TU Clausthal

Clausthal University of Technology

Technical Foundations of the Agent Contest 2008

Tristan M. Behrens, Jürgen Dix, Mehdi Dastani, Michael Köster, Peter Novák

IfI Technical Report Series IfI-08-05

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Technical Foundations of the Agent Contest 2008

Tristan M. Behrens, Jürgen Dix, Mehdi Dastani, Michael Köster, Peter Novák

Abstract

In this document we summarize the technical infrastructure of the Agent Contest 2008, which is the fourth competition in an ongoing series that was initiated in 2005. This document is also a collection of the descriptions that we have issued during the contest.

1 Introduction

The Agent Contest is an international competition that has been created in 2005 in order to allow the comparison of agent-based approaches to systems programming. Benchmarks are provided by letting agent teams solve a cooperative task in a dynamically changing environment, while they have to compete against other teams.

The first Agent Contest was held in 2005 in association with the CLIMA workshop. The scenario was a grid-like world, in which agents had to gather and store resources facing incomplete-information. The first contest was decentralized: the participants had to implement the agents as well as the environment and did not compete with other teams. In 2006 this has been changed by introducing the MASSim platform, which has been the fundament of the contest since then. We kept the scenario and let agent-teams compete for gold. In 2007 we moved to the ProMAS workshop and kept the scenario. In 2008 we have changed the scenario to the cows and cowboys scenario that was designed in order to put stress on the cooperation and coordination aspects of agent programming.

In this technical report we will introduce the MASSim platform in the first section. In the section section we will explain in detail the scenario of the Agent Contest 2008. The third section will consider the communication protocol and the fourth one will explain how to create new scenarios. We will finish with conclusions and future work.

2 MASSim Server

The MASSim (Multi-Agent Systems Simulation) platform is a testing environment that has been designed in order to evaluate coordination and coopera-

tion approaches of multi-agent systems. To that end we employ round-based game simulations with the intention to evaluate agent-based approaches by letting agent teams compete.

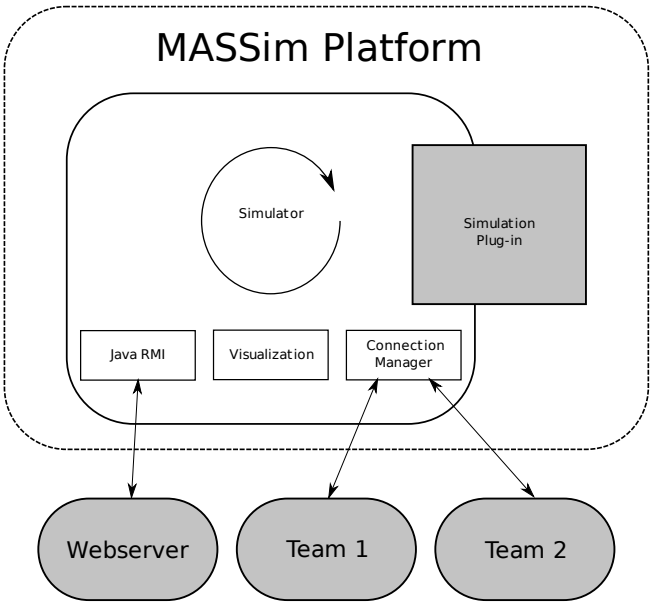


Figure 1: MASSim Plattform overview.

Figure 1 shows the technical infrastructure of MASSim. The platform consists of these components:

- **Core:** is the central component that coordinates the interaction of the other components and implements the tournament schedule.
- **Simulation plug-in:** describes a discrete game and logically contains the environment. This component is based on a plug-in architecture that allows the implementation and use of new scenarios in an elegant way.
- **Agent-server communication:** manages the communication between the server and the agents. The communication relies on the exchange of XML-messages. The agents receive perceptions and can act in the environment by exchanging XML-messages with the server (see the protocol section for details). Not that we do not provide functionality for inter-agent communication.
- **Agent-teams:** this is the only external component. Agent teams connect to the server via TCP/IP, and communicate using XML-messages.

The participants of the contest are free to use their own approaches. The only thing that they have to tackle is the implementation if the communication protocol. Furthermore we usually provide a dummy agent that already provides that functionality and that can be use as a starting point for a more sophisticated implementation.

- **Visualization:** this component renders each state of the evolution of the environment to a SVG file. The SVG files can then be viewed in a manner that resembles videos.
- **Web interface:** provides online-monitoring functionality. People can use the web interface to monitor the progress of a tournament, including the current tournament results and the ongoing matches and simulations.
- **Debug monitor:** is provided for debugging purposes.

3 Scenario

Background Story: An unknown species of cattle was recently discovered in the unexplored flatlands of Lemuria. The cows have some nice features: their carbondioxyde- and methane-output is extremely low compared to the usual cattle and their beef and milk are of supreme quality and taste.

These facts definitely caught the attention of the beef- and milk-industries. The government decided to allow the cows to be captured and bred by everyone who is interested and has the capabilities. Several well-known companies decided to send in their personnel to the fields to catch as many of them as possible. This led to an unprecedented rush for cows. To maximise their success the companies replaced their traditional cowboys by *artificial herders*.

In this year's agent contest the participants have to compete in an environment for cows. Each team controls a set of herders in order to direct the cows into their own corral. The team with the most cows in the corral at the end wins the match.

3.1 General Description and Tournament Structure

Before the tournament, agent teams will be randomly divided into several groups if necessary. In the case of a small number of participating teams, these will form one single group.

Each team from one group will compete against all other teams in the same group in a series of matches. The winners from these groups form a new group. Each team in a new group will again play against all other teams in the group in a series of matches. A single match between two competing teams will consist of several (odd number of) simulations. A simulation between two teams is a competition between them with respect to a certain configuration of the environment.

Winning a simulation yields 3 points for the team, a draw is worth 1 point and a loss 0 points. The winner of the whole tournament is evaluated on the basis of the overall number of collected points in all the matches during the tournament. In the case of equal number of points, the winner will be decided on the basis of the absolute number of captured cows.

Details on the number of simulations per match and the exact structure of the competition will depend on the number of participating teams and will be specified later.

In the contest, the agents from each participating team will be executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, is run on the remote contest simulation server run by the contest organizers.

The interaction/communication between agents from one team should be managed locally, but the interaction between individual agents and their environment (run on the simulation server) will be via Internet. Participating agents connect to the simulation server that provides the information about the environment.

Each agent from each team should connect and communicate to the simulation server using one TCP connection. After the initial phase, during which agents from all competing teams connect to the simulation server, identify and authenticate themselves and get a general match information, the competition will start. The simulation server controls the competition by selecting the competing teams and managing the matches and simulations. In each simulation, the simulation server, in a cyclic fashion, provides sensory information about the environment to the participating agents and expects their reactions within a given time limit.

Each agent reacts to the received sensory information by indicating which action (including the $skip$ action) it wants to perform in the environment. If no reaction is received from the agent within the given time limit, the simulation server assumes that the agent performs the $skip$ action. Agents have only a *local view* of their environment, their *perceptions can be incomplete*, and their *actions may fail*. That means that agents can receive incomplete information about the environment from the simulation server. The simulation server can omit information about particular environment cells, however, the server never provides incorrect information. Also, agent's action can fail. In such a case the simulation server evaluates the agent's action in the simulation step as the skip action.

After a finite number of steps the simulation server stops the cycle and participating agents receive a notification about the end of a simulation. Then the server starts a new simulation possibly involving the same teams.

3.1.1 Preparation Stage and Communication Protocol

Several days before the start of the competition, the contest organisers will contact participants via e-mail with details of time and Internet coordinates (IP addresses/ports) of the simulation server. Participants will also receive agent IDs and passwords necessary for authentication of their agents for the tournament. Agents communicate with the simulation server using TCP protocol and by means of messages in XML format. The details about communication protocol and message format will be specified later.

3.1.2 Initial Phase

At the announced start time of the tournament, the simulation server will go on-line, so that agents from participating teams will be able to connect. After a successful initial handshake during which agents will identify themselves by their IDs and receiving acknowledgment from the server, they should wait for the simulation start. The initial connecting phase will take a reasonable amount of time in order to allow agents to be initialised and connected and will not be less than 5 minutes. The details will be announced later.

3.1.3 Team, Match, and Simulation

An agent team consists of 6 software agents with distinct IDs. There are no restrictions on the implementation of agents, although we encourage the use of approaches based on the state-of-the-art tools, methodologies and languages for programming agents and multi-agent systems as well as the use of computational logic based approaches.

The tournament consists of a number of matches. A match is a sequel of simulations during which two teams of agents compete in several different settings of the environment.

For each match, the server will 1) pick two teams to play it and subsequently 2) start the first simulation of the match. Each simulation in a match starts by notifying the agents from the participating teams and sending them the details of the simulation. These will include for example the size of the grid, corral position, the number of steps the simulation will perform, etc.

A simulation consists of a number of simulation steps. Each step consists of 1) sending a sensory information to agents (one or more) and 2) waiting for their actions and 3) processing agents' replies and calculating the next state of the environment. In the case that an agent does not respond within

a timeout (specified at the beginning of the simulation) by a valid action, it is considered to perform the $skip$ action in the given simulation step.

The (simulated) environment is a rectangular grid consisting of cells. The size of the grid is specified at the start of each simulation and is variable. However, it cannot be more than 150×150 cells. The $[0, 0]$ coordinate of the grid is in the top-left corner (north-west). The simulated environment contains two corrals—one for each team—which serve as a location where cows should be directed to. The environment can contain the following objects in its cells:

- obstacle (a cell with an obstacle cannot be visited by an agent),
- cow,
- agent,
- corral (a cell to which cows are to be directed in order to earn points in a simulation).

There can be only one object in a cell, except that an agent or a cow can enter cells containing corral. At the beginning of a simulation the grid contains obstacles, cows and agents of both teams. Distribution of obstacles, cows and initial positions of agents can be either hand crafted for the particular scenario, or completely random. At the start of each simulation agents will get the details of the environment (grid size, corral position, etc.). Agents will get information about their initial position in the perception information of the first simulation step.

3.1.4 Perception

Agents are located in the grid and the simulation server provides each agent with the following information:

- absolute position of the agent in the grid,
- the content of the cells surrounding the agent and the content of the cell in which the agent currently stands in. Each agent has a viewing range that describes a square of visible cells around the agent with the width which is an odd number and the agent in the center of that square,
- agents will perceive identifiers of cows in every perception,
- agents will perceive the position of their own corral but not the position of the corral of the other team.

If two agents are standing in each other's field of view, they will be able to recognise whether they are enemies, or whether they belong to the same team. The corral position will be made available at the beginning of each match. Note that all perceptions except for the agent's and the corral's position can be omitted by the server, whereas the server never gives wrong informations.

3.1.5 Actions

Agents are allowed to perform one action in a simulation step. The following actions are allowed:

- $skip$ – the agent does nothing,
- $north$ – the agent moves to the north,
- $northeast$ – the agent moves to the northeast,
- $east$ – the agent moves to the east,
- $southeast$ – the agent moves to the southeast,
- $south$ – the agent moves to the south,
- $southwest$ – the agent moves to the southwest,
- $west$ – the agent moves to the west,
- $northwest$ – the agent moves to the northwest.

Note that the $[0, 0]$ coordinate of the grid is in the top-left corner (northwest).

All actions, except the $skip$ action, can fail. The result of a failed action is the same as the result of the $skip$ action. An action can fail either because the conditions for its successful execution are not fulfilled or because of the information distortion.

The agent can do nothing or move into one of the eight directions of the compass. The execution of the $skip$ action has no influence on the local state of the environment around the agent (under the assumption that other agents did not change it). When an agent does not respond to a perception information provided by the simulation server within the given time limit, the agent is considered as performing the $skip$ action. The execution of the move actions changes the position of the agent one cell to the respective direction. A movement action succeeds only when the cell to which an agent is about to move does not contain an obstacle and is not outside of the grid. In the case two agents stand in the adjacent cells and one of them tries to step into the cell the second agent stands in while the second agent performs

e.g. a $skip$ action, the movement action fails. In the case two agents attempt to enter the same cell, only one of the two movement actions succeeds. Exactly which agent succeeds in entering the cell is determined randomly.

To make matters clearer:

- it is impossible for two agents/cows to swap places,
- a cow/agent can only enter a cell in the next step of the simulation if the cell is empty in the current step,
- agents can enter their own corral cells and the ones of the other team,
- if two or more agent/cows intend to enter the same empty cell it is left to chance which agent/cow succeeds.

3.1.6 Cow Movement Algorithm

Cows are simple creatures. They tend to move away from cells that they do not like and to move towards cells they do like. Cows want to move away from agents and trees. On the other hand, they are attracted by empty spaces and they want to stay close to other cows, however not too close. Cows have the tendency to form herds, which tend to be tighter in times when the animals are scared by cowboys.

The cows have two fixed visibility ranges. The number r_c represents the width of the visibility-square with the cow in its center. The number r_{cN} represents the width of the intimacy-square with the cow in the centre. Cows are attracted to other cows that are in the visibility-square and not in the intimacy-square and they are repelled by cows that are in their intimacy-square.

Cows are slower than agents. Each cow only moves every three steps. Our simulation ensures that all cows do not move in the same step using this simple algorithm: At the beginning of the simulation each cow is given a random number $n_{cowID} \in \mathbb{N}$. Let $s \in \mathbb{N}$ be the current step of the simulation. The cow moves if the equation $s \bmod 3 \equiv n_{cowID} \bmod 3$ holds.

If it is time for a cow to move the following happens: For each cow me the set $Cells$ is the contents of all the cells in the visibility range (which is a square with the agent in the centre) of the cow me. The content of a single cell can be either cow, $agent$, $tree$ or $empty$.

Let $w : C \to \mathbb{Z}$ be a weight-function, that maps each cell to an integer according to its contents. A negative number indicates fear of the content and a positive one indicates attraction. The vectors $\vec{p}_c$ and $\vec{p}_{me}$ are the coordinate-vectors of the cell c and the cow me respectively.

Now the position of a cow in the next simulation step is determined as follows:

1. Calculate the weighted linear combination

$$\vec{v} := \sum_{c \in Cells} w(c) \cdot \frac{(\vec{p}_c - \vec{p}_{me})}{|\ \vec{p}_c - \vec{p}_{me}\ |}$$

2. Move me according to the angle of $\vec{v}$.

Cows do not move if the resulting vector is zero. We use a number precision of two digits after decimal point without rounding.

Figure 2 shows how the directions of the compass correspond to angles: if the angle is in the range $[-22.5, 22.5)$ a cow moves east, if the angle is in the range $[22.5, 67.5)$ the cow moves to the northeast et cetera.

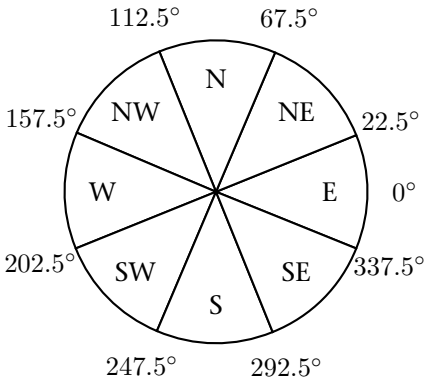


Figure 2: Angles and directions.

There are several constraints for the weights:

- $w(empty) > 0$, cows are attracted by empty spaces,
- $w(cow) > 0$, cows are attracted by other cows if these are not too close,
- $w(cow_{private}) < 0$, cows are attracted by other cows if these are too close,
- $w(agent) < 0$, cows are scared of agents,
- $w(tree) < 0$, cows are scared of trees,
- $w(tree) = -w(empty)$, cows are scared of trees as much as they are attracted by empty spaces,
- $|\ w(cow)\ | < |\ w(agent)\ |$, cows are less attracted by other cows than they are scared of agents, and

- $w(corral) = w(empty)$, cows do not distinguish between corral-cells and empty spaces.

The weights will be announced soon on the website and on our mailing list.

Fig. 3 shows the contents of the cells that are in the visibility range around the cow me. In this example cows have the visibility range 3. Note that we go for bigger values in our simulation and that we just use 3 in this example for the sake of simplicity. The cell in the northwest of me contains a $tree$, the one in the northeast contains an $agent$ and all the other cells are $empty$.

Let the weights be as follows (they will be slightly different in our simulation):

$$\begin{aligned} w(empty) &= 1 \\ w(tree) &= -1 \\ w(agent) &= -2 \end{aligned}$$

The weighted sum is as follows:

$$\begin{aligned} \vec{v} \;:=\; & \left(-\frac{1}{\sqrt{2}}\right) \cdot \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \left(-\frac{2}{\sqrt{2}}\right) \cdot \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \\ & 1 \cdot \left(\begin{bmatrix} 0 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) \\ =\; & \left(\frac{1}{\sqrt{2}}\right) \begin{bmatrix} -1 \\ 5 \end{bmatrix} \end{aligned}$$

Determine the direction and move the cow: The angle of $\vec{v}$ is $258°$, thus the cow moves south.

3.1.7 Final Phase

In the final phase, the simulation server sends a message to each agent allowing them to disconnect from the server. By this, the tournament is over.

3.2 Relation to the Previous Contest Editions

The first two editions were organized in cooperation with the CLIMA workshop series, the third one was the first one in cooperation with the ProMAS workshop series. This scenario is not an extension of the last contests, which were hunts for gold. Instead it is completely new scenario.

The main differences to the gold-mining scenario are:

- the agents have a wider viewing range,
- the agents have only moving actions, now they can move into eight directions instead of four, the pushing action has been abandoned,

tree	*empty*	*agent*
empty	*me*	*empty*
empty	*empty*	*empty*

Figure 3: Example for the cow-algorithm.

- cows were introduced that flock and disperse, and
- the goal is to direct as many cows as possible into one of the corrals.

We believe that the new scenario constitutes a new and very interesting challenge.

3.3 Network Miscellanea

Our simulation server does not provide a facility for inter-agent communication. Agents from a team are allowed to communicate and coordinate their actions locally. Based on the number of participants, organisers will decide whether to run the competition in just one or more rounds.

The continuous connection of agents from the first match to the last one cannot be guaranteed. In the case of agent-to-server connection disruption, agents are allowed to reconnect by connecting and performing the initial tournament phase message exchange again.

Generally, participants are responsible for maintaining connections of their agents to the simulation server. In the case of connection disruption during the running simulation, server will proceed with the tournament simulation, however the action of a disconnected agent will be considered as the *skip* action. In the case of a serious connection disruption, organizers reserve the right to consider each case separately.

The agents should inform the simulation server which action they want to perform within a timeout specified at the beginning of the simulation. The contest organisers do not take any responsibility for the speed of the Internet

connection between the server and participating agents. Timeouts will be set reasonable high, so that even participants with a slow network connection will be able to communicate with the server in an efficient way. Simulation timeouts will not be lower than 2 and higher than 10 seconds per one simulation step.

A ping interface will be provided by the server in order to allow participating agents to test the speed of their connection during the initial phase of the tournament. Note, that only a limited number of ping requests will be processed from one agent in a certain time interval. Details on this limit will be announced later through our mailing list and posted on our website.

3.4 Technical Support and Organisational Issues

We are running a mailing list for all the inquiries regarding Multi-Agent Programming Contest 2008. Feel free to subscribe if you are interested in further details on the contest. Subscription for participants is mandatory. The list address:
`agentcontest2008-general@in.tu-clausthal.de`
To subscribe, please send an e-mail to
`agentcontest2008-general-subscribe@in.tu-clausthal.de`
The most recent information about the Multi-Agent Programming Contest 2008 can be found on the official web site
`http://cig.in.tu-clausthal.de/agentcontest2008/`

4 Protocol

An unknown species of cattle was recently discovered in the unexplored flatlands of Lemuria. The cows have some nice features: their carbondioxyde- and methane-output is extremely low compared to the usual cattle and their beef and milk are of supreme quality and taste.

These facts definitely caught the attention of the beef- and milk-industries. The government decided to allow the cows to be captured and bred by everyone who is interested and has the capabilities. Several well-known companies decided to send in their personnel to the fields to catch as many of them as possible. This led to an unprecedented rush for cows. To maximise their success the companies replaced their traditional cowboys by *artificial herders*.

In this year's agent contest the participants have to compete in an environment for cows. Each team controls a set of herders in order to direct the cows into their own corral. The team with the most cows in the corral at the end wins the match.

4.1 General Agent-2-Server Communication Principles

In this contest, the agents from each participating team will be executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, is run on the remote contest simulation server.

Agents communicate with the contest server using standard TCP/IP stack with socket session interface. The Internet coordinates (IP address and port) of the contest server (and a dedicated test server) will be announced later via the official Contest mailing list.

Agents communicate with the server by exchanging XML messages. Messages are well-formed XML documents, described later in this document. We recommend using standard XML parsers available for many programming languages for generation and processing of these XML messages.

4.1.1 Communication Protocol

Logically, the tournament consists of a number of matches. A match is a sequel of simulations during which two teams of agents compete in several different settings of the environment. However, from agent's point of view, *the tournament consists of a number of simulations in different environment settings and against different opponents*.

The tournament is divided into three phases. During the initial phase, agents connect to the simulation server and identify themselves by username and password (the `AUTH-REQUEST` message). Credentials for each agent will be distributed in advance via e-mail. As a response, agents receive the result of their authentication request (`AUTH-RESPONSE` message) which can either succeed, or fail. After successful authentication, agents should wait until the first simulation of the tournament starts.

At the beginning of each simulation, agents of the two participating teams are notified (`SIM-START` message) and receive simulation specific information:

- simulation ID,
- opponent's ID,
- grid size,
- corral position and size, and
- number of steps the simulation will last.

In each simulation step each agent receives a perception about its environment (the `REQUEST-ACTION` message) and should respond by performing an action (`ACTION` message). Each `REQUEST-ACTION` message contains

- information about the cells in the visibility range of the agent (including the one agent stands on),
- the agent's absolute position in the grid,
- the current simulation step number,
- the number of caught cows and
- the deadline for responding.

The agent has to deliver its response within the given deadline. The action message has to contain the identifier of the action, the agent wants to perform, and action parameters, if required.

When the simulation is finished, participating agents receive a notification about its end (`SIM-END` message) which includes

- the information about the number of caught cows, and
- the information about the result of the simulation (whether the team has won or lost the simulation).

All agents which currently do not participate in a simulation should wait until the simulation server notifies them about either 1) the start of a simulation, they are going to participate in, or 2) the end of the tournament.

At the end of the tournament, all agents receive a notification (`BYE` message). Subsequently the simulation server will terminate the connections to the agents.

4.1.2 Reconnection

When an agent loses connection to the simulation server, the tournament proceeds without disruption, only all the actions of the disconnected agent are considered to be empty ($skip$). Agents are themselves responsible for maintaining the connection to the simulation server and in a case of connection disruption, they are allowed to reconnect.

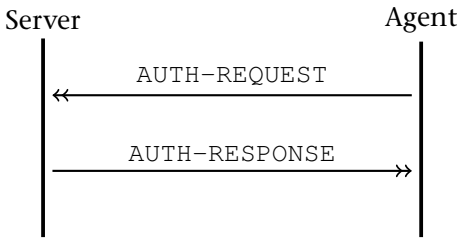
Agent reconnects by performing the same sequence of steps as at the beginning of the tournament. After establishing the connection to the simulation server, it sends `AUTH-REQUEST` message and receives `AUTH-RESPONSE`. After successful authentication, server sends `SIM-START` message to an agent. If an agent participates in a currently running simulation, the `SIM-START` message will be delivered immediately after `AUTH-RESPONSE`. Otherwise an agent will wait until a next simulation in which it participates starts. In the next subsequent step when the agent is picked to perform an action, it receives the standard `REQUEST-ACTION` message containing the perception of the agent at the current simulation step and simulation proceeds in a normal mode.

4.1.3 Ping Interface

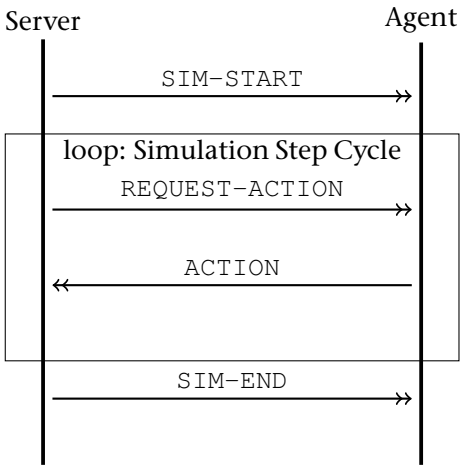
The simulation server provides a ping interface in order to allow agents to test their connection to the simulation server. Agent can send a `PING` message containing a payload data (ASCII string up to 100 characters) and it will receive `PONG` message with the same payload. As all messages contain a timestamp (see description of the message envelope below), agent can also use ping interface to synchronize its time with the server.

4.2 Protocol Sequence Diagram (UML like notation)

- Initial phase

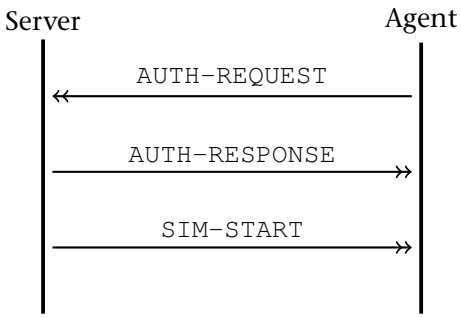


- Simulation

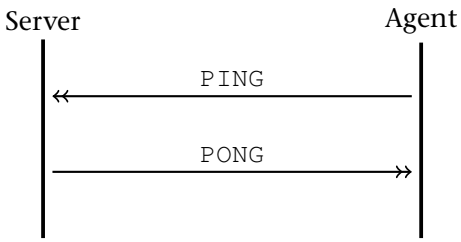


- Final phase

- Reconnect



- Ping



4.3 XML Messages Description

4.3.1 XML message structure

XML messages exchanged between server and agents are zero terminated UTF-8 strings. Each XML message exchanged between the simulation server and agent consists of three parts:

- Standard XML header: Contains the standard XML document header

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

- Message envelope: The root element of all XML messages is `<message>`. It has attributes the timestamp and a message type identifier.
- Message separator: Note that because each message is a UTF-8 zero terminated string, messages are separated by nullbyte.

The timestamp is a numeric string containing the status of the simulation server's global timer at the time of message creation. The unit of the global

timer is milliseconds and it is the result of standard system call "time" on the simulation server (measuring number of milliseconds from January 1st, 1970 UTC). Message type identifier is one of the following values: `auth-request`, `auth-response`, `sim-start`, `sim-end`, `bye`, `request-action`, `action`, `ping`, `pong`.

Messages sent from the server to an agent contain all attributes of the root element. However, the timestamp attribute can be omitted in messages sent from an agent to the server. In the case it is included, server silently ignores it.

Example of a server-2-agent message:

```
<message timestamp="1138900997331" type="request-action">
  <!-- optional data -->
</message>
```

Example of an agent-2-server message:

```
<message type="auth-request">
  <!-- optional data -->
</message>
```

Optional simulation specific data According to the message type, the root element `<message>` can contain simulation specific data.

4.3.2 `AUTH-REQUEST` (agent-2-server)

When the agent connects to the server, it has to authenticate itself using the username and password provided by the contest organizers in advance. This way we prevent the unauthorized use of connection belonging to a contest participant. `AUTH-REQUEST` is the very first message an agent sends to the contest server.

The message envelope contains one element `<authentication>` without subelements. It has two attributes `username` and `password` of type string.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="auth-request">
  <authentication username="xteam5" password="jabjar5"/>
</message>
```

4.3.3 `AUTH-RESPONSE` (server-2-agent)

Upon receiving `AUTH-REQUEST` message, the server verifies the provided credentials and responds by a message `AUTH-RESPONSE` indicating success, or failure of authentication. It has one attribute `timestamp` that represents the time when the message was sent.

The envelope contains one `<authentication>` element without subelements. It has one attribute `result` of type string and its value can be either `"ok"`, or `"fail"`. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204979083919" type="auth-response">
  <authentication result="ok"/>
</message>
```

4.3.4 `SIM-START` (server-2-agent)

At the beginning of each simulation the server picks two teams of agents to participate in the simulation. The simulation starts by notifying the corresponding agents about the details of the starting simulation. This notification is done by sending the `SIM-START` message.

The data about the starting simulation are contained in one `<simulation>` element with the following attributes:

- `id` - unique identifier of the simulation (string),
- `opponent` - unique identifier of the enemy team (string),
- `steps` - number of steps the simulation will perform (numeric),
- `gsizex` - horizontal size of the grid environment (west-east) (numeric),
- `gsizey` - vertical size of the environment (north-south) (numeric),
- `corralx0` - left border of the corral (numeric),
- `corralx1` - right border of the corral (numeric),
- `corraly0` - upper border of the corral (numeric),
- `corraly1` - lower border of the corral (numeric).

Remark: One step involves all agents acting at once. Therefore if a simulation has n steps, it means that each agent will receive `REQUEST-ACTION` messages exactly n times during the simulation (unless it loses the connection to the simulation server).

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204979126544" type="sim-start">
  <simulation
    corralx0="0"
    corralx1="14"
    corraly0="55"
```

```
    corraly1="69"
    gsizex="70" gsizey="70"
    id="stampede"
    opponent="xteam"
    steps="10"/>
</message>
```

4.3.5 SIM-END (server-2-agent)

Each simulation lasts a certain number of steps. At the end of each simulation the server notifies agents about its end and its result.

The message envelope contains one element `<sim-result>` with two attributes `score` and `result`. `score` attribute contains number of caught in the corral of the team the given agent belongs to, and `result` is a string value equal to one of `"win","lose","draw"`. The element `<sim-result>` does not contain subelements.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978760356" type="sim-end">
  <sim-result result="draw" score="9"/>
</message>
```

4.3.6 BYE (server-2-agent)

At the end of the tournament the server notifies each agent that the last simulation has finished and subsequently terminates the connections. There is no data within the message envelope of this message.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978760555" type="bye"/>
```

4.3.7 REQUEST-ACTION (server-2-agent)

In each simulation step the server asks the agents to perform an action and sends them the corresponding perceptions.

The message envelope of the `REQUEST-ACTION` message contains the element `<perception>` with six attributes:

- `step` - current simulation step (numeric),
- `posx` - column of current agent's position (numeric),
- `posy` - row of current agent's position (numeric),

- `score` - number of cows caught in the corral of the agent's team until the current simulation step (numeric),
- `deadline` - server global timer value until which the agent has to deliver a reaction in form of an `ACTION` message (the same format as timestamp),
- `id` - unique identifier of the `REQUEST-ACTION` message (string).

The element `<perception>` contains a number of subelements `<cell>` with two numeric attributes `x` and `y` that denote the cell's relative position to the agent.

If an agent stands near the grid border, or corner, only the perceptions for the existing cells are provided.

Each element `<cell>` contains a number of subelements indicating the content of the given cell. Each subelement is an empty element tag without further subelements:

- `<agent>` - there is an agent in the cell. The string attribute `type` indicates whether it is an agent of the enemy team, or ally. Allowed values for the attribute `type` are `"ally"` and `"enemy"`.
- `<obstacle>` - the cell contains an obstacle. This element has no associated attributes.
- `<cow>` - the cell contains a cow. The string attribute `ID` represents the cow's unique identifier.
- `<corral>` - the cell is a corral cell. The string attribute `type` indicates whether it belongs to the team's or the opponent's corral. Allowed values for the attribute `type` are `"ally"` and `"enemy"`.
- `<empty>` - the cell is empty.
- `<unknown>` - the content of a cell is not provided by the server because of information distortion.

The specific rules on allowed combinations of objects in a cell are provided in the scenario description.

Example (compare to Fig. 4):

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1205147104629" type="request-action">
  <perception
    step="0"
    posx="13"
    posy="35"
```

```
    score="0"
    deadline="1205147112629"
    id="1">
    <cell x="-8" y="-8"><empty/></cell>
    ...
    <cell x="-8" y="0"><agent Type="ally"/></cell>
    ...
    <cell x="-8" y="5"><corral type="ally"/></cell>
    <cell x="-8" y="6"><corral type="ally"/></cell>
    <cell x="-8" y="7"><corral type="ally"/></cell>
    <cell x="-8" y="8"><corral type="ally"/></cell>
    ...
    <cell x="-7" y="5"><corral type="ally"/></cell>
    <cell x="-7" y="6"><corral type="ally"/></cell>
    <cell x="-7" y="7"><corral type="ally"/></cell>
    <cell x="-7" y="8"><corral type="ally"/></cell>
    ...
    <cell x="-6" y="0"><agent type="ally"/></cell>
    ...
    <cell x="-6" y="5"><corral type="ally"/></cell>
    <cell x="-6" y="6"><corral type="ally"/></cell>
    <cell x="-6" y="7"><corral type="ally"/></cell>
    <cell x="-6" y="8"><corral type="ally"/></cell>
    ...
    <cell x="-5" y="2"><cow ID="47"/></cell>
    <cell x="-5" y="3"><unknown/></cell>
    <cell x="-5" y="4"><cow ID="52"/></cell>
    <cell x="-5" y="5"><corral type="ally"/></cell>
    <cell x="-5" y="6"><corral type="ally"/></cell>
    <cell x="-5" y="7"><corral type="ally"/></cell>
    <cell x="-5" y="8"><corral type="ally"/></cell>
    ...
  </perception>
</message>
```

Note that the agent perceives an area that is a square with the size 17 with the agent in the center. Thus each agent is able to see 289 cells. We refrained from depicting all 289 cells in the above example and showed just some of the relevant cells instead. The three dots indicate the missing `<cell>` elements.

4.3.8 `ACTION` (agent-2-server)

The agent should respond to the `REQUEST-ACTION` message by an action it chooses to perform.

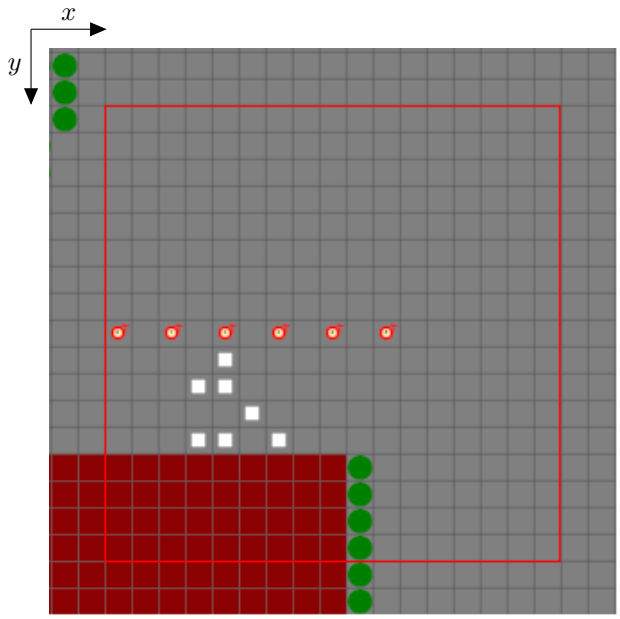


Figure 4: The view range of the agents. The agent is in the center and perceives all the cells in the red rectangle. Cows are white squares, trees are green circles.

The envelope of the `ACTION` message contains one element `<action>` with the attributes `type` and `id`. The attribute `type` indicates an action the agent wants to perform. It contains a string value which can be one of the following strings:

- `"skip"` – (the agent does nothing),
- `"north"` – (the agent moves one cell to the top) ,
- `"northeast"` – (the agent moves one cell to the top and one cell to the right),
- `"east"` – (the agent moves one cell to the right),
- `"southeast"` – (the agent moves one cell to the right and one cell to the bottom),
- `"south"` – (the agent moves one cell to the bottom),
- `"southwest"` – (the agent moves one cell to the bottom and one to the left),

- `"west"` – (the agent moves one cell to the left),
- `"northwest"` – (the agent moves one cell to the left and one to the top).

The attribute `id` is a string which should contain the `REQUEST-ACTION` message identifier. The agents must plainly copy the value of `id` attribute in the `REQUEST-ACTION` message to the `id` attribute of `ACTION` message, otherwise the action message will be discarded.

Note that the corresponding `ACTION` message has to be delivered to the time indicated by the value of attribute `deadline` of the `REQUEST-ACTION` message. Agents should therefore send the `ACTION` message in advance before the indicated deadline is reached so that the server will receive it in time.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="action">
  <action id="70" type="skip"/>
</message>
```

4.3.9 `PING` (agent-2-server)

In order to allow agents to test the speed of their internet connection the simulation server provides a ping interface. The agent is allowed to send `PING` messages to the server.

The message envelope of this message contains the element `<payload>` with one attribute `value`. `value` contains an arbitrary string up to 100 characters long. The payload value is plainly copied to the payload value of the correnponding `PONG` message by the server. The agents can use it to store proprietary data in the case they need it.

Note that if the server receives a `PING` message with a payload longer than 100 characters, it can discard the `PING` message and will not respond to it.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="ping">
  <payload value="hello World"/>
</message>
```

4.3.10 `PONG` (server-2-agent)

When the simulation server receives a `PING` message it immediately responds by sending a `PONG` message.

The envelope contains one element `<payload>` with a string attribute `value`. Its value is copied from the corresponding `PING` message.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978759491" type="pong">
  <payload value="hello World"/>
</message>
```

4.4 Remarks and notes

4.4.1 Ill-formed messages

Not well-formed XML messages received by the server from agents are discarded. This means, that if some obligatory information (element, or attribute) of a given message is missing the server silently ignores it. In the case that a message will contain additional not-required informations, only the first occurence is processed by the server. We strongly recommend to comply with the communicatison protocol described above.

Examples:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="auth-request">
  <authentication username="team1agent1" password="qwErTY"/>
  <authentication username="team1agent32" password="11111WWw"/>
  <some-element arbitrary="234TreE"/>
</message>
<message type="action">
  <authentication username="team1agent1" password="qwErTY"/>
  <authentication username="team1agent1" password="qwErTY"/>
  <some-element arbitrary="234TreE"/>
</message>
```

The message above will be processed as an `AUTH-REQUEST` message with the credentials `team1agent1/qwErTY`.

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="ping">
  <payload value="payload1"/>
  <payload value="payload2"/>
</message>
```

This message will be processed as a regular `PING` message and the `PONG` response will include a payload equal to `payload1`.

4.4.2 Pinging and flooding (DoS attack attempts)

Although we do not impose any upper limits on the frequency of pinging, we strongly discourage abuse of pinging interface by either flooding the simulation server by `PING` messages, or other malformed messages with large

payload. All suspicious cases will be considered as a DoS attack attempt and will be dealt with by organizers. This can possibly lead to team disqualification.

4.5 Further Important Informations

4.5.1 Weights

The weights for the cow-algorithm are as follows:

- $w(cow) \in [1, 10]$ - attraction by cows that are visible and not in the private-range
- $w(cow_{private}) \in [-1, -10]$ - repulsion by cows that are visible and in the private-range
- $w(agent) \in [-100, -300]$ - repulsion by agents that are visible
- $w(empty) \in [1, 10]$ - attraction by empty cells
- $w(tree) := -w(empty)$ - repulsion by trees
- $w(corral) := w(empty)$ - attraction by corral cells

4.5.2 Visibility Ranges

The visibility ranges are as follows:

- cows have a visibility square that is 9 cells in width and in height, the cow is considered to be in the center,
- cows have a private square that is 3 cells in width and in height, the cow is considered to be in the center, and
- agents have a visibility square that is 17 cells in width and in height, the agent is considered to be in the center.

Please refer to the Fig. 4 and Fig. 5 and for an illustration of the ranges.

4.5.3 Probabilities

The probability that the content of a cell is not perceived by the agent is around 10%. The probability that an agent's action fails is around 10%.

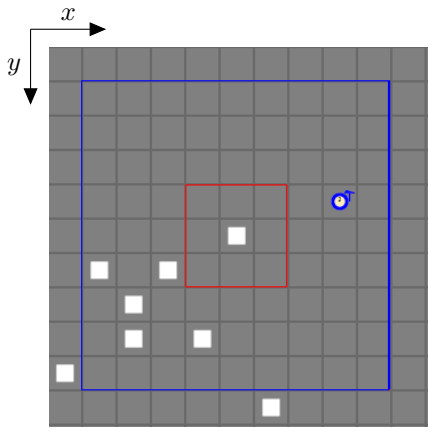


Figure 5: The view range of the cows. The cow is in the center and perceives all the cells in the blue rectangle. Cells in the red rectangle are in the private range.

4.5.4 Points

Each cow that enters a corral-cell gives the team that belongs to the corral one point for the simulation. The cow then disappears. The agent teams that wins by collecting more cows gets 3 points for the overall tournament, the loosing team gets 0 points. In the case of a draw, i.e. both agents have collected the same amount of cows, both teams get 1 point for the tournament.

5 How to change the scenario

One of the important properties wrt. the MASSim server is its modularity. For ease of use and with only low effort it is possible to adapt the scenario or even to replace the simulation plug-in completely. For this you have to change or exchange some Java classes. In the following the necessary modifications are described.

5.1 GridSimulation.java

The `GridSimulation.java` is the entry point for a simulation plug-in developer. It describes the main class `GridSimulation` that defines the simulation framework and the overall simulation structure: Firstly, while instantiating the class, it parses the config file (`GridSimulationConfiguration`)

and generates the initial world state (`GridSimulationWorldState`) of the simulation. Also, it provides an object (`GridSimulationAgent`) which represents the set of participating agents as well as the allowed agents' actions and the actions' impacts on the simulation world state. Secondly, it defines the game cycle. For this purpose the `GridSimulation` class is derived from the class `ParallelizedRandomOrderSimulation` provided by the package `massim.simulation.simplesimulation`. This package offers further classes that implement different kinds of simulations like for example a round robin simulation which is not randomized.

Note, the disjunction of the simulation, the world state and the agent's states is intended in order to allow different roles respectively different types of agents as well as a logical separation between the game mechanism and the data that describes the world. While for minor changes one only have to change the `GridSimulation*` files it might be necessary to add a new class to the package `massim.simulation.simplesimulation` when it comes to other types of games like e.g. auctions.

There is another important set of java-files, the `GridSimulation*Observer` files, wich define the `Observers`. They transmit the information about parts of the simulation to the web server, to the svg generator or to a backup file.

For more information please read the javadoc and the documentation included in the software packages.

6 Conclusion

In this technical report we have explained in detail the technical aspects of the Agent Contest. We have introduced this year's cows-and-cowboys scenario with an exact representation of the physics of the environment. We also have considered the communication protocol on which the whole communication between the agents and the MASSim platform is based. Finally we have provided a manual that allows the interested user to create her own custom-made scenario.

For us the future holds the following: 1) we are going to parallelize the MASSim platform and 2) we will improve the visualization. We did an experiment during this year's contest that showed that parallelizing the MASSim platform makes sense: allowing to have several matches in parallel saved us a lot of time. Furthermore we will switch from SVGs to Flash in respect to the visualization. And we are experimenting on more sophisticated means of real-time monitoring the ingoing matches.

References

[Dastani et al., 2006] Dastani, M., Dix, J., and Novak, P. (2006). The first contest on multi-agent systems based on computational logic. In Toni, F. and Torroni, P., editors, *Computational Logic in Multi-Agent Systems (CLIMA VI)*, volume 3900 of *Lecture Notes in Artificial Intelligence*, pages 373–384, London, UK. Springer.

[Dastani et al., 2007] Dastani, M., Dix, J., and Novak, P. (2007). The second contest on multi-agent systems based on computational logic. In Inoue, K., Satoh, K., and Toni, F., editors, *Proceedings of CLIMA '06, Revised Selected and Invited Papers*, number 4371 in Lecture Notes in Artificial Intelligence, pages 266–283, Hakodate, Japan. Springer.

[Dastani et al., 2008a] Dastani, M., Dix, J., and Novak, P. (2008a). The agent contest on multi-agent systems. In Dastani, M., Ricci, A., El Fallah Seghrouchni, A., and Winikoff, M., editors, *Proceedings of ProMAS '07, Revised Selected and Invited Papers*, number 4908 in Lecture Notes in Artificial Intelligence, Honululu, US. Springer.

[Dastani et al., 2008b] Dastani, M., Fallah-Seghrouchni, A. E., Ricci, A., and Winikoff, M., editors (2008b). *Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007, Honolulu, HI, USA, May 15, 2007, Revised and Invited Papers*, volume 4908 of *Lecture Notes in Computer Science*. Springer.