Chapter 1

Description of the soccer platform

The content of this chapter mostly comes from the official RoboCup Soccer Server manual[1].

1.1 Overview

1.1.1 The server

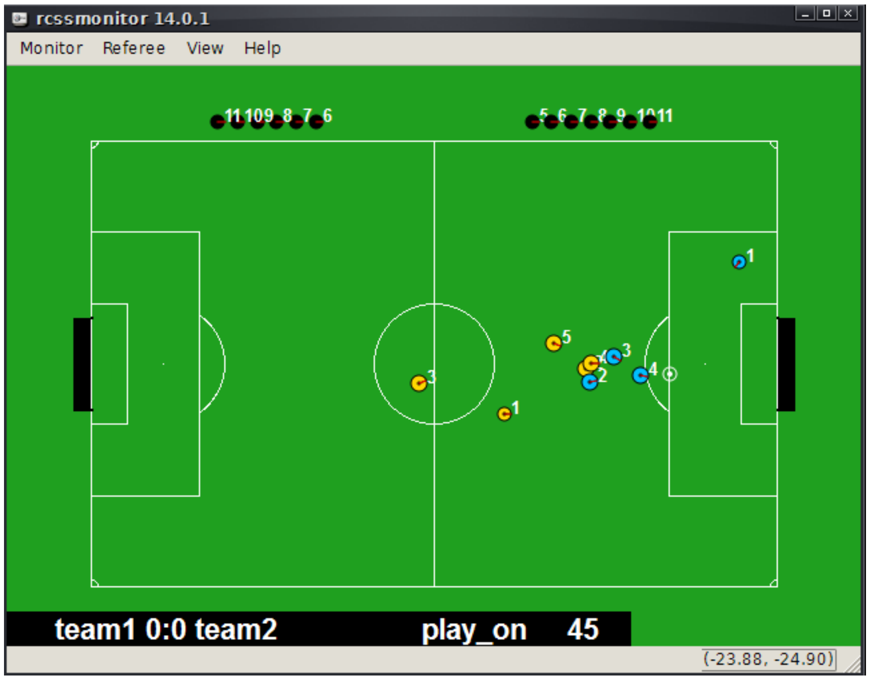
The server is a system that enables various teams to compete in a game of soccer. Since the match is carried out in a client-server style, there are no restrictions as to how teams are built. The only requirement is that the tools used to develop a team support client-server communication via UDP/IP. This is due to the fact that all communication between the server and each client is done via UDP/IP sockets. Each client is a separate process and connects to the server through a specified port. After a player has connected to the server, all messages are transferred through this port. A team can have up to 12 clients, i.e. 11 players (10 elders + 1 goalie) and a coach. The players send requests to the server regarding the actions they want to perform (e.g. kick the ball, turn, run, etc.). The server receives those messages, handles the requests, and updates the environment accordingly. In addition, the server provides all players with environment information (e.g. visual data regarding the position of objects or data about the player's ressources like stamina or speed). It is important to mention that the server is a realtime system working with discrete time intervals (or cycles). Each cycle has a specified duration, and actions that need to be executed in a given cycle must arrive at the server during the right interval.

1.1.2 The monitor

The Soccer Monitor is a visualisation tool that allows people to see what is happening within the server during a game. The information shown on the

[1] http://sourceforge.net/projects/sserver/files/rcssmanual

Figure 1.1: Monitor



monitor include the score, team names, and the positions of all the players and the ball. They also provide simple interfaces to the server. For example, when both teams have connected, the "Kick-OFF" button on the monitor allows a human referee to start the game. You can see on the figure below what it looks like.

1.1.3 Description of the simulation algorithm

In Soccer Server, time is updated in discrete steps. A simulation step is 100ms. During each simulation step, objects (i.e. players and the ball) stay on their positions. If players decide to act within a step, actions are applied to the players and the ball at the transition from one simulation cycle to the next. Depending on the play mode, not all actions are allowed for the players (for instance in 'before kick off' mode, players can turn and move, but they cannot dash), so only allowed actions will be applied and take effect. If during a step, several players kick the ball, all the kicks are applied to the ball and a resulting acceleration is calculated. If the resulting acceleration is larger than the maximum acceleration for the ball, acceleration is normalized to its maximum value. After moving the objects, the server checks for collisions and updates velocities if a collision occurred. When applying accelerations and velocities to the objects, the order of application is randomized. After changing objects positions, and updating velocities and accelerations, the automated referee checks the situation and changes the play mode or the object positions, if necessary. Changes to the play mode are announced immediately. Finally, stamina for each player is updated.

1.1.4 Protocols

When an agent is connected to the server, he must first sends an init message of the form: (**init** TeamName [(**version** VerNum)] [(**goalie**)]). In return, the server attributes the new player a side, an uniform number and indicates in which play mode the game currently is: (**init** Side Unum PlayMode).

When the player is connected, before the game goes live, he can be positioned somewhere in his side using the (**move** x y) command. Notice that $-52.5 \leq x \leq 0$. When the game is started, players can accelerate in their current orientation using (**dash** power), turn using (**turn** angle) or if they are close enough of the ball, kick it: (**kick** Power Direction). Each player can only send one of these command per cycle.

For now, we will configure the server so that every cycle, each player receives fullstate information. This means that all players are fully aware of the position and velocity of all the other mobile objects on the field. The format of the fullstate messages as well as all details concerning the protocols are given on this website:

`http://sourceforge.net/apps/mediawiki/sserver/index.php?title=Users_Manual/Soccer_Server`

1.2 Physics model

1.2.1 The field

The field is internally using a simple cartesian coordinate system. The x and y axis are centered in the middel of the field. The absolute 0° direction logically follows the x axis. Figure 1.2 illustrates these concepts.

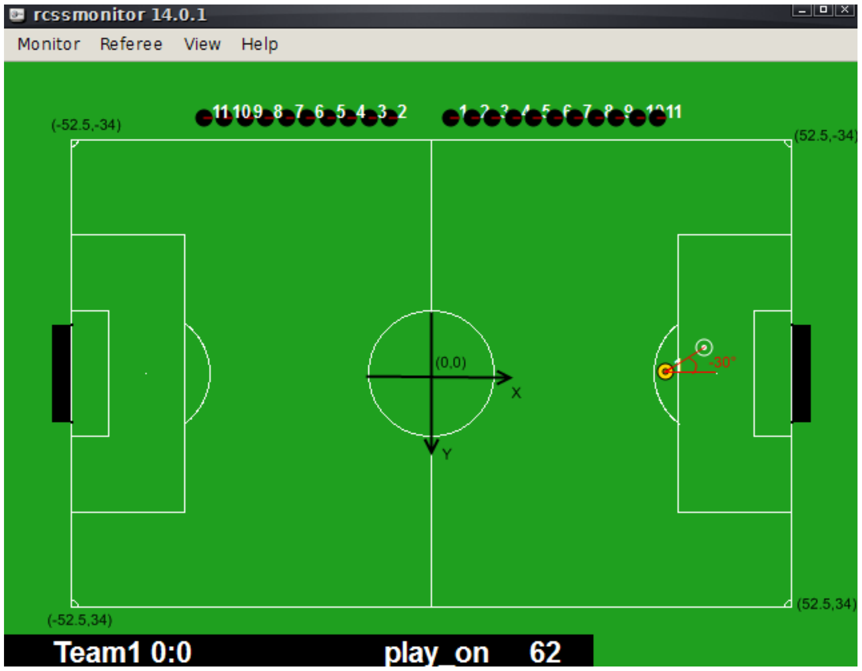
1.2.2 Movement Model

In each simulation step, movement of each object is calculated as following manner:

$$\begin{aligned}
(u_x^{t+1}, u_y^{t+1}) &= (v_x^t, v_y^t) + (a_x^t, a_y^t)\text{: accelerate} \qquad (1.1)\\
(p_x^{t+1}, p_y^{t+1}) &= (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1})\text{: move}\\
(v_x^{t+1}, v_y^{t+1}) &= \text{decay} \times (u_x^{t+1}, u_y^{t+1})\text{: decay speed}\\
(a_x^{t+1}, a_y^{t+1}) &= (0,0)\text{: reset acceleration}
\end{aligned}$$

where, (p_x^t, p_y^t), and (v_x^t, v_y^t) are respectively position and velocity of the object in timestep t. decay is a decay parameter specified by **ball_decay** or

Figure 1.2: Coordinate system



player_decay. (a_x^t, a_y^t) is acceleration of object, which is derived from *Power* parameter in **dash** (in the case the object is a player) or **kick** (in the case of a ball) commands in the following manner:

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

where θ^t is the direction of the object in timestep t and $power\_rate$ is **dash_power_rate** or is calculated from **kick_power_rate** for the ball. In the case of a player, this is just the direction the player is facing. In the case of a ball, its direction is given as the following manner:

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

where θ^t_{ball} and θ^t_{kicker} are directions of ball and kicking player respectively, and *Direction* is the second parameter of a **kick** command.

1.2.3 Collision Model

If at the end of the simulation cycle, two objects overlap, then the objects are moved back until they do not overlap. Then the velocities are multiplied by -0.1. Note that it is possible for the ball to go through a player as long as the ball and the player never overlap at the end of the cycle.

1.2.4 Dash Model

The **dash** command is used to accelerate the player in direction of its body. **dash** takes the acceleration *power* as a parameter. The valid range for the acceleration *power* can be configured in `server.conf`, the respective parameters are ***minpower*** and ***maxpower***. For the current values of parameters for the dash model, see Tab. 1.1.

Each player has a certain amount of stamina that will be consumed by **dash** commands. At the beginning of each half, the stamina of a player is set to ***stamina_max***. If a player accelerates forward (*power* > 0), stamina is reduced by *power*. Accelerating backwards (*power* < 0) is more expensive for the player: stamina is reduced by $-2 \cdot$ *power*. If the player's stamina is lower than the power needed for the **dash**, *power* is reduced so that the **dash** command does not need more stamina than available. Heterogeneous players will use some extra stamina every time the available power is lower than the needed stamina. The amount of extra stamina depends on the player type and the parameters ***extra_stamina_delta_min*** and ***extra_stamina_delta_max***.

After reducing the stamina, the server calculates the *effective dash power* for the **dash** command. The effective dash power *edp* depends on the ***dash_power_rate*** and the current effort of the player. The effort of a player is a value between ***effort_min*** and ***effort_max***; it is dependent on the stamina management of the player (see below).

$$\text{edp} = \text{effort} \cdot \text{dash\_power\_rate} \cdot \text{power} \tag{1.2}$$

edp and the players current body direction are tranformed into vector and added to the players current acceleration vector $\vec{a}_n$ (usually, that should be 0 before, since a player cannot dash more than once a cycle and a player does not get accelerated by other means than dashing).

At the transition from simulation step n to simulation step $n+1$, acceleration $\vec{a}_n$ is applied:

1. $\vec{a}_n$ is normalized to a maximum length of ***player_accel_max***.

2. $\vec{a}_n$ is added to current players speed $\vec{v}_n$. $\vec{v}_n$ will be normalized to a maximum length of ***player_speed_max***. For heterogeneous players, the maximum speed is a value between ***player_speed_max*** + ***player_speed_max_delta_min*** and ***player_speed_max*** + ***player_speed_max_delta_max*** in `player.conf`.

3. Noise $\vec{n}$ and wind $\vec{w}$ will be added to $\vec{v}_n$. Both noise and wind are configurable in `server.conf`. Parameters responsible for the wind are ***wind_force***, ***wind_dir*** and ***wind_rand***. With the current settings, there is no wind on the simulated soccer field. The responsible parameter for the noise is ***player_rand***. Both direction and length of the noise vector are within the interval $[-|\vec{v}_n| \cdot \text{player\_rand} \ldots |\vec{v}_n| \cdot \text{player\_rand}]$.

4. The new position of the player $\vec{p}_{n+1}$ is the old position $\vec{p}_n$ plus the velocity vector $\vec{v}_n$ (i.e. the maximum distance difference for the player between two simulation steps is ***player_speed_max***).

5. ***player_decay*** is applied for the velocity of the player: $\vec{v}_{n+1} = \vec{v}_n \cdot \text{player\_decay}$. Acceleration $\vec{a}_{n+1}$ is set to zero.

1.2.4.1 Stamina Model

For the stamina of a player, there are three important variables: the *stamina* value, *recovery* and *effort*. *stamina* is decreased when dashing and gets replenished slightly each cycle. *recovery* is responsible for how much the *stamina* recovers each cycle, and the *effort* says how effective dashing is (see section above). Important parameters for the stamina model are changeable in the files `server.conf` and `player.conf`. Basically, the algorithm shown in Fig. 1.3 says that every simulation step the stamina is below some threshold, effort or recovery are reduced until a minimum is reached. Every step the stamina of the player is above some threshold, *effort* is increased up to a maximum. The *recovery* value is only reset to 1.0 each half, but it will not be increased during a game.

1.2.5 Kick Model

There are no principal changes to the kick model from soccer server version 6 to soccer server version 7, so your old implementation should still work. However, due to changes in the server parameter file, in some cases multiple kicks are not necessary anymore.

The **kick** command takes two parameters, the *kick power* the player client wants to use (between ***minpower*** and ***maxpower***) and the *angle* the player kicks the ball to. The angle is given in degrees and has to be between ***minmoment*** and ***maxmoment*** (see Tab. 1.2 for current parameter values).

Once the **kick** command arrived at the server, the kick will be executed if the ball is kick-able for the player and the player is not marked offside. The ball is kick-able for the player, if the distance between the player and the ball is between 0 and ***kickable_margin***. Heterogeneous players can have different kickable margins. For the calculation of the distance during this section, it is important to know that if we talk of distance between player and ball, we talk about the minimal distance between the outer shape of both player and ball. So the distance in this section is the distance between the center of both objects **minus** the radius of the ball **minus** the radius of the player.

The first thing to be calculated for the kick is the effective kick power ep:

$$\text{ep} = \text{kick power} \cdot \text{kick\_power\_rate} \tag{1.3}$$

If the ball is not directly in front of the player, the effective kick power will be reduced by a certain amount dependent on the position of the ball with respect to the player. Both angle and distance are important.

Basic Parameters `server.conf`		**Parameters for heterogeneous Players** `player.conf`		
Name	**Value**	**Name**	**Value**	**Range**
minpower	-100			
maxpower	100			
stamina_max	4000			
stamina_inc_max	45	***stamina_inc_max_delta_factor***	-100.0	25
		player_speed_max_delta_min	0.0	— 45
		player_speed_max_delta_max	0.2	
extra_stamina[a]	0.0	***extra_stamina_delta_min***	0.0	0.0
		extra_stamina_delta_max	100.0	— 100.0
dash_power_rate	0.006	***dash_power_rate_delta_min***	0.0	0.006
		dash_power_rate_delta_max	0.002	— 0.008
effort_min	0.6	***effort_min_delta_factor***	-0.002	0.4
		extra_stamina_delta_min	0.0	— 0.6
		extra_stamina_delta_max	100.0	
effort_max[a]	1.0	***effort_max_delta_factor***	-0.002	0.8
		extra_stamina_delta_min	0.0	— 1.0
		extra_stamina_delta_max	100.0	
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	1.0			
player_speed_max	1.0	***player_speed_max_delta_min***	0.0	1.0
		player_speed_max_delta_max	0.2	— 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***	0.0	0.4
		player_decay_delta_max	0.2	— 0.6

[a]Not in `server.conf`, but compiled into the server

Table 1.1: Dash and Stamina Model Parameters for Soccer Server 7

```
{if stamina is below recovery decrement threshold, recovery is reduced}
if stamina $\leq$ recover_dec_thr $\cdot$ stamina_max then
  if recovery $>$ recover_min then
    recovery $\leftarrow$ recovery $-$ recover_dec
  end if
end if

{if stamina is below effort decrement threshold, effort is reduced}
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

{if stamina is above effort increment threshold, effort is increased}
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

{recover the stamina a bit}
stamina $\leftarrow$ stamina $+$ recovery $\cdot$ stamina_inc_max
stamina $\leftarrow$ min(stamina, stamina_max)
```

Figure 1.3: The stamina model algorithm

If the relative angle of the ball is 0° wrt. the body direction of the player client — i.e. the ball is in front of the player — the effective power will stay as it is. The larger the angle gets, the more the effective power will be reduced. The worst case is if the ball is lying behind the player (angle 180°): the effective power is reduced by 25%.

The second important variable for the effective kick power is the distance from the ball to the player: it is quite obvious that – should the kick be executed – the distance between ball and player is between 0 and ***kickable_margin***. If the distance is 0, the effective kick power will not be reduced again. The further the ball is away from the player client, the more the effective kick power will be reduced. If the ball distance is ***kickable_margin***, the effective kick power will be reduced by 25% of the original kick power.

The overall worst case for kicking the ball is if a player kicks a distant ball behind itself: 50% of *kick power* will be used. For the effective kick power, we get the formula 1.4. (dir_diff means the absolute direction difference between ball and the player's body direction, dist_diff means the absolute distance

between ball and player.)
$0 \leq \text{dir\_diff} \leq 180° \quad \wedge \quad 0 \leq \text{dist\_diff} \leq \text{kickable\_margin}$:

$$\text{ep} = \text{ep} \cdot \left(1 - 0.25 \cdot \frac{\text{dir\_diff}}{180°} - 0.25 \cdot \frac{\text{dist\_ball}}{\text{kickable\_margin}}\right) \tag{1.4}$$

The effective kick power is used to calculate $\vec{a}_{n_i}$, an acceleration vector that will be added to the global ball acceleration $\vec{a}_n$ during cycle n (remember that we have a multi agent system and *each* player close to the ball can kick it during the same cycle).

There is a server parameter, ***kick_rand***, that can be used to generate some noise to the ball acceleration. For the default players, ***kick_rand*** is 0 and no noise will be generated. For heterogeneous players, ***kick_rand*** depends on ***kick_rand_delta_factor*** in `player.conf` and on the actual kickable margin. In RoboCup 2000, ***kick_rand*** was used to generate some noise during evaluation round for the normal players.

During the transition from simulation step n to simulation step $n + 1$ acceleration $\vec{a}_n$ is applied:

1. $\vec{a}_n$ is normalized to a maximum length of ***baccel_max***. Currently (Server 7), the maximum acceleration is equal to the maximum effective kick power.

2. $\vec{a}_n$ is added to the current ball speed $\vec{v}_n$. $\vec{v}_n$ will be normalized to a maximum length of ***ball_speed_max***.

3. Noise $\vec{n}$ and wind $\vec{w}$ will be added to $\vec{v}_n$. Both noise and wind are configurable in `server.conf`. Parameters responsible for the wind are ***wind_force***, ***wind_dir*** and ***wind_rand***. The responsible parameter for the noise is ***ball_rand***. Both direction and length of the noise vector are within the interval $[-|\vec{v}_n| \cdot \text{ball\_rand} \ldots |\vec{v}_n| \cdot \text{ball\_rand}]$.

4. The new position of the ball $\vec{p}_{n+1}$ is the old position $\vec{p}_n$ plus the velocity vector $\vec{v}_n$ (i.e. the maximum distance difference for the ball between two simulation steps is ***ball_speed_max***).

5. ***ball_decay*** is applied for the velocity of the ball: $\vec{v}_{n+1} = \vec{v}_n \cdot \text{ball\_decay}$. Acceleration $\vec{a}_{n+1}$ is set to zero.

With the current settings the ball covers a distance up to 45, assuming an optimal kick. 53 cycles after an optimal kick, the distance from the kick off position to the ball is about 43, the remaining velocity is smaller than 0.1. 15 cycles after an optimal kick, the ball covers a distance of 27 – 28 and the remaining veloctity is slightly larger than 1.

Implications from the kick model and the current parameter settings are that it still might be helpful to use several small kicks for a compound kick – for example stopping the ball, kick it to a more advantageous position within the kickable area and kick it to the desired direction. It would be another possibility to accelerate the ball to maximum speed without putting it to relative position (0,0°) using a compound kick.

Basic Parameters `server.conf`		**Parameters for heterogeneous Players** `player.conf`		
Name	**Value**	**Name**	**Value**	**Range**
minpower	-100			
maxpower	100			
minmoment	-180			
maxmoment	180			
kickable_margin	0.7	***kickable_margin_delta_min***	0.0	0.7 — 0.9
		kickable_margin_delta_max	0.2	
kick_power_rate	0.027			
kick_rand	0.0	***kick_rand_delta_factor***	0.5	0.0 — 0.1
		kickable_margin_delta_min	0.0	
		kickable_margin_delta_max	0.2	
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 1.2: Ball and Kick Model Parameters

1.2.6 Move Model

The **move** command can be used to place a player directly onto a desired position on the field. **move** exists to set up the team and does not work during normal play. It is available at the beginning of each half (play mode '`before_kick_off`') and after a goal has been scored (play modes '`goal_r_`*n*' or '`goal_l_`*n*'). In these situations, players can be placed on any position in their own half (i.e. $X < 0$) and can be moved any number of times, as long as the play mode does not change. Players moved to a position on the opponent half will be set to a random position on their own side by the server.

A second purpose of the **move** command is to move the goalie within the penalty area after the goalie caught the ball. If the goalie caught the ball, it can move together with the ball within the penalty area. The goalie is allowed to move ***goalie_max_moves*** times before it kicks the ball. Additional **move** commands do not have any effect and the server will respond with `(error too_many_moves)`.

Parameter in `server.conf`	**Value**
goalie_max_moves	2

Table 1.3: Parameter for the move command

1.2.7 Turn Model

While **dash** is used to accelerate the player in direction of its body, the **turn** command is used to change the players body direction. The argument for the **turn** command is the moment; valid values for the moment are between ***minmoment*** and ***maxmoment***. If the player does not move, the moment is equal to the angle the player will turn. However, there is a concept of inertia that makes it more difficult to turn when you are moving. Specifically, the actual angle the player is turned is as follows:

$$\text{actual\_angle} = \text{moment}/(1.0 + \text{inertia\_moment} \cdot \text{player\_speed}) \qquad (1.5)$$

inertia_moment is a `server.conf` parameter with default value 5.0. Therefore (with default values), when the player is at speed 1.0, the maximum effective turn he can do is ± 30. However, notice that because you can not dash and turn during the same cycle, the fastest that a player can be going when executing a **turn** is ***player_speed_max*** $\cdot$ ***player_decay***, which means the effective turn for a default player (with default values) is ± 60.

For heterogeneous players, the inertia moment is the default ***inertia_value*** plus a value between min. ***player_decay_delta_min*** $\cdot$ ***inertia_moment_delta_factor*** and max. ***player_decay_delta_max*** $\cdot$ ***inertia_moment_delta_factor***.

Basic Parameters `server.conf`		**Parameters for heterogeneous Players** `player.conf`		
Name	**Value**	**Name**	**Value**	**Range**
minmoment	-180			
maxmoment	180			
inertia_moment	5.0	***player_decay_delta_min***	0.0	5.0
		player_decay_delta_max	0.2	10.0
		inertia_moment_delta_factor	25.0	

Table 1.4: Turn Model Parameters

1.2.8 Play Modes and referee messages

The change of the play mode is announced by the referee. Additionally, there are some referee messages announcing events like a goal or a foul. If you have a look into the server source code, you will notice some additional play modes that are currently not used. Both play modes and referee messages are announced using `(referee` *`String`*`)`, where *String* is the respective play mode or message string. The play modes are listed in Tab. 1.5, for the messages see Tab. 1.6.

Play Mode	t_c	subsequent play mode	comment
‘`before_kick_off`’	0	‘`kick_off_`*`Side`*’	at the beginning of a half
‘`play_on`’			during normal play
‘`time_over`’			
‘`kick_off_`*`Side`*’			announce start of play (after pressing the `Kick Off` button)
‘`kick_in_`*`Side`*’			
‘`free_kick_`*`Side`*’			
‘`corner_kick_`*`Side`*’			
‘`goal_kick_`*`Side`*’		‘`play_on`’	play mode changes once the ball leaves the penalty area
‘`goal_`*`Side`*’			currently unused (but see Tab. 1.6).
‘`drop_ball`’	0	‘`play_on`’	
‘`offside_`*`Side`*’	30	‘`free_kick_`*`Side`*’	for the opposite side

where *Side* is either the character ‘`l`’ or ‘`r`’, *OSide* means opponent’s side.
t_c is the time (in number of cycles) until the subsequent play mode will be announced

Table 1.5: Play Modes

Message	t_c	subsequent play mode	comment
goal_*Side*_*n*	50	‘`kick_off_`*`OSide`*’	announce the *n*th goal for a team
foul_*Side*	0	‘`free_kick_`*`OSide`*’	announce a foul
goalie_catch_ball_*Side*	0	‘`free_kick_`*`OSide`*’	
time_up_without_a_team	0	‘`time_over`’	sent if there was no opponent until the end of the second half
time_up	0	‘`time_over`’	sent once the game is over (if the time is $\geq$ second half **and** the scores for each team are different)
half_time	0	‘`before_kick_off`’	
time_extended	0	‘`before_kick_off`’	

where *Side* is either the character ‘`l`’ or ‘`r`’, *OSide* means opponent’s side.
t_c is the time (in number of cycles) until the subsequent play mode will be announced

Table 1.6: Referee Messages