメータ	値
say_msg_size	512
audio_cut_dist	50
hear_max	2
hear_inc	1
hear_decay	2

Table 4.8: say コマンドに関するパラメータ

4.5.6 Turn モデル

dash コマンドは，プレイヤをその体の方向へと加速させるために用いられる．一方，**turn** コマンドは，プレイヤの体の方向を変化させるために用いられる．**turn** コマンドの引数はモーメントである．モーメントとして有効になるのは，***minmoment*** と ***maxmoment*** の間の値である．プレイヤが移動していない場合 (速度が 0)，モーメントはプレイヤがターンする角度に等しい．しかし，ターンには慣性の概念が導入されており，プレイヤが移動しているときには，ターンが行い難くなっている．プレイヤの実際のターン角度は次のようになる:

$$\mathrm{actual\_angle} = \mathrm{moment}/(1.0 + \mathrm{inertia\_moment} \cdot \mathrm{player\_speed}) \tag{4.22}$$

inertia_moment は server.conf 内のパラメータで，デフォルト値は 5.0 である．***inertia_moment*** が 5.0 で，プレイヤのスピードが 1.0 のとき，可能となる最大のターン角度は，±30 となる．しかしながら，同じサイクルでダッシュとターンを同時に実行することはできないため，プレイヤが **turn** を実行するときの最大スピードは，***player_speed_max*** · ***player_decay*** となる．よって，デフォルトタイプのプレイヤが最大スピードでターンを実行する場合，その最大角度は ±60 となる．

ヘテロジニアスプレイヤに設定される慣性モーメントは，デフォルトの ***inertia_value*** に，***player_decay_delta_min*** · ***inertia_moment_delta_factor*** と ***player_decay_delta_max*** · ***inertia_moment_delta_factor*** の間のランダムな値が加えられた値になる．

基本パラメータ server.conf		ヘテロジニアスプレイヤのパラメータ player.conf		
名前	値	名前	値	範囲
minmoment	-180			
maxmoment	180			
inertia_moment	5.0	***player_decay_delta_min*** ***player_decay_delta_max*** ***inertia_moment_delta_factor***	0.0 0.2 25.0	5.0 10.0

Table 4.9: ターンモデルパラメータ

4.5.7 TurnNeck モデル

プレイヤは，**turn_neck** コマンドを用いることによって，体の角度とは独立に，自身の首の角度を変えることができる．プレイヤの首の方向は，プレイヤの視界の方向である．**turn** コマンドがプレイヤの体の方向を変更するのに対し，**turn_neck** はプレイヤの体の方向と相対的な首の角度を変更する．相対方向の最小値と最大値は，server.conf 内のパラメータ，***minmoment*** と ***maxmoment*** によって設定される．首の角度はプレイヤの体の方向に対して相対的であるため，クライアントが **turn** コマンドを実行すると，**turn_neck** コマンドが実行されていなかったとしても，視界の方向が変更されることになる．

更に，**turn_neck** コマンドは．**turn**, **dash**, **kick** などのコマンドと同じサイクル内で実行可能である．**turn_neck** は，**turn** のような慣性による影響を受けない．**turn_neck** の引数の値は，***minneckmoment*** と ***maxneckmoment*** の範囲内でなければならない．

4.6 ヘテロジニアスプレイヤ

バージョン 7 のサーバから，ヘテロジニアスプレイヤが導入された．サーバは，起動時に ***player_types*** 個のプレイヤタイプを生成する．ヘテロジニアスプレイヤは，player.conf で定義されているトレードオフに基づいた異なる能力を持つ．試合を行なう両チームは同じプレイヤタイプ集合を用いる．タイプ 0 はデフォルトタイプで，このタイプに限りパラメータは常に固定である．

server.conf 内のパラメータ	値
minneckang	-90
maxneckang	90
minneckmoment	-180
maxneckmoment	180

Table 4.10: turn_neck コマンドに関するパラメータ

プレイヤがサーバに接続した時，利用可能なプレイヤタイプの情報を受信する (??節を参照)．オンラインコーチは，'before_kick_off' 時であれば無制限にプレイヤタイプを変更できる．それ以外の場合，非 'play_on' の時に限り，***subs_max*** 回までプレイヤタイプの変更が行なえる．プレイヤタイプの変更は，**change_player_type ...** コマンドによって行なう (7.4 節を参照)．

プレイヤが他のプレイヤタイプに交替されるたびに，プレイヤの stamina, recovery, effort はそのプレイヤタイプの初期値 (最大値) にリセットされる．

player.conf 内のパラメータ	値
player_types	7
subs_max	3

Table 4.11: 交替とヘテロジニアスプレイヤに関するパラメータ

4.7 審判モデル

プレイヤが試合のプレイモードを知るために，自動化された審判がプレイヤへとメッセージを送信する．審判のルールと振舞いについては， 2.2.1 節で述べられている．プレイヤは，審判のメッセージを *hear* メッセージとして受け取る．審判からのメッセージは，他のプレイヤとのコミュニケーションメッセージの数に関係なく，常に聞くことができる．

4.7.1 プレイモードと審判メッセージ

プレイモードの変更は，審判によってアナウンスされる．更に，ゴールやファウルのようなイベントのアナウンスも審判のメッセージに含まれる．サーバのソースには，現在使われていない追加のプレイモードも存在する．プレイモードと審判メッセージのいずれも，(referee *String*) という形式でアナウンスされる．*String* が，プレイモードまたは審判メッセージの文字列である．プレイモードは，表 4.12 にリストされている．審判メッセージに関しては，表 4.13 を参照．

プレイモード	t_c	次のプレイモード	コメント
'before_kick_off'	0	'kick_off_*Side*'	ハーフの最初
'play_on'			通常のプレイ中
'time_over'			
'kick_off_*Side*'			試合開始アナウンス (Kick Off ボタンを押した後)
'kick_in_*Side*'			
'free_kick_*Side*'			
'corner_kick_*Side*'			
'goal_kick_*Side*'		'play_on'	ボールがペナルティエリア外へ出ると変更される
'goal_*Side*'			現在未使用 (表 4.13 を参照).
'drop_ball'	0	'play_on'	
'offside_*Side*'	30	'free_kick_*Side*'	反対のサイドへ与えられる

Side は，文字 'l' か 'r' のいずれか, *OSide* はその反対のサイドを意味する
t_c は，次のプレイモードがアナウンスされるまでの時間 (サイクル数)

Table 4.12: プレイモード

メッセージ	t_c	次のプレイモード	コメント
goal_*Side*_*n*	50	'kick_off_*OSide*'	ゴール数のアナウンス (n 点目)
foul_*Side*	0	'free_kick_*OSide*'	ファウルのアナウンス
goalie_catch_ball_*Side*	0	'free_kick_*OSide*'	
time_up_without_a_team	0	'time_over'	タイムアップ (第 2 ハーフの終了時) まで敵チーム存在しなかった場合に送られる
time_up	0	'time_over'	試合終了時に送られる (サイクルが $\geq$ second half かつ両チームのスコアが異なる)
half_time	0	'before_kick_off'	
time_extended	0	'before_kick_off'	

Side は，文字 'l' か 'r' のいずれか, *OSide* はその反対のサイドを意味する
t_c は，次のプレイモードがアナウンスされるまでの時間 (サイクル数)

Table 4.13: 審判メッセージ

4.8 サッカーシミュレーション

4.4 節では，物体が，その加速度と速度に従ってどのように移動させられるかについての説明を行なった．この節では，シミュレーションにおいて，仮想度と速度が物体へいつ適用されるのかについてを説明する．

4.8.1 シミュレーションアルゴリズムの説明

サッカーサーバでは，時間は離散的なステップで更新される．シミュレーションステップは 100ms である．各シミュレーションステップの間，物体 (プレイヤやボール) はその位置に留まっている．プレイヤがステップ内で行動を決定した場合，シミュレーションサイクルが次のステップへと移る時に，プレイヤとボールへと行動の効果が適用される．プレイモードによっては，プレイヤの行動に許可されないものがある (例えば，'before_kick_off' モードの時，プレイヤは **turn** と **move** は実行可能だが，**dash** は実行できない)．許可された行動のみが適用され，効果を得られる．

ステップ中において，複数のプレイヤがボールをキックした場合，全てのキックがボールへと適用され，最終的な加速度が計算される．最終的に得られた加速度がボールの最大加速度を越えていれば，加速度はその最大値へと正規化される．物体の移動後，サーバは衝突のチェックを行ない，もし衝突が起こっていれば速度の更新を行なう (4.4.2 節も参照)．

加速度と速度が物体へ適用される時，その適用順序はランダムである．物体の位置が変化し，速度と加速度が更新された後，審判は状況をチェックし，必要であればプレイモードや物体の位置を変更する．プレイモードの変更はすぐにアナウンスされる．最後に，各プレイヤのスタミナが更新される．

4.9 サッカーサーバの使用

4.9.1 サッカーサーバパラメータ

Table 4.14: server.conf 内の調整可能なパラメータ

名前	Default 値	使用される値 in server.conf	説明
goal_width	7.32	14.02	ゴールの幅
player_size		0.3	プレイヤの半径
player_decay		0.4	プレイヤの速度減衰率
player_rand		0.1	プレイヤのノイズファクタ
player_weight		60.0	プレイヤの重さ
player_speed_max		1.0	プレイヤの最大スピード
player_accel_max		1.0	プレイヤの最大加速
stamina_max		4000.0	最大スタミナ値
stamina_inc_max		45.0	最大スタミナ回復値
recover_dec_thr		0.3	recovery 減少閾値率
recover_min		0.5	最小 recovery 値
recover_dec		0.002	recovery 減少幅
effort_dec_thr		0.3	effort 減少閾値率
effort_min		0.6	最小 effort 値
effort_dec		0.005	effort 減少幅

Table 4.14: (continued)

名前	Default 値	使用される値 in `server.conf`	説明
effort_inc_thr		0.6	effort 増加閾値率
effort_inc		0.01	effort 増加幅
kick_rand		0.0	キックノイズ率
team_actuator_noise			チームごとに特定のアクチュエータノイズを加えるかどうかのフラグ
prand_factor_l			左チームの prand にかける追加ファクタ
prand_factor_r			右チームの prand にかける追加ファクタ
kick_rand_factor_l			左チームの kick_rand にかける追加ファクタ
kick_rand_factor_r			左チームの kick_rand にかける追加ファクタ
ball_size		0.085	ボールの半径
ball_decay		0.94	ボールの速度減衰率
ball_rand		0.05	ボールのノイズファクタ
ball_weight		0.2	ボールの重さ
ball_speed_max		2.7	ボールの最大スピード
ball_accel_max		2.7	ボールの最大加速
dash_power_rate		0.006	ダッシュパワー効果率
kick_power_rate		0.027	キックパワー効果率
kickable_margin		0.7	キッカブルマージン
control_radius			control radius
catch_probability		1.0	キーパのキャッチ成功確率
catchable_area_l		2.0	キーパのキャッチエリアの長さ
catchable_area_w		1.0	キーパのキャッチエリア幅
goalie_max_moves		2	キーパのキャッチ後の最大 move 回数
maxpower		100	最大パワー
minpower		-100	最小パワー
maxmoment		180	最大モーメント
minmoment		-180	最小モーメント
maxneckmoment		180	最大首モーメント
minneckmoment		-180	最小首モーメント
maxneckang		90	最大首角度
minneckang		-90	最小首角度
visible_angle		90.0	標準視野角
visible_distance			触覚的視覚可能距離
audio_cut_dist		50.0	音声の到達距離
quantize_step		0.1	移動物体のための距離の離散化ステップ
quantize_step_l		0.01	静止物体のための距離の離散化ステップ
quantize_step_dir			
quantize_step_dist_team_l			
quantize_step_dist_team_r			
quantize_step_dist_l_team_l			
quantize_step_dist_l_team_r			
quantize_step_dir_team_l			
quantize_step_dir_team_r			

Table 4.14: (continued)

名前	Default 値	使用される値 in `server.conf`	説明
ckick_margin		1.0	コーナーキックマージン
wind_dir	0.0	0.0	風の方向
wind_force	10.0	0.0	風の強さ
wind_rand	0.3	0.0	風のノイズファクター
wind_none	false		風の使用フラグ
wind_random	false		風のランダム発生
inertia_moment		5.0	turn の慣性モーメント
half_time		300	ハーフタイムの秒数
drop_ball_time		200	自動ドロップボールまでの待機サイクル数
port		6000	プレイヤのポート番号
coach_port		6001	(offline) コーチのポート番号
olcoach_port		6002	オンラインコーチのポート番号
say_coach_cnt_max		128	オンラインコーチによる freeform メッセージ使用回数の上限
say_coach_msg_size		128	オンラインコーチによる freeform メッセージ長の上限
simulator_step		100	シミュレーションステップ [単位:ミリ秒]
send_step		150	標準視覚情報ステップ [単位:ミリ秒]
recv_step		10	サーバによるコマンド受信間隔 [単位:ミリ秒]
sense_body_step		100	sense_body ステップ [単位:ミリ秒]
say_msg_size		10	say メッセージ長の上限 [単位:byte]
clang_win_size		300	CLang におけるメッセージ送信時間帯の刻み幅
clang_define_win		1	時間帯あたりの最大メッセージ数
clang_meta_win		1	時間帯あたりの最大メッセージ数
clang_advice_win		1	時間帯あたりの最大メッセージ数
clang_info_win		1	時間帯あたりの最大メッセージ数
clang_mess_delay		50	CLang メッセージのプレイヤへの配送遅延
clang_mess_per_cycle		1	コーチによるメッセージの 1 サイクルあたりの最大数
hear_max		1	hear キャパシティの最大値
hear_inc		1	hear キャパシティの回復幅
hear_decay		1	hear キャパシティの減少幅
catch_ban_cycle		5	キャッチ実行の最小間隔
coach			オフラインコーチの使用フラグ
coach_w_referee			オフラインコーチとトレーナの併用フラグ
old_coach_hear			オフラインコーチの聴覚情報に古いフォーマットを用いるフラグ
send_vi_step		100	オンラインコーチの視覚情報ステップ
use_offside		on	オフサイドルール適用フラグ
offside_active_area_size		2.5	オフサイド検出距離
forbid_kick_off_offside		on	キックオフ前の敵陣侵入許可フラグ
log_file			
record			
record_version		3	RCG ファイルのバージョン指定
record_log		on	クライアントのコマンドログフラグ
record_messages			
send_log		on	
log_times		off	サイクルに消費した時間の記録フラグ

Table 4.14: (continued)

名前	Default 値	使用される値 in `server.conf`	説明
verbose		off	verbose モードフラグ
replay			
offside_kick_margin		9.15	offside kick margin
slow_down_factor			
start_goal_l			
start_goal_r			
fullstate_l			
fullstate_r			

5 サッカーモニタ

5.1 はじめに

サッカーモニタは視覚的なインターフェイスを提供する．モニタを用いることによって，試合の様子をはっきりと見ることができ，試合の制御も行える．サッカーモニタとログプレイヤと共に用いることによって，試合の再生表示を行うことも可能になる．この機能は，クライアントの解析やデバッグのために非常に便利なものである．

5.2 Getting started

クラシックモニタをサーバへ接続するには，以下のコマンドを実行するだけで良い:

```
-> rcssmonitor_classic %soccermonitor -f ConfFileName [-ParameterName Value]*
```

設定ファイルの代わりにプログラム引数を指定することで，クラシックモニタのパラメータを変更できる．(設定とパラメータの詳細は 5.6 節を参照)

"rcsoccersim" スクリプトを用いれば，モニタは自動的に起動し，サーバへと接続される．

```
-> rcsoccersim
```

デフォルトでは，rcssmonitor が起動される．rcsoccersim スクリプトに引数を指定することで，モニタの使い分けが可能である．

```
-> rcsoccersim -frameview // rcssmonitor を起動
-> rcsoccersim -classic // rcssmonitor_classic を起動
```

5.3 サーバからモニタへの通信

サッカーモニタとサッカーサーバは，UDP/IP によってポート番号 6000(default) で接続される．サーバがモニタと接続すると，モニタへの情報送信は毎サイクル行われる．サッカーサーバ バージョン 7 以降は，二つの異なるフォーマット (バージョン 1 とバージョン 2) を提供する．サーバがどちらのフォーマットを送信するかは，モニタの初期化コマンドによって決定される (5.4 を参照)．詳細なデータ構造は，付録 C に記述されている．

5.3.1 Version 1

サッカーサーバとログプレイヤは，dispinfo_t 構造体をモニタへ送信する．dispinfo_t は，以下の三つの型の共用体を含む:

- showinfo_t: シーンの描画に必要な情報

- msginfo_t : プレイヤと審判からのメッセージ (クラシックモニタのウインドウ下部に表示される情報)
- drawinfo_t: 円，直線，点をモニタに描画するための情報 (サーバでは使われない)

dispinfo_t は共用体を含むため，そのサイズは共用体の最も大きいサブパートによって決定される．最も大きいのは msginfo_t であり，最終的な dispinfo_t のサイズは 2052 バイトとなる．共用体は余分なネットワーク負荷を起こす可能性があるため，将来のバージョンでは変更されるかもしれない．異なるプラットフォーム間での互換性を保つために，dispinfo_t 内の値はネットワークバイトオーダで表現される．共用体に含まれる情報がどのタイプかを決定するために，mode 情報を用いる．NO_INFO は有効な情報が含まれないことを示す (サーバから送信される事は無い)．BLANK_MODE は，空白スクリーンの表示をモニタへ伝える (ログプレイヤによって使われる)．これらは，param.h 内で定義されている.

```
NO_INFO    0
SHOW_MODE  1
MSG_MODE   2
DRAW_MODE  3
BLANK_MODE 4
```

以下は，各構造体とそれに含まれるデータの解説である:

Showinfo

showinfo_t 構造体は，サイクル (100ms) 毎にモニタへと送信される．含まれる情報は，プレイヤとボールの状態と位置である．

```
typedef struct {
   char   pmode ;
   team_t team[2] ;
   pos_t  pos[MAX_PLAYER * 2 + 1] ;
   short  time ;
} showinfo_t ;
```

- pmode: 現在のプレイモード (types.h で定義)

```
PM_Null,
PM_BeforeKickOff,
PM_TimeOver,
PM_PlayOn,
PM_KickOff_Left,
PM_KickOff_Right,
PM_KickIn_Left,
PM_KickIn_Right,
PM_FreeKick_Left,
```

```
        PM_FreeKick_Right,
        PM_CornerKick_Left,
        PM_CornerKick_Right,
        PM_GoalKick_Left,
        PM_GoalKick_Right,
        PM_AfterGoal_Left,
        PM_AfterGoal_Right,
        PM_Drop_Ball,
        PM_OffSide_Left,
        PM_OffSide_Right,
PM_PK_Left,
PM_PK_Right,
PM_FirstHalfOver,
PM_Pause,
PM_Human,
PM_Foul_Charge_Left,
PM_Foul_Charge_Right,
PM_Foul_Push_Left,
PM_Foul_Push_Right,
PM_Foul_MultipleAttacker_Left,
PM_Foul_MultipleAttacker_Right,
PM_Foul_BallOut_Left,
PM_Foul_BallOut_Right,
PM_Back_Pass_Left,
PM_Back_Pass_Right,
PM_Free_Kick_Fault_Left,
PM_Free_Kick_Fault_Right,
PM_CatchFault_Left,
PM_CatchFault_Right,
PM_IndFreeKick_Left,
PM_IndFreeKick_Right,
PM_PenaltySetup_Left,
PM_PenaltySetup_Right,
PM_PenaltyReady_Left,
PM_PenaltyReady_Right,
PM_PenaltyTaken_Left,
PM_PenaltyTaken_Right,
PM_PenaltyMiss_Left,
PM_PenaltyMiss_Right,
PM_PenaltyScore_Left,
PM_PenaltyScore_Right,
        PM_MAX
```

- team: チーム情報．インデックス 0 が左チームとなる．

- pos: ボールとプレイヤの位置情報．インデックス 0 はボール，1 から 11 は team[0]，12 から 22 は team[1] である．
- time: 現在のサイクル．short 型のため，0 から 65535 までしか表現できない．

```
typedef struct {
   char  name[16];  /* name of the team */
   short score;     /* current score of the team */
} team_t ;

typedef struct {
   short enable ;
   short side ;
   short unum ;
   short angle ;
   short x ;
   short y ;
} pos_t ;
```

各要素は以下の値を取り得る:

- enable: 物体の状態．フィールドに存在しないプレイヤ(ボール)の状態は，DISABLE となる．他の有効ビットは，モニタがプレイヤの状態や行動をより詳細に描画することを可能にする．types.h で定義．

```
        DISABLE         (0x0000)
        STAND           (0x0001)
        KICK            (0x0002)
        KICK_FAULT      (0x0004)
        GOALIE          (0x0008)
        CATCH           (0x0010)
        CATCH_FAULT     (0x0020)
BALL_TO_PLAYER  (0x0040)
PLAYER_TO_BALL  (0x0080)
DISCARD         (0x0100)
LOST            (0x0200)
BALL_COLLIDE    (0x0400) // player collided with the ball
PLAYER_COLLIDE  (0x0800) // player collided with another player
TACKLE          (0x1000)
TACKLE_FAULT    (0x2000)
BACK_PASS       (0x4000)
FREE_KICK_FAULT (0x8000)
```

- side: プレイヤのチームサイド．LEFT は左から右へ攻めるチーム．NEUTRAL はボール．types.h で定義．

```
LEFT      1
NEUTRAL   0
RIGHT    -1
```

- unum: プレイヤの背番号．範囲は 1 から 11．
- angle: プレイヤの体の方向．範囲は-180 度から 180 度．-180 度のときスクリーンの左を，-90 度のとき上を，0 度のとき右を，90 度のとき下を向いている．
- x, y: プレイヤの位置座標．(0, 0) はフィールドの中央．x はスクリーン右方向へ増加，y はスクリーン下方向へ増加する．エイリアシングを減少させるために，SHOWINFO_SCALE (16) が値に乗ぜられる．よって，フィールドサイズは，x 方向が PITCH_LENGTH * SHOWINFO_SCALE，y 方向が PITCH_WIDTH * SHOWINFO_SCALE となる．

Messageinfo

プレイヤと審判のメッセージを含む情報．

```
typedef struct {
   short board ;
   char  message[2048] ;
} msginfo_t ;
```

- board: メッセージのタイプを示す．MSG_BOARD タイプのメッセージは，左のテキストウインドウのための審判メッセージである．LOG_BOARD タイプのメッセージは，プレイヤとサーバ間のメッセージである．param.h で定義．

```
MSG_BOARD 1
LOG_BOARD 2
```

- message: ヌル文字終端のメッセージ文字列．

Drawinfo

モニタが円，直線，点を描画するための情報．

5.3.2 Version 2

サッカーサーバとログプレイヤは，バージョン 1 で用いられた dispinfo_t の代わりに dispinfo_t2 構造体をモニタへと送信する．dispinfo_t2 五つの異なる型による共用体を含む．データ構造は，付録 C に記述されている．

- showinfo_t2: シーンの描画に必要な情報．プレイヤとボールの位置と速度，チーム名，スコアなどの全ての情報を含む．

- msginfo_t : プレイヤと審判からのメッセージ (クラシックモニタのウインドウ下部に表示される情報)．チームロゴイメージ情報とプレイヤ交替情報も含む．
  - team graphic: チームグラフィクフォーマットは，8x8 のタイルに分割された 256x64 イメージである．メッセージには各タイルの情報が含まれる．

    ```
    (team_graphic_{l|r} (<X> <Y> "<XPM line>" ... "<XPM line>"))
    ```

    X と Y は，完全な 256x64 イメージにおける 8x8 タイルの位置を示す．それぞれ 0 から始まり，X は 31 まで，Y は 7 までである．各 XPM ラインは，8x8 の XPM タイルにおける 1 ラインである．
  - substitutions: 交替は，次の形式によってメッセージボードへ明確に記録される．

    ```
    (change_player_type {l|r} <unum> <player_type>)
    ```

- player_type_t: プレイヤタイプ一人分の能力パラメータ
- server_params_t: サッカーサーバの設定パラメータ
- player_params_t: プレイヤタイプのトレードオフパラメータ

どの情報が共用体に含まれるかは，mode フィールドによって決定される．NO_INFO は有効な情報が含まれないことを示す (サーバから送信される事は無い)．BLANK_MODE は，空白スクリーンの表示をモニタへ伝える (ログプレイヤによって使われる)．これらは，param.h 内で定義されている.

```
NO_INFO      0
SHOW_MODE    1
MSG_MODE     2
BLANK_MODE   4
PT_MODE      7
PARAM_MODE   8
PPARAM_MODE  9
```

5.4 モニタからサーバへの通信

モニタは，以下のコマンドをサーバへと送ることができる (全てのコマンドにおいて，`<variable>`は適切な値で置き換えられる変数である)．

```
(dispinit) | (dispinit version <version>)
```

最初に，モニタとして登録するためのメッセージをサーバへ送信する (プレイヤとの区別のため．プレイヤとモニタは同じポート番号 6000 でサーバと接続する)．”(dispinit)” のみの場合は，バージョン 1 の情報が送られてくる．”(dispinit version 2)” とすれば，バージョン 2 の情報が送られてくる．モニタの設定パラメータを変更することで，バージョンを変更することができる．

```
(dispstart)
```

サーバに試合開始，後半の開始，延長戦の開始のための (kick off) の合図を送信する．試合が既に実行中の場合は無視される．

```
(dispfoul <x> <y> <side>)
```

ファウルの状況を指定して送信する．x と y はファウルの発生した位置座標である．side が LEFT (1) の場合，左チームへフリーキックが与えられる．side が NEUTRAL (0) の場合，ドロップボールが行われる．side が RIGHT (-1) の場合，右チームへフリーキックが与えられる．

```
(dispdiscard <side> <unum>)
```

プレイヤのレッドカードを与える (退場させる) 指示を送信する．side は LEFT または RIGHT で，unum はプレイヤの背番号 (1 - 11) である．

```
(dispplayer <side> <unum> <posx> <posy> <ang>)
```

プレイヤを指定の位置へ移動させる指示を送信する．体の向きも同時に指定可能である．side は LEFT (1) または RIGHT (-1) で，unum はプレイヤの背番号 (1 - 11) である．posx と posy によってプレイヤの新しい位置を指定するが，サーバ内部で実際に使用される値は，これらを SHOWINFO_SCALE で割った値である．ang は，プレイヤの新しい体の方向を度数で示す．このコマンドは，サーババージョン 7.02 で追加された．

```
(compression <level>)
```

バージョン 8.03 以降，サーバはクライアントとの圧縮通信をサポートする．モニタは，上記の圧縮要求をサーバへ送信することで、圧縮通信を開始できる．サーバが ZLib 無しでコンパイルされていれば，上記コマンドに対して，サーバは `(warning compression_unsupported)` という応答を返す．また，`<level>` が 0 から 9 の間でなければ，サーバは `(error illegal_command_form)` という応答を返す．それ以外の場合は，`(ok compression <level>)` という応答を返し，それ以降のクライアントへのメッセージは全て，そのレベルで圧縮される．ただし，新しい compression コマンドが送信されれば，その時点で再びレベルが変更される．圧縮レベルとして 0 以上が選択されると，モニタからサーバへのコマンド自体も圧縮されているとみなされる．レベルを 0 に設定することは，圧縮を完全に解除すること (デフォルト設定) になる．

5.5 試合の記録方法と再生方法

試合を記録するには，以下のオプションを設定してサーバを起動すれば良い．

```
-record LOGFILE
```

(LOGFILE はログファイル名) または，server.conf 内のパラメータを設定する:

```
record.log : on.
```

以下のオプションによって，ログファイルのバージョンを指定できる:

```
-record_version [1/2/3]
```

または，server.conf 内のパラメータを設定する:

```
record_version : 2
```

ログプレイヤによって，記録した試合の再生が可能になる．ログファイルはログプレイヤによって読まれ，ログプレイヤに接続したサッカーモニタへと情報が送信される．ログファイルを再生するには，ログファイル名を引数としてログプレイヤを起動し，サッカーモニタをログプレイヤへと接続し，ログプレイヤウインドウのボタン (スタート，ストップ，逆再生，ステップ再生) を操作すれば良い．

5.5.1 バージョン 1 プロトコル

バージョン 1 のログファイル (サーババージョン 4.06 まで使用) は，連続的な dispinfo_t チャンクのストリームである．dispinfo_t の構造が共用体であるために，多くのバイトを無駄に浪費し，非実用的なサイズのログファイルが生成されてしまう．この問題を解決するために，バージョン 2 として新しいログファイルフォーマットが導入された．

5.5.2 バージョン 2 プロトコル

バージョン 2 のログファイルプロトコルでは，冗長で不用なデータの記録を回避しようとしている．フォーマットは次のようになる:

- ファイルのヘッダ:
  ファイルヘッダはログファイルのバージョンを自動検出するために使用される．ヘッダが無ければ，Unix バージョン 1 と想定される．'ULG' という 3 文字があれば，これは Unix ログファイルであることを示している (Windows フォーマットとの区別のため)．(訳注:公式には，Windows フォーマットのログファイルと言うものは存在しないので，考慮する必要は無い)

- char 型 バージョン :
  ログファイルフォーマットのバージョン．バージョン 2 では，'2' が記録される．

- body:
  ファイルの残りの部分は，以下のフォーマットのチャンクによってデータを格納している:

- short 型 mode :
  dispinfo_t 構造体 (5.5.1 バージョン 1 を参照) の mode に相当．SHOW_MODE は showinfo_t 情報を，MSG_MODE は msginfo_t を意味する．
  - モードが SHOW_MODE であれば，続くデータは showinfo_t 構造体である．
  - モードが MSG_MODE であれば，次のバイトは:
    * short 型 board: ボード情報 (メッセージタイプ) を示す

* short 型 length: メッセージの長さを示す (ヌルターミネータを含む)
* 文字列 (char 配列) msg: length 個の文字によるメッセージ

DRAW_MODE や BLANK_MODE といった他の情報は，ログファイルには記録されない．空間の最適化に関しては，まだ工夫の余地が残っている．チーム名をファイルヘッダの一部に含めてしてしまうことで，一度きりの記録にすることができる．プレイヤの背番号は，配列のインデックスと利用することで暗黙的に指定することができる．

このように，バージョン 2 の情報チャンクは同じサイズにはならない．よって，ログファイルを逆再生するような場合に，固定サイズバイトのシークだけで済ませることはできない．一度にファイル全体を読んでしまうか，少なくとも，showinfo_t チャンクのストリーム位置を記録しなければならない．

異なるプラットフォーム間での互換性を維持するために，値はネットワークバイトオーダで表現される．

5.5.3 バージョン 3 プロトコル

バージョン 3 のプロトコルでは，ヘテロジニアスプレイヤのためのパラメータ情報が追加され，空間の最適化が行われた．フォーマットは次のようになる:

- ファイルのヘッダ:
  バージョン 2 と同じ．ファイルは定数文字列 'ULG' で始まる．
- char 型 バージョン :
  ログファイルフォーマットのバージョン．バージョン 2 では，'3' が記録される．
- body:
  short 型の値によって，次に続くデータ構造が特定される．
  - short 型の値が PM_MODE の場合，
    * プレイモードを特定する char 型の値が続く

    この情報は，プレイモードが変更されたときにのみ記録される．
  - short 型の値が TEAM_MODE の場合，
    * 左サイドのチーム情報を表す team_t 構造体 と
    * 右サイドのチーム情報を表す team_t 構造体 が続く

    チームデータは，新しいチームが接続するか，得点に変化があったときにのみ記録される．
  - short 型の値が SHOW_MODE の場合，
    * ボールとプレイヤの位置情報や状態を表す short_showinfo_t2 構造体が続く
  - short 型の値が MSG_MODE の場合，
    * short 型 board: ボード情報 (メッセージタイプ) を示す
    * short 型 length: メッセージの長さを示す
    * 文字列 (char 配列) msg: length 個の文字によるメッセージ

  - short 型の値が PARAM_MODE の場合，
    * サーバの設定パラメータを表す server_params_t 構造体が続く

    この情報は，ログファイルの最初に一度だけ記録される．
  - short 型の値が PPARAM_MODE の場合，
    * ヘテロジニアスプレイヤのトレードオフパラメータを表す player_params_t 構造体が続く

    この情報は，ログファイルの最初に一度だけ記録される．
  - If the short is PT_MODE,
    * 特定のプレイヤタイプのパラメータを表す player_type_t 構造体が続く

    この情報は，プレイヤタイプごとに一回ずつログファイルの最初に記録される．

データ変換:

- x， y のような位置座標値は，SHOWINFO_SCALE2 が掛けられたメートルである．
- deltax， deltay のような速度ベクトル成分値は，SHOWINFO_SCALE2 が掛けられた meters/cycle である．
- body_angle， head_angle，view_width などの角度情報は，SHOWINFO_SCALE2 が掛けられたラジアンである．
- stamina， effort， recovery といった他の値に関しても，SHOWINFO_SCALE2 が掛けられた値が記録される．

5.6 設定パラメータ

サッカーモニタ (クラシックモニタ) は次のような変更可能なパラメータを持つ:
"使用される値" は，monitor.conf に記述されている現在のパラメータ値である．"デフォルト値" はソースファイルに記述されているパラメータ値である．

上の表に書かれているパラメータは，以下のように，コマンドラインオプションによっても設定可能である．%soccermonitor [-ParameterName Value]* パラメータは，設定ファイル ~/.rcssmonitor_classic を編集することでも変更可能である．設定ファイルの各行は，以下のような，パラメータの名前と値のペアから成る: ParameterName : Value '#' で始まる行はコメント行である．

5.7 What's New

8.03 :

  - 5.4 節で述べられているように，サーバはモニタとの圧縮通信をサポートするようになった．
  - プレイヤ交替の情報がメッセージログに追加された．

パラメータ名	使用される値	デフォルト	説明
host	localhost	localhost	サーバが稼働するホスト名
port	6000	6000	モニタの初期接続ポート番号
version	2	1	モニタプロトコルバージョン
length_magnify	6.0	6.0	フィールドサイズの倍率
goal_width	14.02	7.32	ゴール幅
print_log	off	on	コミュニケーションログの描画フラグ [on/off]
log_line	6	6	ログウインドウのサイズ w
print_mark	on	on	フィールド上のマーク (ロゴ) 描画フラグ [on/off]
mark_file_name	mark.RoboCup.grey.xbm	mark.xbm	マーク画像ファイル名
ball_file_name	ball-s.xbm	ball.xbm	ボール画像ファイル名
player_widget_size	9.0	1.0	プレイヤウィジェットサイズ
player_widget_font	5x8	fixed	プレイヤウィジェットの背番号用フォント
uniform_num_pos_x	2	2	プレイヤの背番号の描画位置 (X)
uniform_num_pos_y	8	8	プレイヤの背番号の描画位置 (Y)
team_l_color	gold	gold	左チームの色
team_r_color	red	red	右チームの色
goalie_l_color	green	green	左チームのキーパの色
goalie_r_color	purple	purple	右チームのキーパの色
neck_l_color	black	black	左チームの首の色
neck_r_color	black	black	右チームの首の色
goalie_neck_l_color	black	black	左チームのキーパの首の色
goalie_neck_r_color	black	black	右チームのキーパの首の色
status_font	7x14bold	fixed	ステータスラインフォント [チーム名，スコア，時間，プレイモード]
popup_msg	off	off	"GOAL!!" と "Offside!" のポップアップ使用フラグ [on/off]
goal_label_width	120	120	"GOAL!!" ポップアップラベルの幅
goal_label_font	-adobe-times-bold-r-*-*-34-*-*-*-*-*-*-*	fixed	"GOAL!!" ポップアップラベルのフォント
goal_score_width	40	40	"GOAL!!" ポップアップのスコアの幅
goal_score_font	-adobe-times-bold-r-*-*-25-*-*-*-*-*-*-*	fixed	"GOAL!!" ポップアップのスコアのフォント
offside_label_width	120	120	"Offside!" ポップアップラベルの幅
offside_label_font	-adobe-times-bold-r-*-*-34-*-*-*-*-*-*-*	fixed	"Offside!" ポップアップラベルのフォント
eval	off	off	エバリュエーションモード用のフラグ
redraw_player	on	off	プレイヤを常に再描画 (RH 5.2 で必要)

– チームグラフィック情報がメッセージログに追加された

7.07 :

– ログプレイヤが server_param，player_param，player_type メッセージを送信していなかった問題を修正した．
– stamina_max が 0 にセットされていたログファイルの再生時にモニタがクラッシュしていた．この原因によってモニタがクラッシュする事は無くなった．

7.05 :

– 特定のサイクルがログプレイヤによって描画されず，“スキップ” してしまう現象が時折見られていた．これは，ログプレイヤがあまりに多くの UDP パケットをモニタへ送信していたことが原因である．この問題の修正のために，新しいパラメータ 'message_delay_interval' をログプレイヤに追加した．ログプレイヤは，メッセージの送信後，1 マイクロ秒間待機し，モニタがパケットを捉える機会を伺う．個の動作は保証されていない．しかし，かなりの効果があるようである．それでも，ログプレイヤとモニタに関する “スキップ” 問題が解決されない場合は，message_delay_interval の値を，デフォルトの 10 から減少させてみて欲しい．message_delay_interval が負の値であれば，遅延は発生しない．
– サーバは，ログファイルに記録する前に，プレイヤとコーチからのメッセージを 128 文字に切り詰めていた．個の問題が修正された．

7.04 :

– クライアントがバージョン 7 以上で接続した場合，全ての角度情報がサーバによって丸められてから送信されるようになった．以前は，単に実数から整数へキャストし，切り詰められているだけであった．そのため，角度情報の離散化に誤差が生じていた．この変更は，モニタとログプレイヤのための dispinfo_t 構造体へ入れられる全ての値に適用される．

7.02 :

– モニタプロトコルに新しいコマンドが追加された:
(dispplayer side unum posx posy ang)
(contributed by Artur Merke)
コマンドの詳細は 5.4 節を参照．

7.00 :

– サッカーモニタへ首の角度の描画機能が追加された．(Ken Nguyen によるソース寄稿)
– プレイヤがボールや他のプレイヤと衝突した場合の視覚効果が追加された．いずれの場合も，モニタはプレイヤの周囲に黒い円を描画する．
– モニタプロトコルバージョン 2 が導入された．(バージョン 2 プロトコルの詳細は 5.5.2 節を，コマンドの詳細は 5.4 節を参照)

– ログファイルフォーマットバージョン 3 が導入された．(フォーマットバージョン 3 の詳細は 5.5.3 節を参照)
– 試合の最後のサイクルが記録されるようにロギング処理を修正した．

6 サッカークライアント

6.1 プロトコル

この節では，サッカークライアントとサッカーサーバの間のプロトコルについての概要を説明する．プロトコルについてのより詳細な解説は，サッカーサーバの節に書かれている．

init と reconnect のコマンドは，サーバが稼働しているホストマシンにおけるプレイヤの UDP ポート (デフォルト:6000) へ送信されるべきである．サーバからの応答後は，サーバによって各プレイヤに割り当てられたポートへ，適切なフォーマットでメッセージを送信しなければならない．サーバは init コマンドへの応答をこのポートから送信する (1.2.1 節を参照)．サーバへ送信する，または，サーバから送信されてくる全てのコマンドは共通の文字による文字列であり，括弧で囲われている．

6.1.1 初期化と再接続

サーバへの接続を行ないたい全てのプレイヤは，自分自身をサーバへ導入しなければならない．これは握手のようなものであり，通常は試合開始前にしか行なわれない．任意でハーフタイム時に再接続を行なうことも可能である．

初期化

クライアントは以下のフォーマットの init コマンドをサーバへ送信する:

(init *TeamName* [(version *VerNum*)] [(goalie)])

キーパは init コマンドに "(goalie)" を含めなければならない．これによって，ボールキャッチやその他の特別なキーパアクションをサーバに許可される．キーパは各チームは一人までしか使用できない点に注意 (キーパの使用は義務ではない)．

サーバは以下のフォーマットで init メッセージへ応答し，クライアントを迎え入れたことを通知する:

(init *Side UniformNumber PlayMode*)

初期化に問題があれば (2 チーム以上接続しようとした，1 チーム 11 プレイヤ以上接続しようとした，1 チームに 2 体以上のキーパを接続しようとした，などのエラーがあった場合)，エラーメッセージが返される:

(error no_more_team_or_player_or_goalie)

Side はあなたのチームがプレイするサイドであり，文字 **l**(left) または **r**(right) で表される．*UniformNumber* はプレイヤの背番号である (各チームのプレイヤは背番号によって識別される)．*PlayMode* は有効なプレイモードの一つを表す文字列である．

クライアントがバージョン 7.00 以上でサーバへ接続した場合，クライアントはサーバパラメータ，プレイヤパラメータ，プレイヤタイプ情報を更に受信するだろう (後の二つはヘテロプレイヤ特性に関するものである)．正確なフォーマットについては，付録を参照してもらいたい．

(server_param *Parameters* ...)

(player_param *Parameters* ...)

(player_type *id Parameters* ...)

これで握手は完了し，あなたのクライアントは正当なプレイヤとして識別される．

再接続

再接続 (リコネクト) することによって，試合を再スタートすること無くプレイヤのプログラムを変更できる．ただし，再接続はプレイモードが play_on 以外のとき (例えば，ハーフタイム時) にしかできない．

再接続を行うには，次のフォーマットで reconnect コマンドを送信する:

(reconnect *TeamName UniformNumber*)

すると，プレイヤは次のフォーマットの応答を受信する:

(reconnect *Side PlayMode*)

または，次のエラーのいずれかを受信する:

(can't reconnect)

これはプレイモードが play_on 以外の場合のエラーメッセージ．

(error reconnect)

再接続対象のクライアントが存在しなかった場合のエラーメッセージ．チー無名が不正であれば，次のエラーメッセージを受信することもあるだろう．**(error no_more_team_or_player_or_goalie)**

もしクライアントのバージョンを 7.00 以上で接続すれば，クライアントはここで再び追加のサーバパラメータ，プレイヤパラメータ，そしてプレイヤタイプ情報を受信する．

切断

サーバとの接続を切る前に，サーバへ bye コマンドをそうん可能である．このコマンドはフィールドからプレイヤを除外するだろう．

(bye)

このコマンドに対しては，サーバからは何の応答も得られない．

バージョンコントロール

サッカーサーバの開発は常に行われており，毎年新しい機能が追加される．これらの機能をサポートするためにプロトコルに変更と改良が加えられる．過去のクライアントとの互換性を保ち，(特に研究者のために) 新しいクライアントと同時に動作させせることを容易にするために，プロトコルバージョンコントロールのためのシステムが実装されている．全てのクライアントは，サーバが適切なフォーマットでメッセージを送ることができるように，通信プロトコルのバージョンを **init** コマンドに含めてサーバへ通知するべきである．
　しかしながら，通信プロトコルが変わらなくとも，審判とシミュレーションのルールは変更されるかもしれず，それは試合全体に影響するだろうということには注意すること．

6.1.2 制御コマンド

試合中，各プレイヤは行動コマンドを送信できる．サーバはサイクルの終了時にコマンドを実行し，受信したコマンドと前サイクルのデータによって次のサイクルをシミュレートする．

Body コマンド

プレイヤのプレイと移動の振る舞いは全て，以下で簡単に説明する body コマンドとして知られる少数のコマンドから成る．
　これらのコマンドの結果は少し複雑になり，多くのシミュレーション要素に依存する．各コマンドの実行についての詳細に関しては，サッカーサーバの節を参照．

(turn *Moment*)

Moment は -180 から 180 の度数である．このコマンドはプレイヤの体の向きを，現在の方向に対して *Moment* 度だけ回転させる．

(dash *Power*)

このコマンドはプレイヤをその体の方向へと加速させる (現在の速度の方向ではない)．*Power* は ***minpower***(使用値: -100) と ***maxpower***(使用値: 100) の間である．

(kick *Power Direction*)

Power で *Direction* の方向へボールを加速する．*Direction* はプレイヤの体の向きに相対な値を指定する．この *Power* も ***minpower*** と ***maxparam*** の間である．

(catch *Direction*)

キーパの特別コマンド: 体の向きに相対な *Direction* の方向でボールのキャッチを試みる．キャッチが成功すると，蹴られるまでボールはキーパの手の中にある．

(move *X* *Y*)

このコマンドはキックオフ前とゴール直後にのみ実行可能である．プレイヤは，X (-54 と 54 の間) と Y (-32 と 32 の間) の位置へ，一シミュレーションサイクルだけで正確に移動する．キックオフ前の配置に役立つ．

(tackle *Power*)

成功すると，*Power* でプレイヤの体の方向へボールを加速する。*Power* は ***minpower*** と ***maxparam*** の間である．

各シミュレーションサイクル内では，上記 6 コマンドのうち 1 つのみを実行可能であることに注意 (クライアントが 1 サイクル中に 1 つ以上のコマンド送ったとしても，実行されるのはサーバが最初に受信したコマンドのみである)．

(turn_neck *Angle*)

このコマンドは，他の行動コマンドと独立しており，同サイクル内での送信と同時実行が可能である．首は それまでの首角度に対して *Angle* 度だけ回転する．結果の首角度は，プレイヤの体の方向に対して ***minneckang***(使用値: -90) と ***maxneckang***(使用値: 90) の間であることに注意．

コミュニケーションコマンド

プレイヤ間でコミュニケーションを行う唯一の方法は，**say** コマンドでメッセージを配信し，**hear** センサでメッセージを受信することである．

(say *Message*)

このコマンドは *Message* をフィールド上にばらまく．そして，充分に近くに存在し (***audio_cut_dist***，使用値 50.0 メートル，で指定される)，十分な聴覚許容量を持つ全てのプレイヤが *Message* を聞くことになる．メッセージは有効な文字による文字列である．

(ok say)

コマンドが成功した．
エラーの場合，以下の応答がサーバから送信される:

(error illegal_command_form)

その他のコマンド

その他のコマンドは，通常次の 2 つの形式である:

- データ要求コマンド

  (sense_body)

sense body 情報の送信をサーバに要求する．バージョン 6.00 以上でクライアントを接続すれば，毎サイクル，自動的に sensebody 情報が送信されることに注意．

(score)

得点上納の送信をサーバに要求する．サーバの応答は次のフォーマットになる:

(score *Time OurScore OpponentScore*)

- モード変更コマンド

(change_view *Width Quality*)

プレイヤの視界パラメータを変更する．*Width* は **narrow**，**normal**，**wide** のいずれか，*Quality* は **high** または **low** のいずれかである．視覚センサから得られる情報量は，視界の広さと質に依存する．情報送信の頻度もこれらのパラメータに依存していることに注意．(例えば，quality を high から low に変更すると送信頻度は 2 倍になり，2 つの see センサ情報の間の時間は半分になる)

6.1.3 センサ情報

センサ情報は，全てのプレイヤへ定期的に送信されるメッセージである (例えば，各サイクル，または 1 サイクル半ごと)．これらの情報はサーバから自動的に送信されてくるので，情報を得るためのメッセージをサーバへ送信する必要は無い．

全てのセンサ情報に時間のラベル (*Time*) が含まれる．これはデータが送信された時点での試合のサイクル数を示している．この時間情報は非常に有益である．

視覚センサ

視覚センサは最も重要なセンサで，少し複雑である．このセンサは，プレイヤの視界内に存在する物体についての情報を返す．

情報の主要なフォーマットは:

(see *Time ObjInfo ObjInfo* . . .)

ObjInfo のフォーマットは:

(*ObjName Distance Direction* [*DistChange DirChange* [*BodyFacingDir HeadFacingDir*]])

または

(*ObjName Direction*)

各物体について得られる情報の量は，その物体までの距離に依存することに注意．物体との距離が離れるほど，得られる情報は少なくなる．*ObjInfo* に関するより詳細な情報については付録を参照．

ObjName は次のフォーマットのうちのいずれか:
(p [*TeamName* [*Unum*]])
(b)
(f *FlagInfo*)
(g *Side*)
p はプレイヤ，**b** はボール，**f** はフラッグ，**g** はゴールを意味する．
Side は 左を表す **l** か 右を表す **r** のいずれかである．*FlagInfo* については付録を参照．

聴覚センサ

聴覚センサは，フィールド上で聞くことができるメッセージを返す．聴覚メッセージは，オンラインコーチ，審判，または他のプレイヤから送られてくる．

バージョン 7 までのフォーマットは次のようになる:

> **(hear *Time Sender Message*)**

Sender は次のうちの 1 つである:
self: 送信者が自分自身のとき．
referee: 送信者が審判のとき．
online_coach_l または **online_coach_r**
Direction: 送信者が自分以外のプレイヤのとき，送信者の相対方向が代わりに返される．

バージョン 8 以降では，送信者が自分以外のプレイヤのとき，送信者の情報 (敵味方，背番号) も含まれる．

Body センサ

Body センサは，残りスタミナ，視界モード，スピードのような，プレイヤの状態を返す．情報は各サイクルの開始時にプレイヤへと送信される:

> **(sense_body *Time* (view_mode { high | low } { narrow | normal | wide }) (stamina *Stamina Effort*) (speed *Speed Angle*) (head_angle *Angle*) (kick *Count*) (dash *Count*) (turn *Count*) (say *Count*) (turn_neck *Count*) (catch *Count*) (move *Count*) (change_view *Count*))**

最後の 8 パラメータは受信されたコマンドのカウントである．失われた，または遅延したメッセージを監視するために使用する．

6.2 クライアントの作り方

この節では，最初のサッカークライアントプログラムを書くための詳細な説明を行う．

6.2.1 サンプルクライアント

サッカーサーバの配布パッケージには，sampleclient という，非常にシンプルなサッカークライアントプログラムが含まれている．サッカーサーバをコンパイルするときに，このサンプルプログラムも自動的にコンパイルされる．

sampleclient はスタンドアローンなクライアントではない: 標準入力からのコマンドをサーバへとリダイレクトし，サーバからの情報を標準出力へとリダイレクトする，単純な'パイプ'である，それゆえ，ユーザが sampleclient を実行しただけでは何も起きない．ユーザはキーボードからコマンドをタイプし，ターミナルに表示されるセンサ情報を読まなければならない．(実際には，1 秒あたりに 17 個ものセンサ情報 (see 情報と sense_body 情報) が送られてくるため，センサ情報を読むことは不可能である)

sampleclient は，クライアントがなすべきこととクライアントがサーバから何を受信するかを理解するのに有益である．

sampleclient の使用法

ここでは，sampleclient の典型的な使用方法を述べる．

1. rcssclient を起動する

   ```
   % rcssclient -server SERVERHOST
   ```

   SERVERHOST はサッカーサーバが実行されているホスト名である．

   すると，プログラムはユーザの入力待ちになる．

   サッカーサーバが通常とは異なるポート番号 (例えば，標準のポート番号である 6000 の代わりに，6005) を用いている場合は，ユーザは次のような形式を使用しなければならない．

   ```
   % rcssclient -server SERVERHOST -port 6005
   ```

2. キーボードから init コマンドをタイプする．

   ```
   (init MYTEAMNAME (version 7))
   ```

   MYTEAMNAME はユーザが使用したいチーム名である．

   すると，プレイヤがフィールド上に現れる．同時に，プログラムはサーバから送られるセンサ情報をターミナルへ出力し始める．典型的な出力は次のようになる:

   ```
   send 6000 : (init foo (version 7))
   recv 1567 : (init r 1 before_kick_off)
   recv 1567 : (server_param 14.02 5 0.3 0.4 0.1 60 1 1 4000 45 0 0.3 0.5 ...
   recv 1567 : (player_param 7 3 3 0 0.2 -100 0 0.2 25 0 0.002 -100 0 0.2 ...
   recv 1567 : (player_type 0 1 45 0.4 5 0.006 0.3 0.7 0 0 1 0.6)
   recv 1567 : (player_type 1 1.16432 28.5679 0.533438 8.33595 0.00733326 ...
   recv 1567 : (player_type 2 1.19861 25.1387 0.437196 5.92991 0.00717675 ...
   recv 1567 : (player_type 3 1.04904 40.0956 0.436023 5.90057 0.00631769 ...
   ```

```
recv 1567 : (player_type 4 1.1723 27.7704 0.568306 9.20764 0.00746072 ...
recv 1567 : (player_type 5 1.12561 32.4392 0.402203 5.05509 0.00621539 ...
recv 1567 : (player_type 6 1.02919 42.0812 0.581564 9.53909 0.00688457 ...
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
recv 1567 : (see 0 ((g r) 61.6 37) ((f r t) 49.4 3) ((f p r t) 37 27) ...
recv 1567 : (sense_body 0 (view_mode high normal) (stamina 4000 1) ...
...
```

最初の行，“send 6000 : (init foo (version 7))” は，clientがサーバへ送信したものを報告している．二行目，”recv 1567 : (init r 1 before_kick_off) は，サーバからの最初の応答を報告している．ここでサーバは，割り当てられたプレイヤは右サイドのチーム (r) で，その背番号は 1 であること，そして現在のプレイモードが before_kick_off であることを，クライアントへ伝えている．その次の 9 行は server_param と player_param で，シミュレーションで使用される様々なパラメータを伝えている．最後に，サーバは通常のセンサ情報，sense_body と see の送信を開始する．サーバはこれらのセンサ情報を 100ms 毎に，または 150ms 毎に送信するため，client はこれらの情報を絶え間なく出力し続ける．

3. プレイヤを初期位置へ配置するために，move コマンドをタイプしなさい．プレイヤはフィールド外側のベンチに現れる．ユーザは 次のような move コマンドによって，プレイヤを初期位置へと移動させる必要がある:

   ```
   (move -10 10)
   ```

   すると，プレイヤは (-10,10) の位置へ移動する．

   前に述べたように，client プログラムは絶え間なくセンサ情報を出力する．そのため，client が ncurses モードで起動していなければ，ユーザは自身がタイプした文字列を見ることができない．その場合，ユーザは何も見ずにコマンドをタイプしなければならない．[1]

4. サッカーモニタの ‘Kick-Off’ ボタンをクリックしなさい．すると，試合が開始される．ユーザは各センサ情報内の時間データ (see または sense_body 情報の最初の数字) が増加していく様子を見ることができる．

5. その後，ユーザは任意の通常コマンド，turn，dash，kick などを使用できる．例えば，次のようにタイプすると，プレイヤを右に回転させることができる:

   ```
   (turn 90)
   ```

   次のようにタイプすると，プレイヤは最大パワーで前方へダッシュすることができる:

   ```
   (dash 100)
   ```

[1] ユーザは任意のファイルやプログラムへ出力をリダイレクトできる．例えば，“% client SERVERHOST > /dev/null” と実行することで，出力を /dev/null へとリダイレクトし，無効にすることができる．すると，ユーザは自身がタイプした文字列を見ることができる．

プレイヤがボールに充分近ければ，次のようにタイプすると，プレイヤは 50 のパワーで自身の左方向へボールをキックすることができる:

```
(kick 50 -90)
```

センサ情報の出力は延々と続くため，ncurses を用いていない場合は，ユーザは入力内容が見えない状態でコマンドをタイプしなければならないことを再度注意しておく．

サンプルクライアントの全体構造

`sampleclient` の構造は単純である．`client` が行う処理は概ね次のようになる:

1. UDP ソケットを開き，サーバのポートに接続する．(`init_connection()`)
2. read-write ループに入り，次の 2 つの処理が並列に実行される．
   - ユーザの入力を標準入力 (通常，キーボード) から読み，それをサーバへ送信する (`send_message()`)．
   - サーバからセンサ情報を受信し (`receive_message()`)，それを標準出力 (通常，コンソール) へ出力する．

並列実行を実現するために，`sampleclient` は `select()` 関数を用いている．この関数は，単一プロセス内で複数のソケットとストリームからの入力を待つことを可能にする．`select()` が呼び出されると，ソケットかストリームのひとつが入力データを得るまで待ち，度のソケット又はストリームがデータを得たかをを伝える．`select()` のより詳しい使用方法については，man ページやマニュアルを参照してもらいたい．

`sampleclient` における重要なヒントは，クライアントがサーバからセンサ情報を受信すると，クライアントはサーバのポート番号を変更しなければならない，ということである．これは，サーバがクライアントからの `init` コマンドを受信すると，新しいポートをクライアントへ割り当てるためである．この処理は，"`client.c`" 内の次のステートメントで行われている:

```
        printf( "recv %d : ", ntohs(serv_addr.sin_port));
+       sock->serv_addr.sin_port = serv_addr.sin_port ;
        buf[n] = '\0' ;
```

6.2.2 シンプルクライアント

完全なサッカークライアントを開発するためにユーザがなすべき事は，'頭脳' 部分のコードを書くことである．頭脳部分は，前説で述べた `sampleclient` を使ってユーザが行ったのと同じことを実行する．言い替えると，ユーザは，受信したセンサ情報に基づいてコマンド文字列を生成するコードを書かなければならない，ということである．

もちろん，これは簡単な仕事ではない (多くの研究者が研究対象として RoboCup に取り組んでいる)．そして，その実装には様々な方法がある．単純に言って，プレイヤクライアントを開発するためには，ユーザは次の関数を実現する必要がある:

[Sensing] センサ情報の解析: 前説で示したように，サーバは様々な情報を S 式で送信する．そのため，クライアントは S 式を構文解析する必要がある．そして，特定の内部表現を得るために，クライアントは情報を分析しなければならない．例えば，プレイヤの位置やフィールドの状態を推測するために，クライアントは視覚情報を分析する必要がある．何故なら，視覚情報は物体 (フィールド上の目印となる物体や移動物体) の相対的な位置しか含まないからである．

[Action Interval] コマンド送信間隔の制御: 体の制御を行うコマンド (turn，dash，kick など) は 100ms に一回しかサーバによって受けつけられない．そのため，クライアントはコマンド送信前に適切な間隔を待つ必要がある．

[Parallelism] センサと行動の並列実行: サッカーサーバはセンサ情報とコマンドを非同期に処理する．そのため，クライアントはセンサプロセス (センサ情報の取扱い) と行動プロセス (コマンド送信の制御) を並列に実行する必要がある．

[Planning] プレイのプランの作成: クライアントは，センサ情報を用いて適切なコマンド列を生成する必要がある．もちろん，これがサッカークライアント開発における最終的な目標である!!

単独動作するプレイヤのシンプルな例，`sclient1` と `sclient2` をここで示す．これらはボールを追いかけて敵ゴールへとキックするだけである．ソースは次の場所から得られる:

`ftp://ci.etl.go.jp/pub/soccer/client/noda-client-2.0.tar.gz`

このサンプルでは，上で述べられた機能が次のように実現されている:

- *Sensing* 関数に関しては，いずれのサンプルにおいても `class BasePlayer`, `class FieldState`, `estimatePos` 関数の共通機能を使用する．これらの機能によって，サンプルプログラムは以下のことを実行する:
  - サーバと接続したソケットからデータを受信し，
  - S 式のデータを解析し，
  - 内部データフォーマット (`class SensorInfo`) へと式を解釈し，
  - 受信データが視覚情報であれば，自身と田の物体の位置を推定する．

  詳細については，ソースコードを参照してもらいたい．
- *Action Interval* と *Parallelism* 関数に関しては，サンプルはそれぞれ異なる方法を使用している．一つ目のサンプル `sclient1` は `select()` 関数のタイムアウトを使用する．二つ目のサンプル `sclient2` はマルチスレッド (pthread) を使用する．これらは後で説明される．
- *Planning* 関数に関しては，いずれのサンプルも以下のような非常に単純なプランナを持つ:
  - 過去 10 サイクルの間ボールを見ていない，またはボールの位置を過去 10 サイクル推定できていなければ，周囲を見渡す．
  - ボールがキック可能であれば，敵ゴールへとボールをキックする．
  - それ以外の場合，ボールを追いかける (ボールへターンしダッシュする)．

sclient1

Action Interval と *Parallelism* を実現するために，sclient1 は select() 関数のタイムアウト機能を使用する．

プログラムの重要な部分は MyPlayer::run() にある．以下にソースコードの一部を示す:

```
//--------------------------------------
// enter main loop

SocketReadSelector selector ;

TimeVal nexttic ;  // indicate the timestamp for next command send
nexttic.update() ; // set nexttic to the current time.

while(True) {

   //--------------------------------------------------
   // setup selector

   selector.clear() ;
   selector.set(socket) ;

   //--------------------------------------------------
   // wait socket input or timeout (100ms) ;

   Int r = selector.selectUntil(nexttic) ;

   if(r == 0) { // in the of timeout. (no sensor input)
      doAction() ;  // enter action part
      nexttic += TimeVal(0,100,0) ; // increase nexttimetic 100ms
   } else {  // got some input
      doSensing() ; // enter sensor part
   }
}
```

class SocketReadSelector は select() の機能を抽象化するクラスであり，"itk/Socket.h" で定義されている．"Int r = selector.selectUntil(nexttic) ;" の行では，プログラムはソケットからの入力か nexttic で示されるタイムアウトを待つ．nexttic は次の tic のタイムスタンプ（シミュレーションステップ）を保持する．関数は，タイムアウトが発生すれば 0 を返し，ソケットに受信データがあればその数を返す．タイムアウトの場合，プログラムは doAction() を呼び出し，コマンドが生成されてサーバへと送信される．それ以外の場合（受信データがある場合），doSensing() が呼び出され，センサ情報が処理される．

`sclient2`

Action Interval と *Parallelism* を実現するために ,`sclient2` は POSIX スレッド (pthread) を使用する .

プログラムの重要な部分は `MyPlayer::run()` にある . 以下にソースコードの一部を示す:

```
//----------------------------------------
// fork sensor thread

forkSensor() ;

//----------------------------------------
// main loop

while(True) {
   if        (!isBallSeenRecently(10)) {
      //----------------------------
      // if ball is not seen recently
      // look around by (turn 60)

      for(UInt i = 0 ; i < 6 ; i++) {
         turn(60) ;
      }
   } else if (kickable()) {
      ...
   }
}
```

"`forkSensor() ;`" ステートメントはセンサ情報を受信、解析するための新しいスレッドを起動する (センサスレッドの動作は "`SimpleClient.*`" と"`ThreadedClient.*`" で定義されている .) 次に , メインスレッドは "*chasing the ball and kick to the goal*" の行動列をを生成するメインループへ入る . *Sensing* 関数はセンサスレッドで並列に扱われるため , メインスレッドがセンサ入力を扱う必要は無い .

行動の間隔を 100ms に保つために ,`sclient2` は `ThreadedPlayer::sendCommandPre()` という関数によって次のシミュレーションステップまで待機する . この関数は "`ThreadedPlayer.cc`" で以下のように定義されている:

```
Bool ThreadedPlayer::sendCommandPre(Bool bodyp) {
   cvSend.lock() ;

   if(bodyp) {
      while(nextSendBodyTime.isFuture())
         cvSend.waitUntil(nextSendBodyTime) ;
   }
```

```
        while(nextSendTime.isFuture()) {
            cvSend.waitUntil(nextSendTime) ;
        }

        return True ;
    } ;
```

この関数では，MutexCondVar cvSend は，上記 sclient1 における select() のタイムアウトと似た機能を提供する．(MutexCondVar は状態変数 (pthread_cond_t) と mutex (pthread_mutex_) の組合せであり，"itk/MutexCondVar.h" で定義されている．) 関数はプレイヤがサーバへコマンドを送信する直前に呼び出され，また，nextSendBodyTime は次のシミュレーションステップのタイムスタンプを示すので，スレッドは次の tic にコマンドを送信するために待機する．

6.2.3 Tips

サッカークライアントプログラムを開発するために Tips をここに集めた．

- デバッグ作業はチーム開発における主要な問題である．よって，容易なデバッグ方法の発見に挑戦してもらいたい．
- ある状態におけるプログラムの変数を確認する単純で良い方法は，プログラムにコアダンプさせるために **abort()** コマンドやいくつかの **assert** を使用することである．そして，gbd によって core をデバッグする．
- サーバから受信した、または，サーバへ送信した全てのメッセージのログを記録する．これはデバッグに非常に有益である．
- 初心者の場合，ソケットやメッセージ解析処理ためには既存のライブラリを使用すると良い．
- init コマンドでバージョン番号をサーバへ渡すことを覚えておくこと．この手続きは任意であるが，デフォルトバージョンである 3.00 は推奨されない．
- キャッチ確立が 1.00 であってもキャッチコマンドが失敗するかもしれない．何故なら，位置に関するセンサ情報は誤差を含むからである．
- 最初に遭遇する重大な問題はタイミング管理かもしれない．クライアントとサーバを同期させるには多くの方法がある．一つの単純な方法は，受信した sense_body 情報を使用することである．
- ネットワークの速度に注意を払う．タイミングの同期が強力でなければ，混雑した遅いネットワークやプロセスリソースが不足している環境では，クライアントは正しく動作しないだろう(例えば，多くのクライアントを遅いマシンで走らせている場合)．この場合，プレイヤは古い位置を観測し，その位置情報で行動しようとするかもしれない．結果として混乱が起こる．

- フラッグの主な使用目的はプレイヤ自身のフィールド上での位置測定である．あなたの最初のクライアントはフラッグを無視し，相対位置でプレイするかもしれない．しかし，近い将来，位置測定モジュールが必要になるかもしれない．この問題のための多くのライブラリが既に存在している．
- センサ情報解析において，バッファの終端をチェックすべきである．センサ情報はS式を採用している．しかし，センサデータがバッファよりも長いと，閉括弧が失われて式が完了されないかもしれない．この場合，式を単純に解析すればプログラムはコアダンプを起こすかもしれない．

7 コーチ

7.1 イントロダクション

コーチとはプレイヤへの補助を行う特殊なクライアントである．コーチにはオンラインコーチとトレーナの 2 種類がある．トレーナは' オフラインコーチ' とも呼ばれるが，区別のためにここでは' トレーナ' と呼ぶ．

7.2 トレーナとオンラインコーチの違い

トレーナは細かい試合の制御が行え，開発段階でのみ使われる．一方，オンラインコーチは公式試合でも使用される．トレーナは，試合の自動実行などの制御が必要な場合に便利である。オンラインコーチは，試合中にプレイヤへ追加のアドバイスや情報を与えるために用いられる。

プレイヤの開発中，例えばドリブルやキックのようなスキルの獲得に機械学習を適用する場合，トレーニングセッションを自動的に作成できれば便利である。そのため、トレーナは以下の能力を持つ。

- プレイモードを制御可能．
- 聴覚メッセージをブロードキャスト配信可能．このメッセージは任意のプレイヤへ向けられたコマンドや何らかの情報で構成でき，ユーザ定義の文法が使用できる．
- プレイヤとボールをフィールド上の任意の位置へ移動させられ，その方向や速度も設定可能．
- 物体の情報をノイズ無しで得られる．

これらの能力についての詳細は 7.3 を参照．

オンラインコーチは，試合を観察し，プレイヤへアドバイスを与えることを意図されている．そのため，オンラインコーチの能力は以下の制限を持つ．

- プレイヤとのコミュニケーションが可能．
- 物体の情報をノイズ無しで得られる．

コーチが各クライアントを中央集権的に制御することを防止するために， コミュニケーションの制約が設定されている (7.7 節参照)．オンラインコーチは，敵のモデリングや試合の分析を行い，見方へ戦略的な情報を与えるための良いツールである．コーチはノイズ無しの完全情報が得られ，かつリアルタイムな意思決定の要求が少ないため，戦略について熟考する時間が得らやすいと予想される．オンラインコーチの詳細については 7.6 節を参照．

7.3 トレーナ

7.3.1 サーバとの接続と審判機能の有無

デフォルトでは，サーバ内蔵の審判モジュール (4.7 節参照) は有効である．トレーナが完全に試合を制御する場合は，審判モジュールを無効にする設定を行なわなければならない．審判モジュールを無効にすると，プレイモードが変更されず，プレイモードに応じたプレイヤの強制移動も行なわれない．そのため，トレーナが独自のルールでこれらの状況に対処しなければならない．

トレーナクライアントが使用されることは，サーバ起動時にサーバに知らされなければならない．トレーナを使用し，審判モジュールを無効にする場合は，オプション-*coach*[1]をサーバのコマンドラインオプションに追加する．server.conf の *coach* による設定でも良い．

審判を有効にしたままでトレーナを接続させたい場合は，オプション-*coach_w_referee* をコマンドラインオプションに追加するか，server.conf の *coach_w_referee* を有効にする．

サーバがトレーナモードで起動されると，トレーナクライアントが接続できる UDP ソケットが用意される．デフォルトのポート番号は 6001 である[2]. 別のポート番号を使う必要がある場合は，*coach_port* で新しいポート番号を設定できる (4.9.1 節参照)．

7.4 コマンド

トレーナとオンラインコーチは以下で述べるコマンドセットを使用できる．項目は 3 つのカテゴリに分けられている．最初のカテゴリはトレーナのみが使用可能なコマンド，二番目はオンラインコーチのみが使用可能な制限を持つコマンド，三番目はトレーナとオンラインコーチの両方で使用されるコマンドである．

7.4.1 トレーナのみが使用可能なコマンド

- **(change_mode PLAY_MODE)**

  プレイモードを PLAY_MODE に変更する．PLAY_MODE は 4.7.1 節で定義されているモードのいずれかでなければならない．ほとんどの場合，プレイモードの要求に対して，サーバはプレイモードを変更するのみであることに注意．通常，ボールの位置は変更されないが，いくつかのケースではプレイヤは移動させられる．例えば，フリーキックとキックインの時，プレイヤがボールから一定距離以内の位置にいれば，プレイヤはその範囲外へと移動させられる．'`before_kick_off`' へ変更した場合，プレイヤは自陣サイドへと移動させられる．その他のプレイモードについては各自で実験してみること．

  サーバからは以下の応答が得られる:

  - **(ok change_mode)**
    コマンドが成功した．

[1]注: このパラメータは' オフラインコーチ' についてのみのものである．オンラインコーチと混同しないように．

[2]オンラインコーチのためのデフォルトポート番号は 6002 である.

  - **(error illegal_mode)**
    指定したモードが不正だった．
  - **(error illegal_command_form)**
    PLAY_MODE 引数が省略された．

- **(move OBJECT X Y [VDIR [VEL$_X$ VEL$_Y$]])**

  OBJECT を絶対位置 (X, Y) へと移動させる．OBJECT にはプレイヤまたはボールを指定する．VDIR が指定されると，OBJECT の絶対方向が VDIR へと変更される (実際に有効になるのは OBJECT がプレイヤの場合のみ)．更に， VEL$_X$ と VEL$_Y$ が指定されると，物体の速度がこの値に応じて変更される．OBJECT のフォーマットは 4.3 節で述べられているものとは異なる．省略形ではなく完全な名前が使用され，プレイヤは **(player TEAM UNUM)**，ボールは **(ball)** となる．従って，有効な move コマンドは **(move (player myteam 1) -20.0 -20.0)** のようになる．

  トレーナは常に左手座標を使用する．

  サーバからは以下の応答が得られる:

  - **(ok move)**
    コマンドが成功した．
  - **(error illegal_object_form)**
    OBJECT の指定が不正だった．
  - **(error illegal_command_form)**
    位置，向き，あるいは速度の指定が不正だった．

- **(check_ball)**

  サーバにボールの位置のチェックを要求する．四つの位置状態が定義されている:

  - **in_field**
    ボールがフィールド内にある．
  - **goal_l**
    ボールが左サイドのゴール内にある．
  - **goal_r**
    ボールが右サイドのゴール内にある．
  - **out_of_field**
    ボールがそれら以外の位置にある．

  ゴールの間を通っていなくても ‘`goal_l`’ または ‘`goal_r`’ の状態になりうる点に注意．

  サーバからは以下の応答が得られる:

  - **(ok check_ball TIME BALLPOSITION)**
    BALLPOSITION は上記の状態のいずれかである．

- **(start)**

  サーバをスタートさせ, 例えば，プレイモードを 'kick_off_l' にセットする．このコマンドによって，モニタのキックオフボタンを押すことと同じ効果を得られる．

  トレーナが init コマンドを送っていない場合，トレーナからの最初のコマンドは，コマンドの種類に関係なくサーバをスタートさせる．

  サーバからは以下の応答が得られる:

  - **(ok start)**
    コマンドが成功した．

- **(recover)**

  プレイヤの stamina, recovery, effort, hear capacity を試合開始時の値にリセットする．

  サーバからは以下の応答が得られる:

  - **(ok recover)**
    コマンドが成功した．

- **(ear MODE)**

  トレーナが聴覚情報を受け取るかどうかを切り替える．MODE は **on** か **off** のいずれかでなければならない. **(ear on)** が送信されると，サーバは全ての聴覚情報をトレーナへ送信する．フォーマットは表 7.3 を参照．**(ear off)** が送信されると，サーバはトレーナへの聴覚情報の送信を停止する．

  サーバからは以下の応答が得られる:

  - **(ok ear on)**
    (ok ear off)
    いずれもコマンドが成功したことを意味する．
  - **(error illegal_mode)**
    MODE が **on** または **off** ではなかった．
  - **(error illegal_command_form)**
    MODE 引数が省略された．

7.4.2 制限付きでオンラインコーチにも使用可能なコマンド

- **(init (version VERSION))** : トレーナ用
- **(init TEAMNAME (version VERSION))** : オンラインコーチ用

  これらのコマンドは，トレーナまたはコーチが使用するプロトコルのバージョンを通知する．オンラインコーチの場合，コーチがどちらのチームに属するかを TEAMNAME によって指定しなければならない．コーチの接続は，そのチームのプレイヤが少なくとも一人以上接続した後でなければならない．

トレーナは init コマンドの実行を要求されない．しかし，init コマンドの実行は推奨される．init コマンドを使用しなかった場合，古いプロトコルによってサーバとの通信が行なわれるからである．

トレーナのデフォルトポート番号は 6001，オンラインコーチのためのデフォルトポート番号は 6002 である．

サーバからは以下の応答が得られる:

- **(init ok)**
  トレーナのコマンドが成功した場合．
- **(init SIDE ok)**
  オンラインコーチのコマンドが成功した場合．SIDE は'l' か'r' のいずれか．

- **(say MESSAGE)**

  トレーナとオンラインコーチは同じ構文でこのコマンドを使うことができるが，オンラインコーチの場合は更に制限が課せられる．詳細は 7.6.2 節を参照．

  このコマンドは，トレーナの場合は全てのクライアントへ，オンラインコーチの場合は味方のみへと MESSAGE を配信する．トレーナに関しては，MESSAGE のフォーマットはプレイヤの場合と同じである．文字列は，長さが *say_coach_msg_size*(4.9.1 節を参照) 以下で，英数字と記号 ().+*/?<>_ で構成されなければならない．

  プレイヤがこれらのメッセージを聞く際のフォーマットは 4.3.1 節を参照.

  サーバからは以下の応答が得られる．

  - **(ok say)**
    コマンドが成功した．
  - **(error illegal_command_form)**
    MESSAGE のフォーマットが不正だった．

- **(change_player_type TEAM_NAME UNUM PLAYER_TYPE)**：トレーナ用

- **(change_player_type UNUM PLAYER_TYPE)**：オンラインコーチ用

  これらのコマンドは，チーム TEAM_NAME の背番号 UNUM のプレイヤをタイプ PLAYER_TYPE のヘテロジニアスプレイヤ (4.6 節を参照) へと変更するために使用される．PLAYER_TYPE は 0 から 6 の数字である．0 はデフォルトのプレイヤタイプを示す．オンラインコーチの場合，プレイヤタイプの変更は自チームのみに限定されるため，引数 TEAM_NAME は省略される．

  トレーナは，交替回数制限 (3 人:*subs_max*) を守らなくて良い．

  オンラインコーチがプレイヤ交替を行なう場合の制限については， 7.6.3 節を参照．

  サーバからは以下の応答が得られる:

  - **(warning no_team_found)**
    指定チームが存在しない．

- **(error illegal_command_form)**
  change_player_type コマンドの引数が文字列と二つの整数でない，またはコマンドが括弧で閉じられていない．
- **(warning no_such_player)**
  指定チームに指定背番号のプレイヤが存在しない．
- **(ok change_player_type TEAM UNUM TYPE)**
  コマンドが成功した．

更に，サーバは次の応答をオンラインコーチへ送ることがある:

- **(warning cannot_sub_while_playon)**
  プレイモードが ‘`play_on`’ である.
- **(warning no_subs_left)**
  試合中に既に 3 回 ($subs\_max$ で指定される) 交替を行なっている．
- **(warning max_of_that_type_on_field)**
  プレイヤタイプがデフォルトではなく，そのタイプのプレイヤが既に 3 人 (pt_max で指定される) フィールド上に存在する．
- **(warning cannot_change_goalie)**
  キーパーのプレイヤタイプを変更しようとした．

サーバは味方へ次の情報を送る:

- **(change_player_type UNUM TYPE)**

敵 (敵のコーチを含む) へは次の情報を送る:

- **(change_player_type UNUM)**

7.4.3 トレーナとオンラインコーチが使用可能なコマンド

- **(look)**

  このコマンドは，フィールド上に存在する以下の物体の位置情報をクライアントへ与える:

  - 左右両サイドのゴール．
  - ボール．
  - サーバと接続中の全てのプレイヤ．

  トレーナとオンラインコーチは，チームサイドに関係なく左手座標の情報を受け取る．これは，コーチは左サイドチームが用いる絶対座標情報を受け取ると言うことである．一般にプレイヤは絶対情報を受け取らず，負の x 方向が自陣方向に向かうように位置測定を行なうのが普通である．

  サーバからは以下の応答が得られる:

- **(ok look TIME (OBJ$_1$ OBJDESC$_1$) (OBJ$_2$ OBJDESC$_2$) . . .)**
  OBJ$_j$ は上で述べた任意のオブジェクトになり得る．物体の名前の構成に関する情報は 4.3 節を参照．OBJDESC$_j$ は以下の形式を持つ:
  * ゴール：X Y
  * ボール：X Y DELTA$_X$ DELTA$_Y$
  * プレイヤ：X Y DELTA$_X$ DELTA$_Y$ BODYANGLE NECKANGLE [POINTING_DIRECTION]

  トレーナとオンラインコーチのいずれであっても，座標は常に左手座標系である．

  トレーナもしくはコーチが視覚情報を定期的に受信する場合は，**(eye on)** コマンドを使わなければならない．

- **(eye MODE)**

  MODE は **on** か **off** のいずれかでなければならない．**(eye on)** が送信されると，サーバからクライアントへ向けて，**(see_global . . .)** 情報 (7.5 節を参照) が 100 ミリ秒毎に自動送信されるようになる．この送信間隔は，*send_vi_step* パラメータによって変更可能である．**(eye off)** が送信されると，サーバは視覚情報の自動送信を停止する．この場合，視覚情報が必要であれば，トレーナとコーチは **(look)** コマンドによって能動的に視覚情報を要求しなければならない．

  サーバからは以下の応答が得られる:

  - **(ok eye on)**
    (ok eye off)
    いずれもコマンドが成功したことを意味する．
  - **(error illegal_mode)**
    MODE が **on**，または **off** ではなかった．
  - **(error illegal_command_form)**
    MODE 引数が省略された．

- **(team_names)**

  このコマンドは，両チームの名前と，チームがそれぞれどちらのサイドでプレイをしているかを，トレーナまたはコーチに向けて送信する．

  サーバからは以下の応答が得られる:

  - **(ok team_names [(team l TEAMNAME$_1$) [(team r TEAMNAME$_2$)]])**
    チームがすでに接続しているかどうかに依存して，片方または両方のチーム名がサーバから発信される．最初に接続したチームが左サイドになる点に注意すること．

7.4.4 オンラインコーチのみが使用可能なコマンド

- **(team_graphic (X Y "XPM line" ... "XPM line"))**
  オンラインコーチは，256x64 XPM としてチームグラフィックをサーバに送信できる．各 **team_graphic** コマンドは 8x8 のタイルを送信する．X と Y はこのタイルの座標である．よって，X の範囲は 0 から 31，Y の範囲は 0 から 7 となる．各 XPM line は 8x8 XPM タイルからの 1 行である．サーバに接続したモニタは，コーチが **team_graphic** を送信する毎に message board によって以下のメッセージを受信する:
  (team_graphic_l—r (X Y "XPM line" ... "XPM line"))
  サーバからは以下の応答が得られる:
  - **(ok team_graphic X Y)**
    タイル毎に，その受信を確認するためにこの文字列が送信される．

7.5 サーバからのメッセージ

上で述べたコマンドへの応答以外にも，サーバはいくつかのコマンドをトレーナとオンラインコーチへ送信する．バージョン 7 以上のクライアントがサーバへ接続すると (**init** コマンドの version 引数が 7.0 以上の場合)，プレイヤクライアントと同様に以下のパラメータメッセージを受信する．

- **(server_param ...)** 1 回
- **(player_param ...)** 1 回
- **(player_type ...)** 各プレイヤタイプにつき 1 回

パラメータメッセージについての詳細は 4.2.2 節を参照．

クライアントが (**eye on**) を送ることで視覚情報の受信を選択した場合，クライアントは 100 ミリ秒 (*send_vi_step*) 毎に以下のフォーマットでメッセージを受信する:

$$(see\_global (OBJ_1 OBJDESC_1)(OBJ_2 OBJDESC_2)\ldots)$$

OBJ_j は物体の名前を示す．名前の構成方法についての情報は，表 4.3 を参照．$\mathrm{OBJDESC}_j$ は以下の形式を持つ:

- ゴール：X Y
- ボール：X Y DELTA_X DELTA_Y
- プレイヤ：X Y DELTA_X DELTA_Y BODYANGLE NECKANGLE [POINTING_DIRECTION]

構文は (**look**) コマンドへの応答と同じである．よって，座標値は常に左手座標である．

クライアントが聴覚情報を受信するために (**ear on**) コマンドをサーバに送信した場合，クライアントは審判とプレイヤからの全ての聴覚情報を受信するようになる．現在，2 種類の hear メッセージがある:

- **(hear TIME referee MESSAGE)** "play_on" や "free_kick_left" のような全ての審判のメッセージ．審判からの有効なメッセージのリストについては 4.7 節を参照．
- **(hear TIME (p "TEAMNAME" NUM) "MESSAGE")** 全てのプレイヤからのメッセージ．メッセージは引用符で囲まれることに注意．

プレイヤのコミュニケーション能力についてのより詳しい解説は 4.3.1 節を参照．

7.6 オンラインコーチ

7.6.1 イントロダクション

オンラインコーチは，公式試合でサーバに接続できる，特権を与えられたクライアントである．オンラインコーチは，フィールド上の物体に関するノイズ無しの完全情報を得られる．コーチに関する研究を促進させるために，2001 年からコーチ競技が開催されている．これによって，通常のチーム開発を行なわない研究グループでも，オンラインコーチに焦点をあてることで RoboCup への参加が可能になった．更に，さまざまなチームに対して単体のコーチの適用を可能にするために，プレイヤとのコミュニケーションに用いられる標準コーチ言語 (CLang) が開発された．オンラインコーチが使用するコマンドとサーバから送られてくるメッセージについての詳細は 7.4 節と 7.5 節を参照．

7.6.2 プレイヤとのコミュニケーション

バージョン 7.00 以前では，オンラインコーチは英数字と記号 ((().+*/?<>_) からなる短い (128 文字，*say_coach_msg_size*) メッセージのみを，プレイモードが 'play_on' 以外の場合に限って話すことができた．このタイプのメッセージは "freeform" メッセージとして今も存在している．しかし，現在は他の標準メッセージタイプが使用可能である．バージョン 8.05 以降，'play_on' の間でも freeform メッセージを送信できる特定のインターバルが設定されている．'play_on' 状態の 600 サイクル (*freeform_wait_period*)) ごとに，20 サイクル (*freeform_send_period*) の間に限り，コーチは freeform メッセージを送信できる．例えば，420 サイクルにプレイモードが 'play_on' に変わり，そのまま 'play_on' の状態が続いたとすると，コーチが freeform メッセージを送信できるのは 1020 から 1040 の間，1620 から 1640 の間，となる．コーチは 1 試合につき *say_coach_cnt_max* 回，freeform メッセージを送信できる．このメッセージの長さは *say_coach_msg_size* 以下でなければならない．試合が延長戦になった場合，6000 サイクル分 (または通常の試合の長さ分) ごとに *say_coach_cnt_max* 回のメッセージが追加される．許可されるメッセージ回数は累積的で，コーチが freeform メッセージの使用回数を残していれば残り回数は延長戦へと持ち越される．コーチが最大使用回数を越えて freeform メッセージを送信しようとした場合，サーバは **(error said_too_many_messages)** をコーチへと送信する．

コーチ競技おいては freeform メッセージの使用は許可されておらず，互換性の理由から CLang のみが使用されるという点に注意されたい．

標準コーチ言語には 3 つのメッセージタイプ:rule, define, delete がある．コーチによるプレイヤの細かい制御を防止するために，以下の方法でコミュニケーションの制限が行なわれている．

コーチは，300 サイクル (*clang_win_size*) ごとに各メッセージタイプを 1 回送信できる．各メッセージタイプの送信許可回数は，パラメータ *clang_define_win*, *clang_del_win*, *clang_rule_win* によって変更可能である（4.9.1 節を参照)．メッセージは 50 サイクル (*clang_mess_delay*) 後にプレイヤへと送信される．プレイモードが 'play_on' でなければ，遅延することなくサイクルごとに 1 メッセージ (*clang_mess_per_cycle*) がプレイヤへと送信される．プレイモードが 'play_on' 以外の時に送られたメッセージは，使用回数制限のための回数にカウントされない．例えば，デフォルト設定では，試合中断中にコーチはサイクルごとに 1 メッセージを，遅延無しでプレイヤへ伝えることができる．各タイプのメッセージがサーバに受信されると，同じ順番でそれらがプレイヤへと送信されることをサーバは保証する．

言語の文法はメッセージの長さに関する制限を指定していない．しかし，現実には標準言語によるメッセージの長さを 8154 文字以上にすることはできない (プレイヤへと送信されるメッセージの最大長が 8K バイトであるため)．

コーチ言語 (CLang) の最初のバージョンはサーババージョン 7.x のために開発された．そして，サーババージョン 8 のために言語の拡張が行なわれた．このため，コーチからのメッセージを受け取るクライアントは，どのバージョンの CLang をサポートするかをサーバに明示しなければならない．CLang バージョンの指定は

(clang (ver MIN MAX))

コマンドによって行なわれる．MIN と MAX は非負の整数でサポートする CLang のバージョンの最小値と最大値を意味する．このコマンドを送信しないクライアントは，コーチのメッセージを受信しない．サーバはコーチのメッセージのバージョンを判断し，プレイヤによってサポートされていない全てのメッセージを除外する．メッセージが除外されると，プレイヤは

(hear TIME online_coach_left—right (unsupported_clang))

を受信する．コーチは各プレイヤがサポートするバージョンについての情報を

(clang (ver (PLAYER_NAME) MIN MAX))

メッセージによって受け取る．バージョン 7 のプレイヤを最新のサーバで用いる場合は，**(clang (ver 7 7))** コマンドを送信しなければならないことになる．

標準コーチ言語についての詳細は 7.7 節で述べられている．

7.6.3 プレイヤタイプの変更

change_player_type コマンド（7.4 節を参照) を使うことで，オンラインコーチはプレイヤタイプを変更できる．プレイモードが 'before_kick_off' であれば，回数は制限されない．もちろん，この変更はヘテロジニアスプレイヤに関する一般的なルールを守らなければならない（4.6 節を参照)．キックオフ後は，プレイヤタイプの変更は 'play_on' 以外の時に限り 3 回 (*subs_max*) まで許可される．

サーバからの応答については 7.4 節の **change_player_type** コマンドの記述を参照．

注意: プレイヤクライアントが接続する前に行なわれたプレイヤ交替に関する情報は，味方チームに関しては **(change_player_type UNUM TYPE)** メッセージ，敵チームに関しては **(change_player_type UNUM)** メッセージによって通知される．

7.7 標準コーチ言語

7.7.1 一般的特性

異なる研究グループによって開発されたチームへのコーチの適用を可能にするために，標準コーチ言語は開発された．設計目的の一つは，プレイヤとコーチの間で解釈に違いが現れないような，明解な意味論を持つことである．低レベルの概念を組み合わせることで新しい高レベルの概念を構築することができるという言語設計がなされている．

更に，プレイモードが 'play_on' 以外の場合に限り，自由な文字列を使用可能な freeform メッセージによってコーチとプレイヤのコミュニケーションが可能である．詳細は 7.6.2 節を参照．コーチを他のチームへも適用することを計画しているならば，freeform メッセージは他のチームには理解されないであろうことを知っておかなければならない．[3]

以下で述べる言語仕様は，コミュニティによって開発された言語を改良，拡張したバージョンで，バージョン 8.x 以降のサーバでサポートされる．初期バージョンの CLang もまだサポートされているが，使用は推奨されない．初期バージョン言語の完全な仕様はバージョン 7 のサーバマニュアルに記述されている．マルチエージェント研究のための有益なツールにするために，興味を持った研究社たちによって CLang の開発が続けられることが期待される．

言語のいくつかのコンセプトは，Unilang[14](定義やいくつかのアクションなど) と SFL[12](変数や位置の算術演算など) からもたらされた．

flex と bison(GNU による lex と yacc の実装) を用いることで，サーバ自身が全てのコーチのメッセージを解析し，C++のクラス階層に基づいて単純な表現を構築できることに注意．コーチのメッセージを解析するために，サーバ内のコードを編集して使用してもよい．その場合，特に `coach_lang*` を参照．

7.7.2 言語の発話例

CLang の基本的な考えは，条件と指示を対応づけるルールとして戦略と行動を記述することである．各ルールは，状況を示す要素 (条件*:condition*) と，条件が true の時に適用可能な指示*:directives* のリストによって構成される．ルールは，プレイヤがどのように行動すべきかを教えるアドバイスとしてだけでなく，ある状況で敵がどのように振舞うかを記述した情報などとしても利用できる．CLang では，ルールは ID を持ち，コーチは後でそれらを参照することができる．

以下に単純なルールの例を示す．このルールは，5 番のプレイヤがボールを持ちフィールド中央に位置していれば，味方の 11 番へパスすることをアドバイスしている:

```
(define
  (definerule
    MyRule1
    direc
    (
      (and
        (bowner our 5)
```

[3]訳者注: コーチ競技では，freeform メッセージの使用は禁止されている．

```
        (bpos (rec (pt -10 -10) (pt 10 10)))
      )
      (do our 5 (pass 11))
    )
  )
)
```

各基本要素についての詳細は後で説明される．今は，ルールに ID "MyRule1" が割り当てられ，指示として定義されていることに関する説明を行なう (観測された振舞いを記述したモデルルールとは異なる)．**bowner** は，コーチのチームの 5 番プレイヤがボール所有者であることを決定する．**bpos** は，矩形によってボールの位置を指定する．最後に，5 番プレイヤが味方の 11 番へとパスを行なうよう指示が行なわれている．CLang において，**(pass 11)** は行動:*action* で，**(do our 5 (pass 11))** は 指示:*directive* である．

デフォルトでは，ルールはオフに設定されている．よって，コーチは **(rule (on MyRule1))** のようなメッセージを送ることでルールを有効化しなければならない．

それでは，言語の概念について，より詳細に解説を行おう．

7.7.3 メッセージタイプの概要

標準コーチ言語には 4 タイプのコーチメッセージ: Rule, Define, Delete, Freeform がある．これらの目的とフォーマット，そしていくつかの例をこの節で示す．

以下のフォーマット記述において大文字の要素は非終端記号を意味している．これらの定義は 7.7.7 節に記されている．

Define-message: Define メッセージは CLang で最も複雑なメッセージである．何故なら，コーチがプレイヤと共有しようとする要素 (条件，指示，領域，行動，ルールなど) を定義し，組み合わせるためのメッセージだからである．要素に ID を割り当てて定義することによって，後のメッセージ内でその要素を ID によって参照して使用することができるようになる．

Conditions: 条件を定義するフォーマット: **(definec CLANG_STR CONDITION)**
Example: **(definec "Defense" (bowner opp 0))**
任意の敵プレイヤがボールを所有しているという条件を定義．

Actions: 行動を定義するフォーマット: **(definea CLANG_STR ACTION)**
Example: **(definea "Pass7" (pass 7))**

Directives: 指示を定義するフォーマット: **(defined CLANG_STR DIRECTIVE)**
Example: **(defined "Pass10to11" (do our 10 (pass 11)))**
10 番のプレイヤが 11 番のプレイヤへパスを行うという指示を定義．

Regions: 領域を定義するフォーマット: **(definer CLANG_STR REGION)**
Example: **(definer "OURHALF" (rec (pt -52.5 -34) (pt 0 34)))**
自陣ハーフの矩形を定義．

Rules: ルールを定義するフォーマット: **(definerule CLANG_VAR model RULE)** or **(definerule CLANG_VAR direc RULE)**
Example: **(definerule Rule1 direc ((playm bko) (do our 7 (pos (pt -20 20)))))**
7番のプレイヤは before_kick_off 時に指定位置へ移動するというルール．
ルール定義に関しては，7.7.4 を参照．

Rule-message: Rule メッセージは，事前に定義したルールの有効/無効を切替えるために使われる．ルール定義後，デフォルトではそのルールは off になっている．

Format: **(rule ACTIVATION_LIST)**

Example: **(rule (on rule2) (off rule1))**

Delete-message: Delete メッセージは，ルールが再び使用されることが無く，メモリから削除可能であることをプレイヤへ伝えるためのメッセージである．このことは，ルールの削除後，その ID が他のネストされたルール定義 (7.7.4 節参照) 内に現れてはならないことも意味する．

Format: **(delete ID_LIST)**

Examples: **(delete Rule1) (delete (Rule1 Rule2)) (delete all)**

それぞれ，一つのルールを削除，リスト内の二つのルールを削除，全てのルールを削除を意味する

Freeform-message: Freeform メッセージは自由な文字列で，7.6.2 節で述べられた制限内で送信可能である．

Format: **(freeform "STRING")**

STRING は引用符で囲まれなければならない点に注意．

7.7.4 ルールの定義

ルールの定義は CLang において重要な部分である．この節では，ルール定義についてより詳細に解説する．ルールは条件と行動を含む指示のリストから構成されることを覚えておくこと．

上記のように，ルール定義のフォーマットは **(definerule DEFINE_RULE)** であり，以下の要素を用いる:

```
<DEFINE_RULE> : <CLANG_VAR> model <RULE>
              | <CLANG_VAR> direc <RULE>

<RULE> : (<CONDITION> <DIRECTIVE_LIST>)
```

```
| (<CONDITION> <RULE_LIST>)
| <ID_LIST>
```

各ルールは **CLANG_VAR** の定義に沿う名前を割り当てられる．更に，ルールには二つのモードのいずれかを指定できる．**model** は観察された振舞いを記述するルールを定義し，**direc** は特定の条件で実行する指示を定義する．
実際のルール内容の指定にはいくつかの方法がある:

- (CONDITION DIRECTIVE_LIST)

  直接的な方法．7.7.3 節での例はこのフォーマットによるものである．CONDITION は状況を意味し，DIRECTIVE_LIST は特定の指示を意味する．リストは任意の人数のプレイヤへの指示を含むことが可能で，同じプレイヤへ複数の指示を行なうこともできる点に注意．後者の場合，どの指示を守るかはプレイヤの責任となる．

- (CONDITION RULE_LIST)

  ルールをより拡張された戦略へと組み合わせるための非常に強力なフォーマットである．RULE_LIST 内の各ルールは既に条件を含み，この定義はネストされたルールを形成する．例えば，いくつかのルールを同時に有効にするために使うことができる．ディフェンダーのホームポジションを指定するいくつかのルール (2 番のプレイヤに対する pos2a と pos2b，3 番のプレイヤに対する pos3a と pos3b) が既に存在しているとすると，

  (definerule defenseformation direc ((bowner our {0}) (pos2a pos3a)))

  そして

  (definerule offenseformation direc ((bowner opp {0}) (pos2b pos3b)))

  を定義することで，ルールが有効になる時 (ここでは，どちらのチームがボールを所有しているかが条件となる) を指定することができる．このような定義を評価するために，外側の条件は内部の条件へ論理 and によって分配結合されることになっている．例えば，pos2a が以下のような定義であれば，

  ((time > 20) (do our {2} (pos (pt -40 10))))

  上記の定義は以下のルールを生成することになる．

  ((and (bowner our {0}) (time > 20)) (do our {2} (pos (pt -40 10))))

- ID_LIST

  上記のフォーマットと同様に，既に存在するルールを組み合わせることができる．二つのルールが以下のように定義されているとすると:

  (definerule position2 direc ((true) (home (pt -40 -10))))

  (definerule mark2 direc ((bowner opp {10}) (mark 10)))

  2 番のプレイヤの振舞いとして以下のように結合できる:

  (definerule player2 direc (position2 mark2))

7.7.5 要素の意味と構文

これまでに述べたフォーマットで使用された非終端記号の意味と構文を解説する．

ルールは左側に条件を持ち，右側に行動のセットを持つ．各ルールは本質的に if-then 命令文とみなすことができる:

```
if CONDITION
then { DIRECTIVE_1 DIRECTIVE_2 ... }
```

プレイヤのプログラム内では，コーチから与えられた全てのアドバイスを小さなルールベースとして容易に表現することができる．アドバイスへの追従は，現在の世界の状態を条件に照合し，指示の実行を試みることで容易に行なえるだろう．注: 現在の状況に複数の条件が適用可能で，かつ指示内容が異なる場合，指示の選択はプレイヤの責任である．プレイヤは指示内容についても慎重に考慮すべきである点にも注意．例えば，合致した指示が 5 番プレイヤへのパスのみであったが，5 番プレイヤへのパスコースが敵に消されていた場合，ボールを持ったプレイヤは指示を無視する必要があるかも知れない．

- Conditions:
  条件は，状態を記述したアトミックな命題の論理結合によって形成される:
  - **(true)**
    常に true.
  - **(false)**
    常に false.
  - **(ppos TEAM UNUM_SET INT INT REGION)**
    最初の INT が MINIMUM，二つ目の INT が MAXIMUM を意味する．チーム TEAM のプレイヤで，背番号が UNUM_SET 内の MINIMUM 以上，MAXIMUM 以下のプレイヤが領域 REGION 内に存在すると，true となる．領域と背番号についての詳細は後で述べる．TEAM は ”our” か ”opp” のいずれかである．コーチのメッセージは自チームのプレイヤにのみ配信されるので，TEAM の曖昧性はない．
  - **(bpos REGION)**
    ボールが領域 REGIONn 内に存在すれば true．
  - **(bowner TEAM UNUM_SET)**
    チーム TEAM のプレイヤで，背番号が UNUM_SET に該当するプレイヤが一人以上ボールをコントロールできる状態であれば，true．ボール所有者とは，最後にボールに接触した (例えば，ボールをキッカブルエリア内に入れた) プレイヤのことである．ボールがプレイヤの制御を離れた後でも，これは変わらない．
  - **(playm PLAY_MODE)**
    プレイモードが PLAY_MODE であれば true．PLAY_MODE の正当な値については 7.7.7 節を参照．

- **(COND_COMP)**
  時間，得点差，味方または敵の得点を定数値と比較できる．演算子として $<$ $>$ $<=$ $==$ $!=$ $>=$ が使用可能．
  Examples: (time $>$ 20) (2 $>=$ opp_goals)
- **unum CLANG_VAR UNUM_SET**
  CLANG_VAR がインスタンス化されていれば，変数 CLANG_VAR によって示される背番号が UNUM_SET 集合内に含まれるかどうかがチェックされる．変数がまだ束縛されていなければ，特定の集合へ束縛される．

論理結合は以下のようになる:

- **(and CONDITION$_1$ CONDITION$_2$ ...CONDITION$_n$)**
- **(or CONDITION$_1$ CONDITION$_2$ ...CONDITION$_n$)**
- **(not CONDITION)**

例: "敵の 3 番プレイヤが領域 X 内に存在し，ボールを制御している"という条件は以下のようになる．

(and (ppos opp {3} X) (bowner opp {3}))

- Directives:

  指示は基本的に個々のプレイヤ集合のための行動のリストで，二つの形式を取る:

  - **(do TEAM UNUM_SET ACTION_LIST)** (肯定モード: プレイヤはこれらの行動を取るべきである)
  - **(dont TEAM UNUM_SET ACTION_LIST)** (否定モード: プレイヤはこれらの行動を取るべきではない)

  もし肯定モードの行動が互いに排他的であれば，どちらに従うべきかの判断はプレイヤの責任である．**model** モードにおけるルールでは，敵味方を含めたプレイヤのプランや振舞いについての知識を指示によって伝達する．

- Actions:

  - **(pos REGION)**
    プレイヤ自身の位置を REGION 内に移動させる．
  - **(home REGION)**
    プレイヤのデフォルト位置を REGION 内にする．この指示は主にチームフォーメーションの指定を意図している．
  - **(mark UNUM_SET)**
    UNUM_SET に含まれる敵プレイヤをマークする．
  - **(markl REGION)**
    現在のボール位置から REGION へのパスコースをマークする．

- **(markl UNUM_SET)**
  現在のボール位置から UNUM_SET に含まれる敵プレイヤへのパスコースをマークする．
- **(oline REGION)**
  プレイヤまたはチームのオフサイドトラップラインを REGION にセットする．
- **(htype TYPE)**
  プレイヤのヘテロジニアスタイプが TYPE である．TYPE 番号については 4.6 節で述べられている．値が-1 の場合，プレイヤはヘテロジニアスタイプについての情報を消去すべきである．
- **(pass REGION)**
  REGION 内にいるプレイヤにボールをパスする．
- **(pass UNUM_SET)**
  UNUM_SET に含まれるプレイヤにボールをパスする．
- **(dribble REGION)**
  REGION へとドリブルする．
- **(clear REGION)**
  REGION からボールをクリアする．これは，ボールを REGION の外へと蹴り出すことを意味する．
- **(shoot)**
  ゴールへとボールをシュートする．
- **(hold)**
  ボールをホールドする．例えば，現在の位置に立ったまま敵プレイヤからボールをキープする．
- **(intercept)**
  ボールを追跡し，制御下に置くことを試みる．
- **(tackle UNUM_SET)**
  UNUM_SET に含まれるプレイヤ (またはボール所有者?) にタックルする．

- Regions:

  REGION として以下のいずれもが使用可能:

  - a POINT
    後でより詳細に定義する．
  - **(rec POINT**$_1$ **POINT**$_2$**)**
    ピッチのラインに平行な辺を持つ矩形を定義する．
  - **(tri POINT**$_1$ **POINT**$_2$ **POINT**$_3$**)**
    引数の点によって構成される三角形を定義する．

- **(arc POINT RADIUS_SMALL RADIUS_LARGE ANGLE_BEGIN ANGLE_SPAN)**
  共通の中心点を持つ二つの円にはさまれたドーナツ型の弧を定義する．引数は，二つの円の共通の中心点とそれぞれの半径，弧の開始角度と角度範囲の大きさを意味する．例えば，半径 r の円は “(arc (pt 0 0) 0 r 0 360)” と定義でき，U字型の領域は “(arc (pt 0 0) 5 10 0 180)” と定義できる．
- **(null)**
  ヌル (空の) 領域．
- **(reg REG$_1$ REG$_2$ . . . REG$_n$)**
  引数の領域の和集合によって構成される領域を定義する．

POINT として以下のいずれもが使用可能:

- **(pt X Y)**
  X と Y は実数かつグローバル座標である．絶対位置 (X,Y) を意味する．
- **(pt ball)** 現在のボールの絶対位置．
- **(pt TEAM UNUM)** チーム TEAM('our' または'opp') の UNUM 番プレイヤの現在位置．UNUM は変数である点に注意．
- **(POINT$_1$ OP POINT$_2$)**
  二つの位置を数学的に結合し，新しい位置を生成する．POINT$_i$ 自身が算術演算子から構成することができるため，再帰的な構造を形成することができる．演算子は自然な方法で定義される．例えば，

$$\textbf{(pt } X_1\ Y_1\textbf{)}\ \textbf{OP}\ \textbf{(pt } X_2\ Y_2\textbf{)} = \textbf{(pt } X_1\textbf{OP}X_2\ Y_1\textbf{OP}Y_2\ \textbf{)}$$

  となる．OP は $+\ -\ *\ /$ のいずれかである．

これらの相対的な位置の使用は，“ボールの位置へ移動”，“ボールから 10m 以内に味方が二人いるならば” などといったような概念の表現を容易にする．

オンラインコーチは，チームのサイドに関係なく常に左手座標で視覚情報を受け取ることに注意．しかし，コーチが右サイドのチームにメッセージを送る場合は，メッセージ内では右手座標を用いなければならない．左手座標から右手座標へと座標値を変換するには，-1 倍するいだけで良い．

- UNUM SETS:

  Unum set はプレイヤの背番号の集合である．順番が重要でなく，サーバによって変更されるかも知れないという意味で，これらは集合である．集合の中に 0 が含まれていれば，集合は全てのプレイヤ 1 - 11 を含む．集合は変数を含むことができる．

  Format: $\{$ NUM$_1$ NUM$_2$ . . . NUM$_n$ $\}$

- Variables:

  技術的には，UNUM が現れるところであればどこであっても変数を使うことができる．しかし，変数がインスタンス化されているか導き出せることを確認することが

重要である．スコープは最も込み入った規則である．例えば，以下の例では **X** のスコープは 4 行目である．

```
1  (definerule rule1 model
2    (bowner our {0})
3    ((true)           (do our {5} (mark 11)))
4    ((bowner our {X}) (do our {X} (shoot)))
5  )
```

この例は，変数がどのようにしてインスタンス化できるかも示している: 定数として UNUM を持つ条件においてのみ (POINT 内の UNUM は条件の UNUM としてカウントされない)，変数は採用される．その値はどの背番号が条件を true にするかをチェックすることでセットされる．例では，**X** はボール所有者の背番号によってインスタンス化される．**ppos** のような条件では，背番号の集合として変数をインスタンス化することができる:

(ppos our {X} 1 11 REGION)

この例では，**X** は REGION 内に存在するプレイヤの背番号の集合としてインスタンス化されなければならない．次のようなインスタンス化はサポートされないことに注意:

(ppos our {5} 1 1 (rec (pt ball) (pt our {X})))

7.7.6 参考リソース

- CLang コーパスは実際の CLang メッセージの例を蓄積している:

  http://www-2.cs.cmu.edu/~ pfr/soccer/clang_corpus.html

- The Multi-Agent Modeling Special Interest Group (MAMSIG) は，コーチング可能なチームとオンラインコーチのバイナリやソースコードを提供する:

  http://www.cl-ki.uni-osnabrueck.de/~ tsteffen/mamsig

- The Coach-mailing-list では CLang の詳細，競技ルール，コーチング手法についての議論を行なっている:

  http://robocup.biglist.com/coach-l/

7.7.7 構文

標準コーチ言語の完全な文法:

```
<MESSAGE> : <FREEFORM_MESS> | <DEFINE_MESS> | <RULE_MESS> | <DEL_MESS>

<RULE_MESS> : (rule <ACTIVATION_LIST>)
```

```
<DEL_MESS> : (delete <ID_LIST>)

<DEFINE_MESS> : (define <DEFINE_TOKEN_LIST>)

<FREEFORM_MESS> : (freeform <CLANG_STR>)

<DEFINE_TOKEN_LIST> : <DEFINE_TOKEN_LIST> <DEFINE_TOKEN>
                                        | <DEFINE_TOKEN>

<DEFINE_TOKEN> :  (definec <CLANG_STR> <CONDITION>)
                                | (defined <CLANG_STR> <DIRECTIVE>)
                                | (definer <CLANG_STR> <REGION>)
                                | (definea <CLANG_STR> <ACTION>)
                                | (definerule <DEFINE_RULE>)

<DEFINE_RULE> :   <CLANG_VAR> model <RULE>
                                | <CLANG_VAR> direc <RULE>

<RULE> : (<CONDITION> <DIRECTIVE_LIST>)
                | (<CONDITION> <RULE_LIST>)
                | <ID_LIST>


<ACTIVATION_LIST> : <ACTIVATION_LIST> <ACTIVATION_ELEMENT>
                                        | <ACTIVATION_ELEMENT>

<ACTIVATION_ELEMENT> : (on|off <ID_LIST>)


<ACTION> : (pos <REGION>)
                        | (home <REGION>)
                        | (mark <UNUM_SET>)
                        | (markl <UNUM_SET>)
                        | (markl <REGION>)
                        | (oline <REGION>)
                        | (htype <INTEGER>)
                        | <CLANG_STR>
                        | (pass <REGION>)
                        | (pass <UNUM_SET>)
                        | (dribble <REGION>)
                        | (clear <REGION>)
                        | (shoot)
                        | (hold)
                        | (intercept)
```

```
                        | (tackle <UNUM_SET>)

<CONDITION> : (true)
                        | (false)
                        | (ppos <TEAM> <UNUM_SET> <INTEGER> <INTEGER> <REGION>)
                        | (bpos <REGION>)
                        | (bowner <TEAM> <UNUM_SET>)
                        | (playm <PLAY_MODE>)
                        | (and <CONDITION_LIST>)
                        | (or <CONDITION_LIST>)
                        | (not <CONDITION>)
                        | <CLANG_STR>
                        | (<COND_COMP>)
                        | (unum <CLANG_VAR> <UNUM_SET>)
                        | (unum <CLANG_STR> <UNUM_SET>)

<COND_COMP> : <TIME_COMP>
                        | <OPP_GOAL_COMP>
                        | <OUR_GOAL_COMP>
                        | <GOAL_DIFF_COMP>

<TIME_COMP> : time <COMP> <INTEGER>
                        | <INTEGER> <COMP> time

<OPP_GOAL_COMP> : opp_goals <COMP> <INTEGER>
                        | <INTEGER> <COMP> opp_goals

<OUR_GOAL_COMP> : our_goals <COMP> <INTEGER>
                        | <INTEGER> <COMP> our_goals

<GOAL_DIFF_COMP> : goal_diff <COMP> <INTEGER>
                        | <INTEGER> <COMP> goal_diff

<COMP> : < | <= | == | != | >= | >

<PLAY_MODE> : bko | time_over | play_on | ko_our | ko_opp
              | ki_our | ki_opp | fk_our | fk_opp
              | ck_our | ck_opp | gk_opp | gk_our
                          | gc_our | gc_opp | ag_opp | ag_our


<DIRECTIVE> : (do|dont <TEAM> <UNUM_SET> <ACTION_LIST>)
                        | <CLANG_STR>
```

```
<REGION> : (null)
                        | (arc <POINT> <REAL> <REAL> <REAL> <REAL>)
                        | (reg <REGION_LIST>)
                        | <CLANG_STR>
                        | <POINT>
                        | (tri <POINT> <POINT> <POINT>)
                        | (rec <POINT> <POINT>)

<POINT> : (pt <REAL> <REAL>)
                | (pt ball)
                | (pt <TEAM> <INTEGER>)
                | (pt <TEAM> <CLANG_VAR>)
                | (pt <TEAM> <CLANG_STR>)
                | (<POINT_ARITH>)

<POINT_ARITH> : <POINT_ARITH> <OP> <POINT_ARITH>
                                | <POINT>

<OP> : + | - | * | /


<REGION_LIST> : <REGION_LIST> <REGION>
                        | <REGION>

<UNUM_SET> : { <UNUM_LIST> }

<UNUM_LIST> : <UNUM>
                        | <UNUM_LIST> <UNUM>

<UNUM> : <INTEGER> | <CLANG_VAR> | <CLANG_STR>

<ACTION_LIST> : <ACTION_LIST> <ACTION>
                                | <ACTION>

<DIRECTIVE_LIST> : <DIRECTIVE_LIST> <DIRECTIVE>
                                | <DIRECTIVE>

<CONDITION_LIST> : <CONDITION_LIST> <CONDITION>
                                | <CONDITION>

<RULE_LIST> : <RULE_LIST> <RULE>
                        | <RULE>

<ID_LIST> : <CLANG_VAR>
```

```
                     | (<ID_LIST2>)
                     | all

<ID_LIST2> : <ID_LIST2> <CLANG_VAR>
                     | <CLANG_VAR>

<CLANG_STR> : "[0-9A-Za-z\(\)\.\+\-\*\/\?\<\>\_ ]+"

<CLANG_VAR> : [abe-oqrt-zA-Z_]+[a-zA-Z0-9_]*

<TEAM> : our | opp
```

パラメータ名	使用値	default	説明
coach_port	6001	6001	トレーナの接続ポート
say_msg_size	512	256	トレーナまたはコーチが使用できる freeform メッセージの最大長
say_coach_cnt_max	128	128	オンラインコーチが使用できる freeform メッセージの最大回数
send_vi_step	100	100	コーチの視覚情報受信間隔
clang_win_size	100	100	コーチのメッセージの使用インターバル
clang_define_win	1	1	許可インターバル内での define メッセージの使用可能回数
clang_rule_win	1	1	許可インターバル内での rule メッセージの使用可能回数
clang_del_win	1	1	許可インターバル内での delete メッセージの使用可能回数
clang_mess_delay	50	50	コーチのメッセージの配信遅延サイクル
clang_mess_per_cycle	1	1	play_on 以外のプレイモードでのサイクル毎のメッセージ配信数

トレーナからサーバへ	サーバからトレーナへ
(init (version VERSION)) VERSION ::= 実数	trainer: (init ok)
(change_mode PLAY_MODE) PLAY_MODE ::= プレイモードの一つ	(ok change_mode) (error illegal_mode) (error illegal_command_form)
(move OBJECT X Y [VDIR [DELTA_X DELTA_Y]]) OBJECT ::= 物体名の一つ X ::= -52.5–52.5 Y ::= -34–34 VDIR ::= -180–180 DELTA_X, DELTA_Y ::= [float]	(ok move) (error illegal_object_form) (error illegal_command_form)
(check_ball)	(ok check_ball TIME BPOS) TIME ::= シミュレーションサイクル BPOS ::= in_field | goal_SIDE | out_of_field SIDE ::= l | r
(start) (recover)	(ok start) (ok recover)
(change_player_type TEAM_NAME UNUM PLAYER_TYPE) TEAM_NAME ::= 文字列 UNUM ::= 1–11 PLAYER_TYPE ::= 0–6	(warning no_team_found) (error illegal_command_form) (warning no_such_player) (ok change_player_type TEAM UNUM TYPE)
(ear MODE) MODE ::= on | off	(ok ear on) (ok ear off) (error illegal_mode) (error illegal_command_form)

Table 7.1: トレーナとサーバの通信プロトコル

コーチからサーバへ	サーバからコーチへ
(init TEAMNAME [COACHNAME] (version VERSION)) VERSION ::= 実数 TEAMNAME ::= 文字列 COACHNAME ::= 文字列	(init SIDE ok) SIDE ::= l | r
(change_player_type UNUM PLAYER_TYPE) UNUM ::= 1–11 PLAYER_TYPE ::= 0–6	(warning no_team_found) (error illegal_command_form) (warning no_such_player) (ok change_player_type TEAM UNUM TYPE) (warning cannot_sub_while_playon) (warning no_subs_left) (warning max_of_that_type_on_field) (warning cannot_change_goalie)

Table 7.2: オンラインコーチとサーバの通信プロトコル

<table>
<tr><th>クライアントからサーバへ</th><th>サーバからクライアントへ</th></tr>
<tr><td>(say MESSAGE)
(see Section 7.4.2)</td><td>(ok say)
(error illegal_command_form)</td></tr>
<tr><td>(look)</td><td>(ok look TIME
(OBJ$_1$ OBJDESC$_1$)
(OBJ$_2$ OBJDESC$_2$) . . .)
OBJ$_j$ ::= object name
(see Section 4.3
OBJDESC$_j$ ::= X Y |
X Y DELTA$_x$ DELTA$_y$ |
X Y DELTA$_x$ DELTA$_y$
BODYANG NECKANG</td></tr>
<tr><td>(eye MODE)
MODE ::= on | off</td><td>(ok eye on)
(ok eye off)
(error illegal_mode)
(error illegal_command_form)</td></tr>
<tr><td>トレーナ/コーチが eye モードを on にしている場合 (下記の “eye” コマンドを参照)，このメッセージは send_vi_step ミリ秒毎に自動的に送られる．</td><td>(see_global TIME
(OBJ$_1$ OBJDESC$_1$)
(OBJ$_2$ OBJDESC$_2$) . . .)</td></tr>
<tr><td>トレーナがこのメッセージを得るには ‘ear’ コマンドを使わなければならない．オンラインコーチは常にこのメッセージを受け取る．</td><td>(hear TIME referee MESSAGE)
(hear TIME
(p ”TEAMNAME” NUM)
”MESSAGE”)
TIME ::= メッセージが送られたサイクル
TEAMNAME ::= 文字列
NUM ::= 1–11
MESSAGE ::= 文字列</td></tr>
<tr><td>(team_names)</td><td>(ok team_names
[(team l TEAMNAME1)
[(team r TEAMNAME2)]])</td></tr>
</table>

Table 7.3: トレーナ/コーチとサーバとの通信プロトコル

8 References and Further Reading

8.1 General papers

[1] Minoru Asada and Hiroaki Kitano, editors. *RoboCup-98: Robot Soccer World Cup II*. LNAI 1604. Springer, Berlin, Heidelberg, New York, 1999.

[2] Hans-Dieter Burkhard, Markus Hannebauer, and Jan Wendler. AT Humboldt — Development, Practice and Theory. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *Lecture Notes in Computer Science*, pages 357–372. RoboCup Federation, Springer–Verlag, 1997.

[3] Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors. *Team Descriptions Simulation League RoboCup'99*, Stockholm, Sweden, July 1999.

[4] John F. Kennedy. Urgent National Needs. Congressional Record – House (25 may 1961), 1961.

[5] Hiroaki Kitano, editor. *Proceedings of the IROS-96 Workshop on RoboCup*, Osaka, Japan, November 1996.

[6] Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I.* Springer Verlag, Berlin, 1998.

[7] Hiroaki Kitano, Minoru Asada, Yasou Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The Robot World Cup Initiative. In *Proc. of IJCAI-95 Workshop on Entertainment and AI/Alife*, pages 19–24, 1995.

[8] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents '97)*, pages 340–347, New York, 5–8 1997. ACM Press.

[9] Stefan Lanser, Christoph Zierl, Olaf Munkelt, and Bernd Radig. MORAL - A Vision-based Object Recognition System for Autonomous Mobile Systems. In *7^{th} International Conference on Computer Analysis of Images and Patterns, Kiel*, pages 33–41. Springer–Verlag, September 1997.

[10] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Coevolving Soccer Softbot Team Coordination with Genetic Programming. In Hiroaki Kitano, editor, *Proceedings of the RoboCup−97 Workshop at the 15^{th} International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 115–118, 1997.

[11] Alan Mackworth. *On Seeing Robots*, chapter 1, pages 1–13. World Scientific Press, 1993.

[12] Andreas G. Nie, Angelika Honemann, Andres Pegam, Collin Rogowski, Leonhard Hennig, Marco Diedrich, Philipp Hugelmeyer, Sean Buttinger, and Timo Steffens. the osnabrueck robocup agents project. Technical report, Institute of Cognitive Science, Osnabrueck, 2001.

[13] Itsuki Noda, Shoji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. Overview of RoboCup-97. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 20–41. Springer–Verlag, 1997.

[14] Luis Paulo Reis and Nuno Lau. Coach unilang - a standard language for coaching a (robo)soccer team. In Andreas Birk, Silvia Coradeshi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*. Springer, Berlin, 2002. (to appear).

[15] The goals of RoboCup. by RoboCup Federation on `http://www.robocup.org/overview/22.html`, 2000. Verified on 12$^{\text{th}}$ February 2001.

[16] Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors. *RoboCup-2000: Robot Soccer World Cup IV*, Berlin, 2001. Springer Verlag. To appear.

8.2 Doctoral Theses

[17] Klaus Dorer. *Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Freiburg, December 1999. (German only).

[18] Johan Kummeneje. RoboCup as a Means to Research, Education, and Dissemination. Ph. Lic. Thesis, March 2001. Department of Computer and Systems Sciences, Stockholm University and the Royal Institute of Technology.

[19] Peter Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, December 1998.

8.3 Undergraduate and Master's Theses

[20] Fredrik Heintz. RoboSoc a System for Developing RoboCup Agents for Educational Use. Master's thesis, IDA 00/26, Linköping university, Sweden, March 2000.

[21] Jan Murray. My goal is my castle – Die höheren Fähigkeiten eines RoboCup-Agenten am Beispiel des Torwarts. Studienarbeit, Universität Koblenz-Landau, Germany, March 1999. (German only).

[22] Jan Murray. Soccer Agents Think in UML. Diploma thesis, Universität Koblenz-Landau, 2001.

[23] Oliver Obst. RoboLog: Eine deduktive Schnittstelle zum RoboCup Soccer Server. Diploma thesis, Universität Koblenz-Landau, February 1999. (German only).

8.4 Platforms to start building team upon

8.5 Education-related articles

8.6 Machine Learning

[24] Sebastian Buck and Martin A. Riedmiller. Learning situation dependent success rates of actions in a robocup scenario. In *Pacific Rim International Conference on Artificial Intelligence*, page 809, 2000.

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

8.7 Decision Making

[26] V.S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*. MIT Press, Cambridge, Massachusetts, 2000.

8.8 Other supporting documents

[27] Laws of the games. by FIFA on `http://www.fifa.com`, 2000. Verified on 12$^{\text{th}}$ February 2001.

[28] W.R. Stevens. *UNIX Network Programming*. Prentice Hall, 1990.

8.9 Team Descriptions

8.9.1 1996

8.9.2 1997

8.9.3 1998

[29] Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-98 Champion Simulator Team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. RoboCup Federation, Springer–Verlag, 1998.

8.9.4 1999

[30] Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-99 Simulator Team. In Silvia Coradeschi, Tucker Balch, Gerhard Kraetzschmar, and Peter Stone, editors, *Team Descriptions Simulation League RoboCup'99*, pages 7–11. RoboCup Federation, Linköping University Electronic Press, 1999.

8.9.5 2000

8.9.6 2001

A GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such parties remain in full compliance.

A.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

B Soccerserver

B.1 Soccerserver Parameters

Table B.1: `server.conf` 内の調整可能なパラメータ

名前	Default 値	使用される値 in `server.conf`	説明
goal_width	7.32	14.02	ゴールの幅
player_size		0.3	プレイヤの半径
player_decay		0.4	プレイヤの速度減衰率
player_rand		0.1	プレイヤの ノイズファクタ
player_weight		60.0	プレイヤの重さ
player_speed_max		1.0	プレイヤの最大スピード
player_accel_max		1.0	プレイヤの最大加速
stamina_max		4000.0	最大スタミナ値
stamina_inc_max		45.0	最大スタミナ回復値
recover_dec_thr		0.3	recovery 減少閾値率
recover_min		0.5	最小 recovery 値
recover_dec		0.002	recovery 減少幅
effort_dec_thr		0.3	effort 減少閾値率
effort_min		0.6	最小 effort 値
effort_dec		0.005	effort 減少幅
effort_inc_thr		0.6	effort 増加閾値率
effort_inc		0.01	effort 増加幅
kick_rand		0.0	キック ノイズ率
team_actuator_noise			チームごとに特定のアクチュエータ ノイズを加えるかど うかのフラグ
prand_factor_l			左チームの prand にかける 追加ファクタ
prand_factor_r			右チームの prand にかける 追加ファクタ
kick_rand_factor_l			左チームの kick_rand にかける 追加ファクタ
kick_rand_factor_r			左チームの kick_rand にかける 追加ファクタ
ball_size		0.085	ボールの半径
ball_decay		0.94	ボールの速度減衰率
ball_rand		0.05	ボールの ノイズファクタ
ball_weight		0.2	ボールの重さ
ball_speed_max		2.7	ボールの最大スピード
ball_accel_max		2.7	ボールの最大加速
dash_power_rate		0.006	ダッシュパワー効果率
kick_power_rate		0.027	キックパワー効果率
kickable_margin		0.7	キッカブルマージン
control_radius			control radius

Table B.1: (continued)

名前	Default 値	使用される値 in server.conf	説明
catch_probability		1.0	キーパのキャッチ成功確率
catchable_area_l		2.0	キーパのキャッチエリアの長さ
catchable_area_w		1.0	キーパのキャッチエリア幅
goalie_max_moves		2	キーパのキャッチ後の最大 move 回数
maxpower		100	最大パワー
minpower		-100	最小パワー
maxmoment		180	最大モーメント
minmoment		-180	最小モーメント
maxneckmoment		180	最大首モーメント
minneckmoment		-180	最小首モーメント
maxneckang		90	最大首角度
minneckang		-90	最小首角度
visible_angle		90.0	標準視野角
visible_distance			触覚的視覚可能距離
audio_cut_dist		50.0	音声の到達距離
quantize_step		0.1	移動物体のための距離の離散化ステップ
quantize_step_l		0.01	静止物体のための距離の離散化ステップ
quantize_step_dir			
quantize_step_dist_team_l			
quantize_step_dist_team_r			
quantize_step_dist_l_team_l			
quantize_step_dist_l_team_r			
quantize_step_dir_team_l			
quantize_step_dir_team_r			
ckick_margin		1.0	コーナーキックマージン
wind_dir	0.0	0.0	風の方向
wind_force	10.0	0.0	風の強さ
wind_rand	0.3	0.0	風のノイズファクター
wind_none	false		風の使用フラグ
wind_random	false		風のランダム発生
inertia_moment		5.0	turn の慣性モーメント
half_time		300	ハーフタイムの秒数
drop_ball_time		200	自動ドロップボールまでの待機サイクル数
port		6000	プレイヤのポート番号
coach_port		6001	(offline) コーチのポート番号
olcoach_port		6002	オンラインコーチのポート番号
say_coach_cnt_max		128	オンラインコーチによる freeform メッセージ使用回数の上限
say_coach_msg_size		128	オンラインコーチによる freeform メッセージ長の上限
simulator_step		100	シミュレーションステップ [単位:ミリ秒]
send_step		150	標準視覚情報ステップ [単位:ミリ秒]
recv_step		10	サーバによるコマンド受信間隔 [単位:ミリ秒]
sense_body_step		100	sense_body ステップ [単位:ミリ秒]
say_msg_size		10	say メッセージ長の上限 [単位:byte]
clang_win_size		300	CLang におけるメッセージ送信時間帯の刻み幅

Table B.1: (continued)

名前	Default 値	使用される値 in `server.conf`	説明
clang_define_win		1	時間帯あたりの最大メッセージ数
clang_meta_win		1	時間帯あたりの最大メッセージ数
clang_advice_win		1	時間帯あたりの最大メッセージ数
clang_info_win		1	時間帯あたりの最大メッセージ数
clang_mess_delay		50	CLang メッセージのプレイヤへの配送遅延
clang_mess_per_cycle		1	コーチによるメッセージの 1 サイクルあたりの最大数
hear_max		1	hear キャパシティの最大値
hear_inc		1	hear キャパシティの回復幅
hear_decay		1	hear キャパシティの減少幅
catch_ban_cycle		5	キャッチ実行の最小間隔
coach			オフラインコーチの使用フラグ
coach_w_referee			オフラインコーチとトレーナの併用フラグ
old_coach_hear			オフラインコーチの聴覚情報に 古いフォーマットを用いるフラグ
send_vi_step		100	オンラインコーチの視覚情報ステップ
use_offside		on	オフサイドルール適用フラグ
offside_active_area_size		2.5	オフサイド検出距離
forbid_kick_off_offside		on	キックオフ前の敵陣侵入許可フラグ
log_file			
record			
record_version		3	RCG ファイルのバージョン指定
record_log		on	クライアントのコマンドログフラグ
record_messages			
send_log		on	
log_times		off	サイクルに消費した時間の記録フラグ
verbose		off	verbose モードフラグ
replay			
offside_kick_margin		9.15	offside kick margin
slow_down_factor			
start_goal_l			
start_goal_r			
fullstate_l			
fullstate_r			

B.2 Playmodes

rcssserver/src/types.h で定義されているプレイモード (ビューワのためのプレイモードも含む)．

```
PM_Null,
PM_BeforeKickOff,
PM_TimeOver,
PM_PlayOn,
PM_KickOff_Left,
PM_KickOff_Right,
PM_KickIn_Left,
```

```
        PM_KickIn_Right,
        PM_FreeKick_Left,
        PM_FreeKick_Right,
        PM_CornerKick_Left,
        PM_CornerKick_Right,
        PM_GoalKick_Left,
        PM_GoalKick_Right,
        PM_AfterGoal_Left,
        PM_AfterGoal_Right,
        PM_Drop_Ball,
        PM_OffSide_Left,
        PM_OffSide_Right,

        // added for 3D viewer/commentator/small league
        PM_PK_Left,
        PM_PK_Right,
        PM_FirstHalfOver,
        PM_Pause,
        PM_Human,
        PM_Foul_Charge_Left,
        PM_Foul_Charge_Right,
        PM_Foul_Push_Left,
        PM_Foul_Push_Right,
        PM_Foul_MultipleAttacker_Left,
        PM_Foul_MultipleAttacker_Right,
        PM_Foul_BallOut_Left,
        PM_Foul_BallOut_Right,
          PM_Back_Pass_Left,
          PM_Back_Pass_Right,
          PM_Free_Kick_Fault_Left,
          PM_Free_Kick_Fault_Right,
          PM_CatchFault_Left,
          PM_CatchFault_Right,
          PM_IndFreeKick_Left,
          PM_IndFreeKick_Right,
          PM_PenaltySetup_Left,
          PM_PenaltySetup_Right,
          PM_PenaltyReady_Left,
          PM_PenaltyReady_Right,
          PM_PenaltyTaken_Left,
          PM_PenaltyTaken_Right,
          PM_PenaltyMiss_Left,
          PM_PenaltyMiss_Right,
          PM_PenaltyScore_Left,
```

```
PM_PenaltyScore_Right,
PM_MAX
```

C Soccermonitor

以下は，サッカーサーバとサッカーモニタの通信で用いられるデータ構造の解説である．

C.1 Monitor Communication Version 1

3 つの異なるタイプの情報を持つコンテナ:

```
typedef struct {
   short   mode ;
   union {
      showinfo_t show ;
      msginfo_t  msg ;
      drawinfo_t draw ;
   } body ;
} dispinfo_t ;
```

C.1.1 Showinfo

状況を描画するための情報を含む．

```
typedef struct {
   char   pmode ;
   team_t team[2] ;
   pos_t  pos[MAX_PLAYER * 2 + 1] ;
   short  time ;
} showinfo_t ;

typedef struct {
   char  name[16];  /* name of the team */
   short score;     /* current score of the team */
} team_t ;

typedef struct {
   short enable ;
   short side ;
   short unum ;
   short angle ;
   short x ;
   short y ;
```

```
} pos_t ;
```

C.1.2 Messageinfo

Messageinfo_t は，プレイヤからサーバへの全てのメッセージと審判からの全てのメッセージを含む．

```
typedef struct {
   short board ;
   char  message[2048] ;
} msginfo_t ;
```

C.1.3 Drawinfo

Drawinfo_t によって，サーバはシンプルなグラフィック要素の描画をモニタへ通知できる．

```
typedef struct {
   short mode ;
   union {
      pointinfo_t  pinfo ;
      circleinfo_t cinfo ;
      lineinfo_t   linfo ;
   } object ;
} drawinfo_t ;

typedef struct {
   short x ;
   short y ;
   char  color[COLOR_NAME_MAX] ;
} pointinfo_t ;

typedef struct {
   short x ;
   short y ;
   short r ;
   char  color[COLOR_NAME_MAX] ;
} circleinfo_t ;

typedef struct {
   short x1 ;
   short y1 ;
   short x2 ;
   short y2 ;
   char  color[COLOR_NAME_MAX] ;
```

```
} lineinfo_t ;
```

共用体が含んでいるメッセージタイプは mode によって決定される．

```
DrawClear  0
DrawPoint  1
DrawCircle 2
DrawLine   3
```

C.2 Monitor Communication Version 2

5 つの異なるタイプの情報を持つコンテナ:

```
typedef struct {
  short mode;
  union {
      showinfo_t2     show;
      msginfo_t       msg;
      player_type_t   ptinfo;
      server_params_t sparams;
      player_params_t pparams;
  } body;
} dispinfo_t2 ;
```

C.2.1 Showinfo

showinfo_t2 構造体はプレイヤとボールの状態や位置情報を含み，モニタへと毎サイクル (100ms) 送信される．

```
typedef struct {
  char     pmode ; // the play mode
  team_t   team[2] ; // team names and scores
  ball_t   ball; // ball information
  player_t pos[MAX_PLAYER * 2]; // the 22 players
  short    time ; // current simulation time
} showinfo_t2 ;
```

各要素の値は，

- pmode: プレイモード (B.2 を参照)．
- team: team_t 構造体の配列．インデックス 0 が左チーム。インデックス 1 が右チーム．
- ball: ボールの位置情報．
- pos: プレイヤの位置情報の配列．インデックス 0 から 10 が team[0]，11 から 21 が team[1] に対応．インデックスはプレイヤの背番号順．

- time: 現在のサイクル．

```
typedef struct {
  short mode;
  short type;
  long  x;
  long  y;
  long  deltax;
  long  deltay;
  long  body_angle;
  long  head_angle;
  long  view_width;
  short view_quality;
  long  stamina;
  long  effort;
  long  recovery;
  short kick_count;
  short dash_count;
  short turn_count;
  short say_count;
  short tneck_count;
  short catch_count;
  short move_count;
  short chg_view_count;
} player_t;

typedef struct {
  long x;
  long y;
  long deltax;
  long deltay;
} ball_t;

typedef struct {
   char  name[16];
   short score;
} team_t ;
```

C.2.2 Messageinfo

クライアントとサーバの通信と審判からのメッセージの情報を含む．

```
typedef struct {
   short board ;
   char  message[2048] ; /* max_message_length_for_display */
```

```
} msginfo_t ;
```

- board: メッセージのタイプを示す．MSG_BOARD (1) タイプのメッセージは審判のメッセージ (クラシックモニタの左ウインドウに表示)．LOG_BOARD (2) タイプのメッセージはプレイヤとサーバの通信メッセージ (クラシックモニタの右ウインドウに表示)．それぞれの値は，rcssserver/src/param.h に定義されている．
- message: メッセージ内容．ゼロ終端を持つ文字配列．

C.2.3 Server Parameters

```
typedef struct {
} server_params_t;
```

サーバパラメータの完全な表は付録 B.1 に書かれている．

C.2.4 Player Type

```
typedef struct {
    short id;
    long  player_speed_max;
    long  stamina_inc_max;
    long  player_decay;
    long  inertia_moment;
    long  dash_power_rate;
    long  player_size;
    long  kickable_margin;
    long  kick_rand;
    long  extra_stamina;
    long  effort_max;
    long  effort_min;

    long sparelong1;
    long sparelong2;
    long sparelong3;
    long sparelong4;
    long sparelong5;
    long sparelong6;
    long sparelong7;
    long sparelong8;
    long sparelong9;
    long sparelong10;
} player_type_t;
```

C.2.5 Player Parameters

```
typedef struct {
    short player_types;
    short subs_max;
    short pt_max;

    long  player_speed_max_delta_min;
    long  player_speed_max_delta_max;
    long  stamina_inc_max_delta_factor;

    long  player_decay_delta_min;
    long  player_decay_delta_max;
    long  inertia_moment_delta_factor;

    long  dash_power_rate_delta_min;
    long  dash_power_rate_delta_max;
    long  player_size_delta_factor;

    long  kickable_margin_delta_min;
    long  kickable_margin_delta_max;
    long  kick_rand_delta_factor;

    long  extra_stamina_delta_min;
    long  extra_stamina_delta_max;
    long  effort_max_delta_factor;
    long  effort_min_delta_factor;

    long  sparelong1;
    long  sparelong2;
    long  sparelong3;
    long  sparelong4;
    long  sparelong5;
    long  sparelong6;
    long  sparelong7;
    long  sparelong8;
    long  sparelong9;
    long  sparelong10;

    short spareshort1;
    short spareshort2;
    short spareshort3;
    short spareshort4;
    short spareshort5;
```

```
    short spareshort6;
    short spareshort7;
    short spareshort8;
    short spareshort9;
    short spareshort10;

} player_params_t;
```

Index