Noname manuscript No.
(will be inserted by the editor)

The Multi-Agent Programming Contest from 2005–2010

From gold collecting to herding cows

Tristan Behrens · **Mehdi Dastani** · **Jürgen Dix** ·
Michael Köster · **Peter Novák**

the date of receipt and acceptance should be inserted later

Abstract The Multi-Agent Programming Contest is an annual series of competitions in programming multi-agent systems in which a team of agents participates in a simulated cooperative scenario. It started in 2005 and is organised in 2010 for the sixth time. The Contest is an attempt to stimulate research in the area of multi-agent system development and programming by (i) identifying key problems in the field and (ii) collecting suitable benchmarks that can serve as milestones for testing multi-agent programming languages, platforms and tools.

This article provides a short history of the contest and reports in more detail on the *cows and cowboys* scenario introduced in 2009. We briefly discuss the underlying technological background and conclude with a critical discussion of the experiences and lessons learned.

Keywords AgentContest · multi-agent programming · competition · multi-agent benchmark problems · coordination

CR Subject Classification TBD

1 Introduction

The Multi-Agent Programming Contest series (Contest) is an attempt to stimulate research in the area of multi-agent system development and programming by 1) identifying key problems, 2) collecting suitable benchmarks, and 3) gathering test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. In particular, the Contest is implemented as an international on-line tournament between multi-agent systems designed to solve a cooperative task in a

P. Novák
Department of Cybernetics, Czech Technical University in Prague, Karlovo náměstí 13, CZ-12135 Prague 2, Czech Republic, E-mail: peter.novak@fel.cvut.cz

M. Dastani
Intelligent Systems Group, Utrecht University, P.O.Box 80.089, NL-3508 TB Utrecht, The Netherlands, E-mail: mehdi@cs.uu.nl

J. Dix · T. Behrens · M. Köster
Department of Informatics, Clausthal University of Technology, Julius-Albert-Str. 4, D-38678 Clausthal-Zellerfeld, Germany, E-mail: {dix, behrens}@in.tu-clausthal.de, michael.koester@tu-clausthal.de

dynamic environment. Here, we provide a detailed report of the 2009 edition of the Contest (the 4th subsequent Contest edition which we organize already since 2006). The 2010 Contest is scheduled for September 2010 and has not taken place at the time when this article was finalized. Besides the description of the 2009 Contest we also give an overview of the Contest series, the motivation behind it and the underlying technological background.

1.1 Motivation

Competitions in research communities such as the one of artificial intelligence and autonomous agent and multi-agent systems benefited tremendously from a rise of various successful efforts towards establishing a competitions applying research results in more, or less realistic challenging scenarios. Apart from platforms for evaluation and comparison of state of the art in a particular area, such competitions also serve as a driver and catalyst developments of the particular research community and their evolution tends to generate challenging research problems and thus push the boundaries of the established knowledge in the area.

Apart from the already mentioned general ambitions of a research competition, the main motivation behind the Contest was to provide a fair platform for comparison of single- and multi-agent programming frameworks and thus taste, test and challenge the current state of the art in the area. Historically, our main focus was on deliberative techniques that are based on formal approaches and computational logics. It is this focus on deliberation that resulted in some of the most characteristic features of the contest scenarios. The Contest scenarios reflect multi-agent system environments in which individual agents operate. Each agent operates in such an environment by sensing its state, deliberating about actions to be selected, and perform the decided actions in the environment. In this contest, the underlying technical infrastructure implements an environment and manages the interaction between individual agents and the environment. In particular, the contest infrastructure provides agents some information about the state of the environment, allows them to deliberate about their next step in a relatively wide time-window, and takes the agents' decisions and realizes the effect of those decisions.

The second important feature of the latest Contest editions which historically emerged is the emphasis on scenarios which not only encourage cooperative problem solving, but rather enforce it by their structure and mechanics. The Contest organizers put gradually more and more emphasis on scenarios where coordination among agents is a necessary condition for succeeding in the Contest.

Thirdly, unlike many other multi-agent competitions, in the design of the underlying technical infrastructure of the Contest we took a rather liberal stance and made sure not to impose any unnecessary constraints on the participating implementations. Namely, first and foremost we are interested in evaluating novel approaches to implementation of multi-agent coordination, task sharing, and joint activities.

As a consequence, the design and implementation of a suitable multi-agent backend infrastructure, such as e.g., inter-agent communication middleware, the employed programming language/framework as well as an execution platform, suitable for their solution are left to creativity and competence of the participants. Except a soft requirement that the implemented solutions should satisfy the definition of a decentralized multi-agent system, the Contest organizers do not impose any further constraints on the systems taking part in the Contest.

Finally, while not being a hard criterion for defining success in the Contest, implementation and facilitation of state-of-the-art techniques and approaches to programming single, as well as multi-agent systems, related methodologies, design tools and debugging techniques was always encouraged. However, in order to provide a competitive basis for evaluation of system complying with this criterion, participation of multi-agent solutions built on top of perhaps even radically different approaches was never discouraged.

1.2 Plan of this paper

In the next section we report on the past contest editions and discuss related competitions. Then we give a detailed explanation of the underlying MASSim platform and their use in the classroom. In Section 4, the main part of this paper, the 2009 edition of the contest is described. Emphasis is put on the new scenario and how it ensures cooperative behaviour among the agents. Section 5 addresses the lessons learned in the past few years and gives an outlook to the future.

2 History and other contests

The Multi-Agent Programming Contest was initiated in 2005 and its evolution can be described by three distinct phases. The first phase of the contest, which consisted on one edition, was based on the *food gatherers scenario*. This edition was organised in 2005 by M. Dastani and J. Dix (with the invaluable help of Peter Novak on all technical matters). The second phase was based on the *gold miners scenario* and consisted of two editions. These editions were organised in 2006 and 2007 by M. Dastani, J. Dix and P. Novak. Finally, the third phase was based on the *cow and cowboys scenario* and consisted of two editions. The first edition was organised in 2008 by T. Behrens, M. Dastani, J. Dix, and P. Novak, and the last edition of the this phase was organised in 2009 by the same group extended with M. Koster. Videos from all edition of this contests, the software packages as well as further information can be found on our web page[1].

The first edition of the Multi-Agent Programming Contest[2] [8] was organised in association with CLIMA-VI workshop. The scenario was a grid-like world populated by food-tokens, a depot and agents. The goal was to collect food and store it in the depot. Participants were required to submit a description of analysis, design and implementation of a multi-agent system according to the constraints given by the organizers. Also the participants were required to submit an executable implementation of a complete MAS, that is agents and environments. The submitted implementations were then compared by an evaluation committee (nine people) with respect to the following criteria:

1. the original, innovative, and effective application of computational logic techniques in solving specific multi-agent issues identified in this application,
2. the performance of the executable implementation, based on the amount of food that is collected by the multi-agent system in a certain period of time (all were executed on the same machine), and
3. the quality of the description of analysis, design and implementation of the multi-agent system, the elegance of its design and implementation, and the ease of installation and execution of the program.

[1] `http://multiagentcontest.org`

[2] `http://multiagentcontest.org/2005`

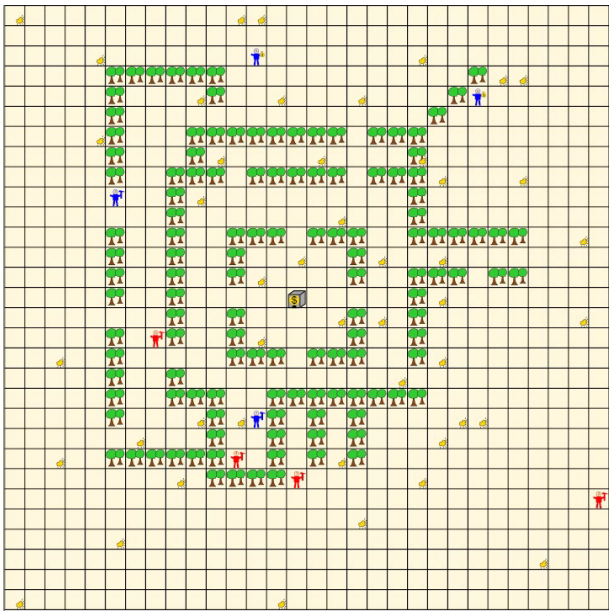


Fig. 1 The gold-miners scenario of 2006 and 2007.

In 2005 we have had four participating teams with two winning teams. The two winning teams were Simon Coffey and Dorian Gaertner [25], from Imperial College London, UK, and Carlos Cares, Xavier Franch and Enric Mayol [24], from Universitat Politecnica de Catalunya, Spain, and Universidad de la Frontera, Temuco, Chile. The third team consisted of Robert Logie, Jon G. Hall and Kevin G. Waugh [26], from Osaka Gakuin University, Japan, and the Open University, UK. The last team that participated in this edition were Eder Mateus, Nunes Goncalves and Guilherme Bittencourt [27] from Federal University of Santa Catarina. The food gatherers phase made it obvious that it is necessary to provide the environment to the participants. A single shared environment would make it easier to directly compare different agent-implementations and would of course allow the participants to concentrate on the agents, which are the focus of the competition.

The second edition of Multi-Agent Programming Contest[3] [9] was in 2006 and organised in association with CLIMA-VII workshop. This edition was marked by the publication of an open, internet-based simulation platform in which the environment was integrated. Thus everyone who could handle TCP/IP has been capable of participating in the contest. The scenario was quite similar to the food gathering scenario. Again there has been a grid-like world and a depot for collecting resources. Some obstacles were added to the environment and the food-resources have been replaced by gold-pieces. Two teams of agents competed in one and the same environment for gold. Figure 1 shows a screenshot of the visualization.

In 2006 we had three participating teams. The winner of the tournament was team *brazil* by Rafael Bordini, Jomi Hübner, and Daniel Tralamazza [22] from University of Durham, UK, Universidade Regional de Blumenau, Brazil, and Ecole Polytechnique Federale de Lausanne, Switzerland. The second place took team *spain* by Carlos Cares, Xavier Franch and

[3] `http://multiagentcontest.org/2006`

Enric Mayol [21] from Universitat Politecnica de Catalunya, Spain, and Universidad de La Frontera, Chile. The third team *germany* by Stephan Schiffel and Michael Tielscher [20] from Dresden University of Technology, Germany.

The third edition of Multi-Agent Programming Contest[4] [10] was in 2007 and organized in association with ProMAS'07 workshop. This edition of the contest was the second installment of the gold miners scenario. The environment has been adapted only in one detail, i.e., agents could be pushed away by other agents. There were six participating teams. The following is the list of participating teams where the order of the teams in the list reflects their final ranking in the competition.

1. *JiacIVteam* by A. Hessler, B. Hirsch, and J. Keiser from DAI-Labor, Technische Universität Berlin, Germany [7].
2. *microJiacteam* by E. Tuguldur, and M. Patzlaff from DAI-Labor, Technische Universität Berlin, Germany [4].
3. *Jasonteam* by J.F. Hübner from Universidade Regional de Blumenau, Brazil, and R.H. Bordini from Durham University, United Kingdom [5].
4. *FLUXteam* by S. Schiffel, M. Thielscher, and D. Thu Trang from Dresden University of Technology, Germany [3].
5. *APLteam* by L. Astefanoaei, C.P. Mol, M.P. Sindlar, and N.A.M. Tinnemeier from Utrecht University, Netherlands [6].
6. *JACKteam* by Sebastian Sardina, and Dave Scerri from RMIT University, Australia.

In 2006 the idea behind the specific design of the environment was to confront the participants with fundamental problems like obstacle avoiding and environment exploration. But during two runs of the gold miners scenario it became clear, that the scenario has been relatively easy to be handled by agents. Agents were really good at using path-planning and role-assignment to solve the given tasks, but the agents were missing *cooperation*. In fact the scenario made it easy to develop agent-teams that do not interact, but still solve the problem and achieve a high-score. Putting serious thought and experimentation into that problem yielded the next scenario: Cows and cowboys.

The fourth edition of Multi-Agent Programming Contest[5] [13] was in 2008 and organised in association with ProMAS'08 workshop. This contest edition was based on the cows and cowboys scenario. In this new scenario, the environment became highly dynamic such that individual agents were forced to cooperate and coordinate their actions. The agents were operating in a grid-like environment where a couple of obstacles were placed to make parts of the map inaccessible. Moreover, the environment consisted of cows that moved in the environment. The behavior of cows were simulated by a flocking-algorithm which was based on a mathematical model of attraction and repulsion. The goal has been to find cows, get close to them and scare them into so called corrals. Every cow that ended up in a corral yielded a point for the score. In comparison to the recent scenario(s), the set of agent-actions has been restricted to moving and the perception of the agents has been increased in viewing-range. Figure 2 shows a screenshot of the cows and cowboys scenario. In this edition of the contest we had seven participating teams. The following is the list of these teams where the order of the teams in the list reflects their final ranking in the competition.

1. *JIAC-TNG* by A. Hessler, J. Keiser, T. Küster, M. Patzlaff, A. Thiele, and Erdene-Ochir Tuguldur from Technische Universität Berlin, Germany [15].

[4] `http://multiagentcontest.org/2007`
[5] `http://multiagentcontest.org/2008`

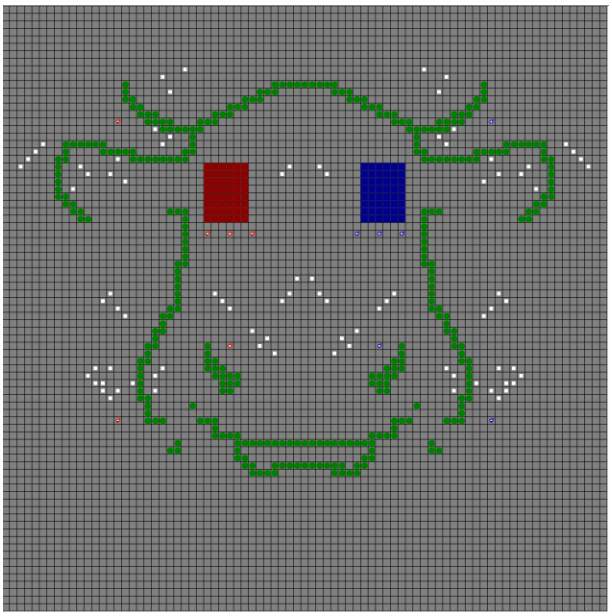


Fig. 2 The cows and cowboys scenario of 2008 and 2009.

2. *Jadex* by G. Balthasar, J. Sudeikat and W. Renz from Hamburg University of Applied Sciences, Germany [16].
3. *SHABaN* by A.T. Rahmani, A. Saberi, M. Mohammadi, A. Nikanjam, E. Adeli Mosabbeb and M. Abdoos, from Iran University Of Science and Technology, Iran [17].
4. *Krzaczory* by J. Szklarski from Institute of Fundamental Technological Research, Poland [12].
5. *Jason* by J. Hübner and Gauthier Picard from ENS Mines of Saint Etienne, France, and R. Bordini from University of Durham, UK [18].
6. *Bogtrotters* by M. Dragone, D. Lillis, C. Muldoon, R. Tynan, R.W. Collier and G.M.P. O'Hare from University College Dublin, Ireland [14].
7. *KANGAL* from Bogazici University, Istanbul, Turkey.

The fifth edition of Multi-Agent Programming Contest[6] was in 2009 and organised in association with CLIMA-X workshop. In this edition, the previous version of the cows and cowboys scenario was extended. This time, cows did not get removed from the map anymore when they enter a corral. Also fences were introduced that could be opened via switches. Finally the flocking-algorithm of the cows was adapted to improve the cow-behaviour.

2.1 Survey of related competitions

Our attempt to foster research in development of multi-agent systems for solving cooperative tasks in highly dynamic environments is by far not a solitary endeavour. In fact, RoboCup soccer challenge[7] is probably the most prominent series of competitions in the wider AI community, which stems from a motivation similar to ours. However, unlike the Multi-Agent

[6] `http://multiagentcontest.org/2009`
[7] `http://www.robocup.org/`

Programming Contest, competitions such as RoboCup soccer are aimed at benchmarking the state of the art in robotics, multi-robotics, and their integrations. As a consequence, the RoboCup soccer leagues, whether real or simulated, do not particularly focus on the state-of-the-art approaches based on formal methods of deliberation and complex planning. Such techniques are not yet mature enough to compete with more ad-hoc robot control approaches in scenarios encouraging extremely fast, though imprecise decision making in (almost) continuous spaces, such as the game of soccer.

Unlike RoboCup soccer leagues, the *RoboCup Rescue league*[8] aims at benchmarking a similar segment of AI approaches applicable in multi-agent systems. In its scenario, the concepts of team cooperation and coordination are extremely important. However, factors such as the complexity of the simulated environment (complex maps of real-world cities), hard constraints imposed in the scenario, such as e.g., the limited bandwidth of inter-agent communication and the necessity to execute the resulting multi-agent teams on the organizers' technical infrastructure significantly increase the threshold of technological difficulty participants have to overcome. In Multi-Agent Programming Contest such issues are completely at a the liberty of the Contest participants and engineering innovation. Creativity in these issues is welcomed as a first-class issue to be evaluated in the analysis of the tournament results.

Another example of a multi-agent competition similar in nature to the Multi-Agent Programming Contest is the *ORTS Real-Time Strategy Game AI Competition* (ORTS Competition). While the motivation of this series of competitions is similar to ours, the focus of the ORTS competition is on adversarial reasoning in real-time strategy games. Hence, again the speed of the participating systems matters and the evaluation scenarios presume different type of reasoning capabilities in the implemented agent teams. Although a simulation of ORTS evolves in a step-wise fashion similar to the Multi-Agent Programming Contest, the pace is significantly higher. ORTS updates the simulation 8 times per second, whereas we update once every 4 seconds. The Multi-Agent Programming Contest-environments have been discrete in space up to now. And finally, ORTS is not as open as Multi-Agent Programming Contest, when it comes to connecting agents or similar artificial entities.

Finally, a well established tournament of multi-agent systems is the *Trading Agents Competition*[9]. This contest is designed to spur research on common problems, promote definitions of benchmarks and standard problem descriptions, and showcase current technologies. The agents compete against each other in challenging market games. Each edition of this contest consists of various market games such as Auction games, Market Design games, and Supply Chain Management games. The main focus of this tournament is on models of economic behaviour aiming at maximizing (expected) revenue or profits. The focus of this contest is similar to Multi-Agent Programming Contest as models of economic behavior are specific deliberation models. However, the focus of Multi-Agent Programming Contest is on general deliberation models rather than economic behavioral models.

3 The MASSim platform

The MASSim (Multi-Agent Systems Simulation) platform is a testing environment that has been designed in order to evaluate coordination and cooperation approaches of multi-agent systems. To this end we employ round-based game simulations with the intention to evaluate agent-based approaches by letting agent teams compete against each other.

[8] `http://www.robocuprescue.org/`

[9] `http://www.sics.se/tac`

The platform itself is implemented in Java running on every operating system that provides a Java runtime environment. Participants' teams are connected via TCP/IP and exchange plain XML messages with the server. This allows developers to connect their multi-agent system (possible written in a different programming language) to the server by implementing a simple XML protocol.

A live broadcast of the simulation as well as the results of the tournament are accessible by the participants through a web server that supports Apache Tomcat. However, since the configuration of Apache Tomcat is a rather complex task we also offer a small Java program that provides a live broadcast. This enables developers to see the progress of the simulation easily. Also, after a simulation run a SVG-movie depicting the simulation progress is generated.

The actual simulation is programmed as a plug-in. The server offers an interface allowing to replace the simulation by exchanging a few classes. Therefore, a simulation developer only has to care for the game properties and not for the communication with the agents nor the visualization of the game. This permits us to evolve the simulation and to use various simulations by just loading different configuration files.

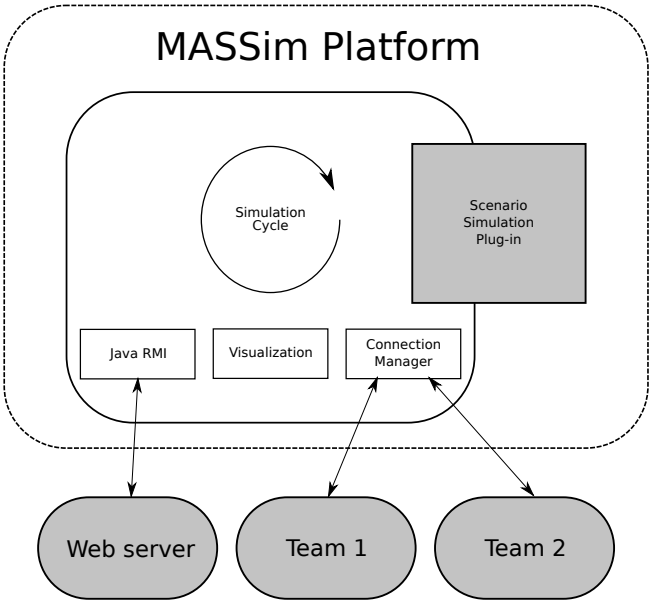


Fig. 3 MASSim platform overview

Figure 3 summarizes the technical infrastructure of MASSim. In detail, the platform consists of the following 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 of the agents. 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.
- **Agent teams:** connect to the server via TCP/IP, and communicate using XML-messages.
- **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. We also offer a script to convert these files to a flash movie.
- **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.

The modularity of the platform facilitates to use this system in the classroom. Besides the availability of a multi-agent system the students only have to start the server and the monitor in order to develop new agents. This, in combination with the competition among the students, can help to popularize agent orient programming and the approach of multi agent systems in general.

4 Multi-Agent Programming Contest 2009: Cows and herders

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 had to compete in an environment for cows. Each team controlled a group of herders in order to direct the cows into their own corral. The team with the highest number of cows in the corral at the end won the match. In the following subsections we present a general description of the Multi-Agent Programming Contest 2009. Additional information as well as the software (including all environments from the contest) are published at `http://multiagentcontest.org/2009`. The concrete structure of the XML messages are described in the appendix.

4.1 General Description

Each team competes against all other teams in a series of matches. A single match between two competing teams consists of several 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 contest, the agents from each participating team are 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 is managed locally, but the interaction between individual agents and their environment (run on the simulation server) are via Internet. Participating agents connect to the simulation server that provide the information about the environment.

Each agent from each team connects to and communicates with 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 starts. 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. After a finite number of steps the simulation server stops the cycle and the participating agents receive a notification about the end of a simulation. Then the server starts a new simulation possibly involving the same teams.

4.2 Cooperative cows herding

The 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. Furthermore there can be fences that can be opened using switches.

Each cell of the grid can be occupied by exactly one of the following objects:

- *Agents* are steered by the participants and can move from one cell to an adjacent cell.
- An *obstacle* blocks a cell.
- *Cows* are steered using a flocking algorithm. They can move from one cell to an adjacent cell. Cows tend to form herds on free areas, keeping the distance to obstacles. If and agent approaches, cows get frightened and flee.
- *Fences* can be opened using a button. To open a fence and keep it open an agent has to stand on a cell adjacent to the respective button. Thus, a switch is activated if the agent is next to the switch and:
  1. the current cell (where the agent is in) does not contain an open or closed fence, and
  2. the position of the agent is not diagonal to the switch.

Note that an agent cannot open a fence and then definitely go through it. Instead it needs help from an ally. Moreover, when fences closes, agents and cows that stand on a fence cell get pushed into a free cell. There are two corrals, which are rectangular areas, one for each team. Cows have to be pushed into these corrals. Each teams learns the position and the dimensions of its corral at the beginning of each simulation.

4.3 Agent Perceptions and Actions

Each agent perceives the contents of the cells in a fixed vicinity around it. It can distinguish between empty cells, obstacles, cows (distinguished by unique identifiers), fences, buttons, and other agents. The participants will learn the position and dimensions of their team's corral at the beginning of each simulation. Each agent can move to one of the adjacent cells

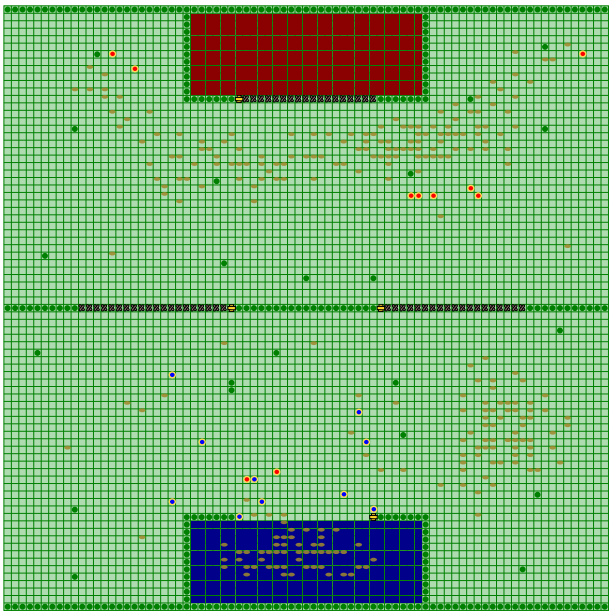


Fig. 4 The environment is a grid-like world. Agents (red and blue) are steered by the participants. Obstacles (green) block cells. Cows (brown ovals) are steered by a cow-algorithm. Fences (x-shapes) can be opened by letting an agent stand on a reachable cell adjacent to the button (slash-shaped). Cows have to be pushed into the corrals (red and blue rectangles).

if the cell is not blocked by an obstacle, cow, fence, button or another agent. Each agent perceives a square of cells of size 17×17 with the agent in the center. Each team has 10 agents.

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*.

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.

4.4 Cow Movement Algorithm

Although we do not consider the details of the cow movement algorithm to be very important, we will sketch it here. The complete algorithm is available in the source-code.

For each cow the algorithm considers all the cells that can be reached by it in one step. Then the weight of these cells is computed. The cow moves to that cell whose weight is maximal. If there are several maxima, the cow moves randomly to one of them.

The weights for attractive cells – empty space, other cows, and corral cells – are positive. The weights for repellent cells – agents and obstacles (trees, gates) – are negative. Finally, cows are slower than agents. They move every third step.

Algorithm 1 Cow movement algorithm.

Require: a cow represented by its position vector $c \in \mathbb{N} \times \mathbb{N}$
1: let N be the set of the 9 cells adjacent to c, including c;
2: remove from N all those cells that are not reachable;
3: calculate the weights of all cells $n \in N$;
4: determine the set $M \subseteq N$, where the weight for each $m \in M$ is maximal;
5: randomly pick a cell $m \in M$;
6: move the cow to m;

Algorithm 2 Calculate the weight of a given cell.

Require: a cell represented by its position vector $n \in \mathbb{N} \times \mathbb{N}$, and a cow-visibility range $r \in \mathbb{N}$
1: determine the set C of all cells that are a in the rectangle $[n_x - r, n_y - r + n_x + r, n_y + r]$ and that are on the map;
2: set *ret* to 0;
3: **for all** $c \in C$ **do**
4: calculate d the distance between c and n;
5: get the weight w of c in respect to the cell content;
6: add w/d to *ret*;
7: **end for**
8: **return** *ret*

4.5 Comparison to Multi-Agent Programming Contest 2008 and 2010

The 2009 edition of the Multi-Agent Programming Contest is essentially the same as in 2008 except for four minor differences:

- Cows are not removed from the environment if they enter the corrals. The number of cows in the corrals after the last step counts.
- The cow movement algorithm has been improved in order to yield a more convincing behavior of the cows.
- The new scenario also introduces fences.
- The team-size has been increased.

In 2010 we slightly changed the environment:

- Cows are moving faster.
- The score is calculated differently. Instead of determining the result of a simulation only at the end, we calculate an average score. This changes some messages sent from the server to the agents. For details we refer to the protocol description.
- The team-size has been increased.

4.6 Communication protocol

4.6.1 General Agent-2-Server Communication Principles

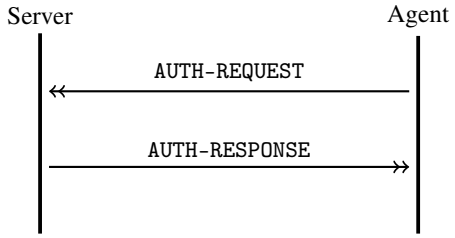
The agents from each participating team are executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, runs on the remote contest simulation server. Agents communicate with the contest server using standard TCP/IP stack with socket session interface. Agents communicate with the server by exchanging XML messages. Messages are well-formed XML documents. The exact XML structure is described in the appendix.

4.6.2 Communication Protocol Overview

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 the following three phases.

1. the initial phase,
2. the simulation phase, and
3. the final phase.

During the initial phase, agents connect to the simulation server and identify themselves by username and password (`AUTH-REQUEST` message). 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. Below is a picture of the initial phase (UML-like notation).



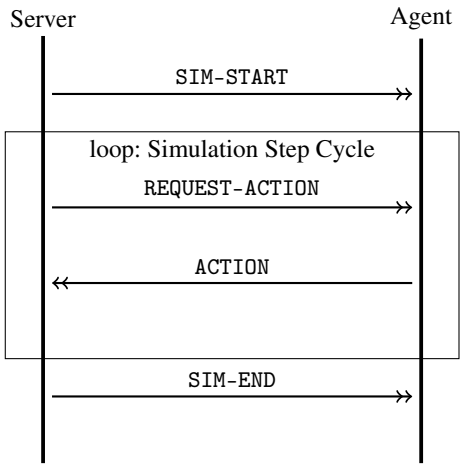
At the beginning of each simulation, agents of the two participating teams are notified (`SIM-START` message) and receive simulation specific information:

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

In each simulation step each agent receives a perception about its environment (`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 current number of cows in the team's corral, 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. Below is a picture of the simulation phase:



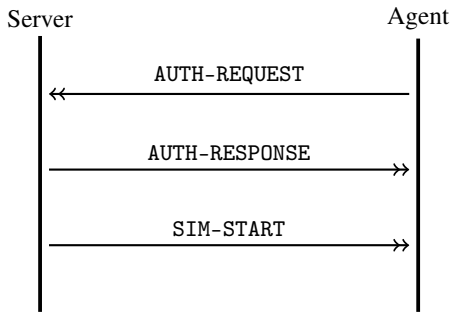
When the simulation is finished, participating agents receive a notification about its end (`SIM-END` message) which includes the information about the number of caught cows, and the information about the result of the simulation (whether the team has won or lost the simulation).

All agents which currently do not participate in a simulation have to 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. Below is a picture of the final phase.

4.6.3 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. Below is a picture for the reconnection phase.



4.7 Participants, Approaches and Results

The first team was *AF-ABLE* formed by Rem Collier, Mauro Dragone, David Lillis, Jennifer Treanor, Howell Jordan and Greg O'Hare from University College Dublin, Ireland [23]. Their solution architecture consists of AFAPL2 agents running on Agent Factory platform. The agents' low-level behaviours (explore, open fence, etc.) were written in Java. A centralized task allocation method was used that is based on costs and values. The behaviour code appeared to be rather complex and buggy and caused inconsistent performance. The team did not implement successful offensive behaviours, thus many cows conceded. Also there were no defensive behaviours, thus many successfully-herded cows escaped from the corral or got 'stolen' by other teams. The lessons learned were that agent-oriented software must be engineered carefully. Secondly, an automated test suite for behaviours is essential for the development process. The team considers to move more logic to the agent layer for the next contest.

The second team was *Jadex@HAW* formed by Gregor Balthasar, Jan Sudeikat, and Wolfgang Renz from MMLab, HAW Hamburg, Germany [1]. It was their second time participating in the Multi-Agent Programming Contest. The team used the Jadex BDI-agents middleware (v0.96). All algorithms for planning were implemented in Java. They used decentralized coordination instead of the centralized coordination that they did use in their first participation. They utilized the Tropos methodology and toolsets as a guideline. This yielded a more stable multi-agent system that needed nearly no supervision, and it increased the autonomy of the single agents drastically. Difficulties encountered were the debugging of decentralized coordination and covering all situations that can appear during a simulation.

The third team was *Jason-DTU* formed by Niklas Skamriis Boss, Andreas Schmidt Jensen and Jørgen Villadsen from the Department of Informatics and Mathematical Modelling, Technical University of Denmark [2]. Their starting point was a new advanced AI course in spring 2009 with more than 50 students. The course included two lessons and a project on logic-based agent programming using Jason. Algorithms such as A^* were implemented in Java. The Prometheus methodology was used as a guideline for development. The code from last year for the integration with the contest simulator was provided by the 2008 Jason team (cf. RomanFarmers). The Jason framework appeared to allow for easy implementation of agents with goals and plans. Three kinds of cowboys were developed: herders, a scout and a leader. A design with a single leader delegating targets leads to a less autonomous approach. The choice to heavily limit the number of cows in a single cluster is probably not optimal. And they did not implement a strategy to prohibit the opposing team from scoring points

The fourth team was *JIAC-V* formed by Axel Hessler, Thomas Konnerth, Jan Keiser, Benjamin Hirsch, Tobias Kuester, Marcel Patzlaff, Alexander Thiele, and Tuguldur Erdene-Ochir fromTechnical University Berlin, Germany [11]. It was their third participation in the Multi-Agent Programming Contest. The JIAC meta-model was the frame for design and implementation. The team has implemented an ontology-based world model (beliefs and communication vocabulary). Domain dependent (cowboy) capabilities (plans, rules) were aggregated to roles, composed with standards roles (memory, communication) to form the agent and then executed by JIAC runtime. In this implementation, they have used decentralized coordination and cooperation. Their own agile MIAC methodology and their JIAC Toolipse were used to guide the process. According to this team, the Multi-Agent Programming Contest is an interesting scenario for teaching agent-oriented principles. The team found and fixed many bugs in their framework. This concerned amongst other things the lifecycle and execution cycle of agents and the interpretation of the world model. They also tuned several core components (performance and reliability), which lead to an performance improvement by factor 8. Finally, they implemented many features that make the life of the agent-oriented programmer easier (easier to learn, easier to use, easier to debug, easier to deploy).

The fifth team was *microJIAC* formed by Anand Bayarbilig and Erdene-Ochir Tuguldur from DAI-Labor, TU-Berlin, germany [28]. They used the MicroJIAC agent framework, which has been developed by the DAI-Labor. They implemented model-based reflex agents, which consist of a world model and rules. All agents are equal and there is no specialised agent. The agents are fully self-organized and coordination/cooperation is reached by sharing perceptions/intentions. The scenario appeared to be too complex for one programmer alone. Driving a single cow does not lead to a very high score. Debugging self-organization is complicated. Finally, the agents were constructed for maps with many fences, which became visible on maps with not many fences.

The sixth team was *RomanFarmers* formed by Jomi F. Hübner from Federal University of Santa Catarina, Brazil, Rafael H. Bordini from Federal University of Rio Grande do Sul, Brazil, Gustavo Pacianotto Gouveia, Ricardo Hahn Pereira, Jaime S. Sichman from University of Sao Paulo, Brazil, Gauthier Picard from Ecole des Mines de Saint-Etienne, France, and Michele Piunti from Universita di Bologna, Italy [19]. The design was based on three paradigms and abstraction levels: 1) Organisation Oriented Programming (MOISE) to define concepts such as groups, shared plans and goals to herd, explore, and pass fences, 2) Agent Oriented Programming (Jason) to define how goals are achieved by the agents, and 3) Object Oriented Programming (Java) to define algorithms, for example, to find paths and cluster of cows. Participating in this contest have resulted in some improvement in Jason and MOISE. There was only one technical bug found in Jason. The main difficulties were debugging (several agents, tools, languages, decentralisation) and tuning of parameters (clusters max size, number of cows per cowboy).

The seventh team was *smaperteam* formed by Chenguang Zhou from RMIT, Australia [29]. It was the first time for the team to participate in the Multi-Agent Programming Contest. The team used the JACK Intelligent Agents framework for implementing the agents. The participants of this team did indicate that debugging their multi-agent program is a difficult task. Herder agents were still centralized, they communicate with a coordinator which does path finding for them by A*. The system was not stable and needed more testing.

The eight and final team was *unknown* formed by Slawomir Deren and Peter Novak from TU Clausthal, Germany. The team used the Jazzyk language with three modules. They used Open Agent Architecture for exchanging of messages. The agents consisted of two subteams, each subteam consisted of one leader and four herders. The leader searched for the

cows and coordinated the herders agents. There were two agents that were responsible for opening fences. Agents communicated in each simulation step and shared the information. The leader computed the moving direction of cows using A*. Difficulties and results were as follows. The agents were able to drive only one group of cows, the general performance of the agents was inefficient, and the amount of leaders was too small for searching.

4.8 Results

The *JIAC* team, the winner of the Multi-Agent Programming Contest 2007 and 2008, was able to climb the top position again. While in 2007 its victory was quite a triumph, in 2008 the *Jadex* team managed to be close on *JIAC*'s heels. In this year's contest it was a fight at eye level: The *Jadex* team was only three points behind the *JIAC* team. Maybe in 2010 somebody else will ascend the throne: We are convinced that it will be an exciting duel.

The following overview shows how many points each team scored and how many cows were gathered. It is the number of points that decides who wins.

1. JIAC V (1627 cows, 60 points)
2. Jadex@HAW (1345 cows, 57 points)
3. Roman Farmers (840 cows, 37 points)
4. Jason DTU (433 cows, 30 points)
5. smaperteam (194 cows, 23 points)
6. Micro JIAC (363 cows, 21 points)
7. AFABLE (468 cows, 20 points)
8. unknown (12 cows, 1 point)

As one can see the field of participants is divided into four parts: *JIAC* and *Jadex* were dominating the contest while *Roman Farmers* and *Jason DTU* performed very well but were not able to keep track of the first two teams. Highly competitive were the games between the third group, consisting of *smaperteam*, *Micro JIAC* and *AFABLE*. But even the last team succeeded to steel one point from the *smaperteam*.

5 Conclusion: Experiences and outlook

The initial idea for setting up Multi-Agent Programming Contest was to promote research in the area of multi-agent programming languages, development tools and techniques by evaluating the state-of-the-art approaches and identify key problems in this area. Soon we have realized that the contest should be designed carefully in order to enable objective evaluation and comparison of the multi-agent programming languages, development tools and platforms used by the participating teams. During the last five editions of this contest, we have modified and extended various components of the contest to meet these objectives. In particular, we have extended and modified the scenario, simulation software and the evaluation criteria of our contest.

After the successful organization of the last five editions of this contest, we still cannot give an overall account of the impact of this contest to the multi-agent programming research area. Nonetheless we have noticed that various prominent research groups in the area of multi-agent programming are enthusiastic about the contest and participate in various contest editions. After each edition, they provide us their experiences, feedbacks and suggestions which we use to improve the next edition of the contest.

They have indicated that their designed and developed programming languages, tools, and platforms are getting extended and fine-tuned based on their experiences from various editions of this contest. Beside detecting problems and weaknesses related to their approaches, they have reported general problems related to the development of multi-agent programs to the multi-agent programming community. For example, this research community is now getting aware of the need for effective debugging tools and testing approaches for multi-agent programs.

We also recognize that our contest is challenging different research groups and motivate them to work together and integrate their approaches. This is done, for example, by relating development methodologies to multi-agent programming languages, and by building standards for interactions among different agent models, between agent and environment models, between agent models and development tools, and between agent models and their organisations. The participating teams have been confronted with the need to respect fundamental programming principles such as modularity and separation of concerns. The contest has also challenged the performance and scalability of the existing platforms. Finally, the technical infrastructure and software that are built for this contest are used for teaching purposes at different universities, in particular, at the universities of the participating teams.

References

1. Gregor Balthasar, Jan Sudeikat, and Wolfgang Renz. On the decentralized coordination of herding activities: A jadex-based solution. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
2. Niklas Skamriis Boss, Andreas Schmidt Jensen, and Jørgen Villadsen. Building multi-agent systems using jason. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
3. M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors. *An Agent Team Based on FLUX for the ProMAS Contest 2007*, volume 4908 of *Lecture Notes in Artificial Intelligence*, Honululu, US, 2008. Springer.
4. M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors. *Collecting Gold*, volume 4908 of *Lecture Notes in Artificial Intelligence*, Honululu, US, 2008. Springer.
5. M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors. *Developing a Team of Gold Miners Using Jason*, volume 4908 of *Lecture Notes in Artificial Intelligence*, Honululu, US, 2008. Springer.
6. M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors. *Going for Gold with 2APL*, volume 4908 of *Lecture Notes in Artificial Intelligence*, Honululu, US, 2008. Springer.
7. M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors. *JIAC IV in Multi-Agent Programming Contest 2007*, volume 4908 of *Lecture Notes in Artificial Intelligence*, Honululu, US, 2008. Springer.
8. Mehdi Dastani, Jürgen Dix, and Peter Novák. The first contest on multi-agent systems based on computational logic. In Francesca Toni and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI*, volume 3900 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2005.
9. Mehdi Dastani, Jürgen Dix, and Peter Novák. The second contest on multi-agent systems based on computational logic. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems, 7th International Workshop, CLIMA VII*, volume 4371 of *Lecture Notes on Computer Science*, pages 266–283. Springer, 2006.
10. Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition - 3rd edition. In M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors, *Proceedings of the fifth internation workshop on Programming Multi-Agent Systems (ProMAS'07)*, number 4908 in Lecture Notes in Artificial Intelligence, Honululu, US, 2008. Springer.
11. Axel Hessler, Benjamin Hirsch, and Tobias Küster. Herding cows with jiac-v. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
12. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *AC08 System Description*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
13. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *Agent Contest Competition: 4th edition*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.

14. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *Dublin Bogtrotters: Agent Herders*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
15. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *Herding Agents - JIAC TNG in Multi-Agent Programming Contest 2008*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
16. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *On Herding Artificial Cows: Using Jadex to Coordinate Cowboy Agents*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
17. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *SHABaN Multi-agent Team to Herd Cows*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
18. Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors. *Using Jason and Moise+ to Develop a Team of Cowboys*, volume 5442 of *Lecture Notes in Computer Science*, Estoril, Portugal, 2009. Springer.
19. Jomi F. Hübner and Rafael H. Bordini. Using agent- and organisation-oriented programming to develop a team of agents for a competitive game. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
20. K. Inoue, K. Satoh, and F. Toni, editors. *Multi-Agent FLUX for the Gold Mining Domain*, volume 4371 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.
21. K. Inoue, K. Satoh, and F. Toni, editors. *Using Antimodels to Define Agents Strategy*, volume 4371 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.
22. K. Inoue, K. Satoh, and F. Toni, editors. *Using Jason to Implement a Team of Gold Miners*, volume 4371 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.
23. Howell Jordan, Jennifer Treanor, David Lillis, Mauro Dragone, Rem W. Collier, and G.M.P. O'Hare. Af-able in the multi agent contest 2009. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
24. F. Toni and P. Torroni, editors. *Extending tropos for a prolog implementation: A case study using the food collecting agent problem*, volume 3400 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.
25. F. Toni and P. Torroni, editors. *Implementing pheromone-based, negotiating forager agents*, volume 3400 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.
26. F. Toni and P. Torroni, editors. *Reactive food gathering*, volume 3400 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.
27. F. Toni and P. Torroni, editors. *Strategies for multi-agent coordination in a grid world*, volume 3400 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.
28. Erdene-Ochir Tuguldur and Marcel Patzlaff. Moo - microjiac agents operating oxen. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.
29. Nitin Yadav, Chenguang Zhou, Sebastian Sardina, and Ralph Ronnquist. A bdi agent system for the cow herding domain. *Annals of Mathematics and Artificial Intelligence*, This Volume, 2010.

A 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.

A.1 `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`.

Example:

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

A.2 `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>
```

A.3 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). center;

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"
    lineOfSight="8"
    opponent="xteam"
    steps="10"/>
</message>
```

A.4 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>
```

A.5 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"/>
```

A.6 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 in the corral of the agent's team at 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".
- <switch> - the cell contains a switch.
- <fence> - the cell contains a segment of a fence. Allowed values for the attribute open are "true" and "false".
- <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. 5):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1243942024822" type="request-action">
<perception deadline="1243942032822" id="1" posx="16" posy="20" score="0"
step="0">
<cell x="-8" y="-8">
<empty/>
</cell>
<cell x="-8" y="-7">
<empty/>
</cell>
<cell x="-8" y="-6">
<empty/>
</cell>
<cell x="-8" y="-5">
<obstacle/>
</cell>
<cell x="-8" y="-4">
<obstacle/>
</cell>
<cell x="-8" y="-3">
<obstacle/>
</cell>
<cell x="-8" y="-2">
<obstacle/>
</cell>
<cell x="-8" y="-1">
<obstacle/>
```

```
</cell>
<cell x="-8" y="0">
<obstacle/>
</cell>
<cell x="-8" y="1">
<obstacle/>
</cell>
<cell x="-8" y="2">
<obstacle/>
</cell>
<cell x="-8" y="3">
<obstacle/>
</cell>
...
<cell x="-7" y="-4">
<corral type="ally"/>
</cell>
<cell x="-7" y="-3">
<corral type="ally"/>
</cell>
<cell x="-7" y="-2">
<corral type="ally"/>
</cell>
<cell x="-7" y="-1">
<corral type="ally"/>
</cell>
<cell x="-7" y="0">
<corral type="ally"/>
</cell>
<cell x="-7" y="1">
<corral type="ally"/>
</cell>
<cell x="-7" y="2">
<corral type="ally"/>
</cell>
...
<cell x="-1" y="-4">
<fence open="false"/>
</cell>
<cell x="-1" y="-3">
<fence open="false"/>
</cell>
<cell x="-1" y="-2">
<fence open="false"/>
</cell>
<cell x="-1" y="-1">
<fence open="false"/>
</cell>
<cell x="-1" y="0">
<fence open="false"/>
</cell>
<cell x="-1" y="1">
<fence open="false"/>
</cell>
<cell x="-1" y="2">
<switch/>
</cell>
...
<cell x="2" y="-3">
<cow ID="0"/>
```

```
</cell>
<cell x="2" y="-2">
<empty/>
</cell>
...
<cell x="5" y="-2">
<agent type="enemy"/>
</cell>
...
<cell x="8" y="8">
<empty/>
</cell>
</perception>
</message>
```

Note that the agent perceives an area that is a square with the size 17 with the agent in the center (see Fig. 5). 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.

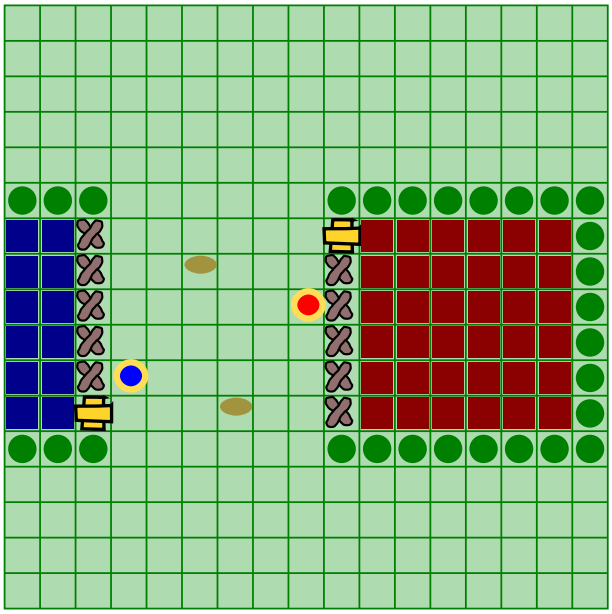


Fig. 5 The view range of the agents. The agent in the center perceives all depicted cells.

A.7 `ACTION` (agent-2-server)

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

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 `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>
```

A.8 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`.