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 original publication is available at www.springerlink.com
http://dx.doi.org/10.1007/978-3-642-11198-3_3

Abstract The Multi-Agent Programming Contest is an annual international event on programming multi-agent systems: Teams of agents participate 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 since it started and reports in more detail on the *cows and cowboys* scenario implemented for the 2008, 2009 and 2010 contest editions. 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 (shortly 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

T. Behrens · J. Dix · 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

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

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

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 dynamic environment. Here, we provide a detailed report of the last two editions of the Contest (the 4th and 5th subsequent Contest edition which we organize already since 2006). Besides the description of the last two Contest's we also give an overview of the Contest series, the motivation behind it and the underlying technological background.

1.1 Motivation

Research communities such as *artificial intelligence* and *autonomous agent and multi-agent systems* benefited tremendously in recent years from a rise of various successful efforts towards establishing competitions to evaluate research results in more or less realistic scenarios. Apart from comparing state of the art systems in a particular area, such competitions also serve as a driver and catalyst of developments of the respective research community. Competitions tend to generate challenging research problems and thus push the boundaries of the established state of the art.

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 are designed so that individual agents are embodied in a dynamic environment and operate by sensing its state, deliberating about actions to be selected, and performing the decided actions in it. 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, takes the agents' decisions and realizes the effect of those decisions.

The second important feature of the latest Contest editions is the emphasis on scenarios which not only encourage cooperative problem solving, but rather enforce it by their structure. 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. But we do not reject approaches that are programmed in a language that has nothing to do with agent systems.

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 for 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 Paper Outline

In the next section we report on the past Contest editions and discuss related competitions. Section 3 is giving an overview of related competitions. In Section 4 we provide a detailed description of the underlying *MASSim* technical infrastructure.

In Sections 5 and 6,, the main part of this paper, the 2009 and 2010 editions of the contest are described. Emphasis is put on the new scenario and how (we believe!) it ensures cooperative behaviour among the agents. Section 7 addresses the lessons learned in the past few years and gives an outlook to the future.

2 History of The Contest

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 of 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 Novák on all 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. Novák. Finally, the third phase was based on the *cow and cowboys scenario* and consists of three editions. The first edition was organised in 2008 by T. Behrens, M. Dastani, J. Dix, and P. Novák, and the last two editions of the this phase were organised in 2009 and 2010 by the same group and, in addition, by M. Köster. Videos from all editions of these 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] [**?**] took place in association with the 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 with respect to the following criteria:

1. original, innovative, and effective applications of computational logic techniques in solving specific multi-agent issues identified in this application,
2. 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

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

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

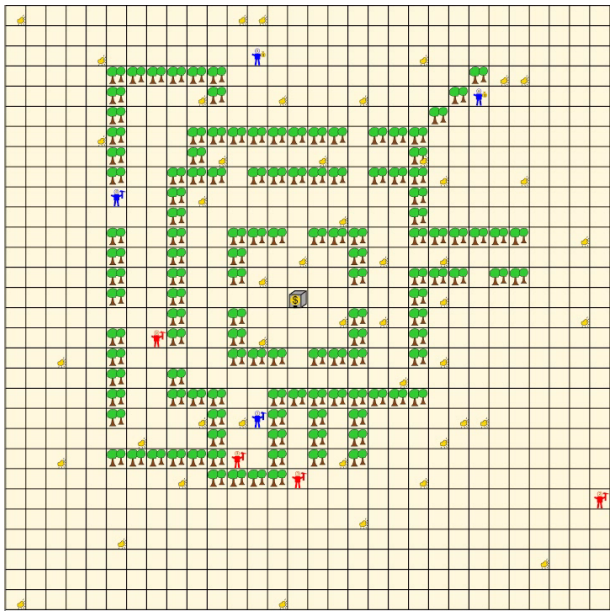


Fig. 1 The gold-miners scenario used in 2006 and 2007 editions. The maze is constructed of trees (solid obstacles) and in its centre there is the depot for the collected gold nuggets. There are two teams of agents/miners (red and blue) navigating within the grid and searching for the gold nuggets scattered around.

3. 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.

In 2005 we have had four participating teams with two winning teams. The two winning implementations were by S. Coffey and D. Gaertner ([**?**]), from Imperial College London, UK, and by C. Cares, X. Franch and E. Mayol ([**?**]), from Universitat Politecnica de Catalunya, Spain, and Universidad de la Frontera, Temuco, Chile. The third team consisted of R. Logie, J.G. Hall and K.G. Waugh ([**?**]), from Osaka Gakuin University, Japan, and the Open University, UK. The last team that participated in this edition consisted of E. Mateus, N. Goncalves and G. Bittencourt ([**?**]) from Federal University of Santa Catarina, Brasil.

The 2005 contest edition scenario made it obvious that in order for a fair comparison of the participating implementations it is necessary to provide a standard technical infrastructure to the participants. A single shared environment would make it easier to directly compare different agent-implementations and would take away the burden to deal with low-level technical details from the participants and allow them to concentrate on the implementation of agents' internal logic, which is the focus of the competition.

The second edition of Multi-Agent Programming Contest[3] [**?**] took place in 2006 and was organised again in association with the CLIMA-VII workshop. This edition was marked by the publication of an open, internet-based simulation platform in which the environment was integrated. Thus, the contest scenario was decoupled from an agent implementation platform and as a consequence, the technical threshold for entering the contest was lowered as much as possible. In fact, the only technological requirement on the side of an agent team

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

implementor was the capability to handle TCP/IP socket connections and process relatively simple XML documents exchanged as messages. The simulation scenario was however still quite similar to the food gathering scenario. Again there was 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 together with a description of depicted entities.

In the 2006 edition of the Contest we had only three officially participating teams and a quick naive agent team implementation by the Contest organizers intended as a base line benchmark implementation. The winner of the tournament was the team *brazil* jointly developed by R. Bordini, J. Hübner, and D. Tralamazza [**?**]) from University of Durham, UK, Universidade Regional de Blumenau, Brazil, and École Polytechnique Fédérale de Lausanne, Switzerland. The second place took the team *spain* by C. Cares, X. Franch and E. Mayol ([**?**]) from Universitat Politecnica de Catalunya, Spain, and Universidad de La Frontera, Chile. The team *germany* developed by S. Schiffel and M. Tielscher ([**?**]) of Dresden University of Technology, Germany finished third place. In fact, curiously, the unofficially competing benchmark agent team, *CLIMABot* (developed within one evening by our former student Michael Köster just for testing purposes), managed to achieve the highest score.

The third edition of Multi-Agent Programming Contest[4] [**?**], organized in 2007 in association with the ProMAS'07 workshop, was the second installment of the gold miners scenario. The environment has been improved in only one detail: Agents could be pushed away by other agents. We implemented this feature in order to hinder implementation of defensive depot blocking strategies and in turn to increase the competitiveness of the scenario. There were six participating teams (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 [**?**].
2. *microJiacteam* by E. Tuguldur, and M. Patzlaff from DAI-Labor, Technische Universität Berlin, Germany [**?**].
3. *Jasonteam* by J.F. Hübner from Universidade Regional de Blumenau, Brazil, and R.H. Bordini from Durham University, United Kingdom [**?**].
4. *FLUXteam* by S. Schiffel, M. Thielscher, and D. Thu Trang from Dresden University of Technology, Germany [**?**].
5. *APLteam* by L. Astefanoaei, C.P. Mol, M.P. Sindlar, and N.A.M. Tinnemeier from Utrecht University, Netherlands [**?**].
6. *JACKteam* by S. Sardina, and D. Scerri from RMIT University, Australia.

The idea behind the simulation scenario employed in the 2006 edition was to confront the participants with challenges for the development of individual agents. The core problems were basically *obstacle avoidance* and *environment exploration*. During the two editions employing the gold miners scenario it became clear that the scenario has been relatively easily handled by agents: *Using cooperative behaviours within a team did not yield significantly better results than solutions based on perfectly self-interested agents*. In fact the scenario made it easy to develop agent-teams that do not interact at all, but still solve the problem and achieve a high-score merely due to efficient path-planning and information sharing. Reflection on these shortcomings resulted in a redesign of the simulation scenario and a proposal for a new type of game.

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

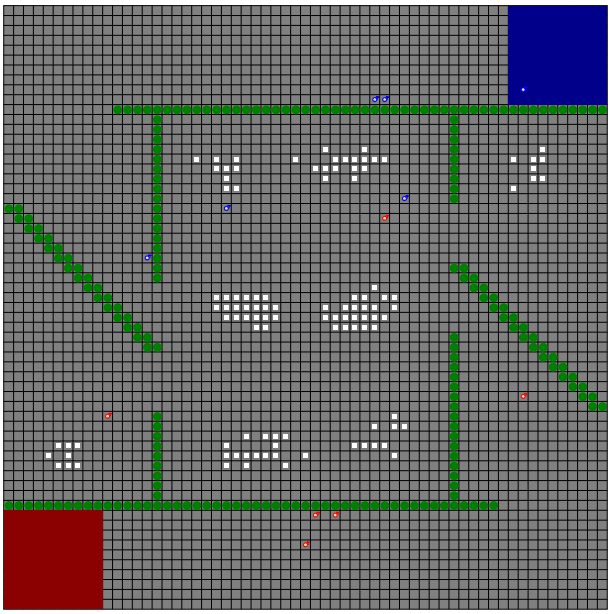


Fig. 2 The cows and cowboys scenario employed in 2008 and 2009 contest editions. The green cells represent obstacles, there are two rectangular corrals each belonging to one of the two agent teams (red and blue) and finally a number of cows scattered around the grid.

We organized the fourth edition of Multi-Agent Programming Contest[5] [**?**] in the spring 2008 again in association with the ProMAS'08 workshop. This edition was based on the new cows and cowboys scenario. In this new scenario, the environment became more dynamic: The individual agents were indirectly forced to cooperate and coordinate their actions in order to achieve any positive results. The agents were operating in a grid-like environment again featuring trees and bushes as obstacles. The environment contained cows, active entities in themselves, that moved in the environment. Their behavior was implemented by a relatively simple flocking-algorithm based on a model of attraction and repulsion. The cows were modeled so that they were strongly repelled by (or scared of) agents in the environment but they were also attracted to each other at the same time. The goal of the agent teams was to find cows, get close to them and push (scare) them into a corral belonging to each individual team. Each cow that ended up in a corral scored one point. In comparison to the past scenario, the set of agent-actions has been restricted to moving only and their range of view has been enlarged. Figure 2 shows a screenshot of the cows and cowboys scenario.

In this edition of the contest we had seven participating teams. The following 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 [**?**].
2. *Jadex* by G. Balthasar, J. Sudeikat and W. Renz from Hamburg University of Applied Sciences, Germany [**?**].
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 [**?**].

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

4. *Krzaczory* by J. Szklarski from Institute of Fundamental Technological Research, Poland [**?**].
5. *Jason* by J. Hübner and G. Picard from ENS Mines of Saint Etienne, France, and R. Bordini from University of Durham, UK [**?**].
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 [**?**].
7. *KANGAL* from Bogazici University, Istanbul, Turkey.

We organized the fifth edition of Multi-Agent Programming Contest[6] in 2009 in association with the CLIMA-X workshop. In this edition, the previous version of the cows and cowboys scenario was slightly extended. The cows were not removed from the environment anymore upon entering a corral. We also introduced a new feature: Fences that could be opened via switches. Finally the flocking-algorithm of the cows was adapted to improve the cow behaviour.

This minor redesign of the old 2008 edition scenario was again aimed at increasing competitiveness of the scenario as well as to increase the need for cooperation. In particular, persistence of cows in corrals enabled more aggressive game strategies, such as entering the opponents corral and pushing the collected cows away from it. Secondly, due to the introduction of fences, the agent team had to coordinate its members in order for one of the cowboys to keep a fence open while the remaining team members were pushing cows along the desired trajectory. Due to the improvements in the cow-mobility model, it was also more difficult to separate an individual cows from the herd. We present the details in section 5.

Finally, in 2010, we restructured the team and established a Steering Committee. Its main task is to collect and maintain the know how of the Contest organization and to overlook the Contest edition organization and advises the organizing committee in order to ensure persistent striving for the long-term aims. The Organization Committee, on the other hand, takes care for the operational aspects of the Contest organization. It is appointed by the Steering Committee for the purpose of organizing the single upcoming edition of the Contest, possibly associated with another workshop or conference.

Considering the scenario we only changed some minor details to enforce even more the cooperation of the agents. Also, we modified the calculation of the score. While in the previous scenarios only the number of cows at the end of the game were counted, for the Contest in 2010 we introduced a new measurement based on the *average of captured cows per step*. This way, we alleviated the effectiveness of bad strategies like stealing cows in the last few steps.

The sixth edition of Multi-Agent Programming Contest[7] is described in section 6.

3 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, the RoboCup soccer challenge[8] 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 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

[6] `http://multiagentcontest.org/2009`
[7] `http://multiagentcontest.org/2010`
[8] `http://www.robocup.org/`

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*[9] 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*[10]. 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.

Finally, a well established tournament of multi-agent systems is the Trading Agents Competition[11]. 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. Unlike the Multi-Agent Programming Contest, the main focus of this tournament is on models of economic behaviour aiming at maximizing (expected) revenue or profits.

4 The *MASSim* Platform

The *MASSim* (*Multi-Agent Systems Simulation*) platform is a testbed environment that we designed specifically for the needs of the Multi-Agent Programming Contest tournament series. Its aim is to enable a fair evaluation of coordination and cooperation approaches of multi-agent systems featuring rather complex deliberative agents. To this end we employ round-based game simulations with the intention to benchmark various agent-based approaches by letting agent teams compete against each other.

The platform itself is implemented in Java running on each operating system that provides a Java runtime environment. Participants' teams are connected via TCP/IP and exchange plain XML messages with the simulation server. This allows developers to connect

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

[10] `http://skatgame.net/mburo/orts/index.html`

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

their multi-agent system (possible written in a different programming language) to the server by implementing socket communication and processing relatively simple protocol based on XML messages.

The actual simulation scenario is programmed as a plug-in. The server offers an interface allowing to replace the simulation by exchanging a few classes. As a result, a simulation developer only has to care for the core game properties and not for the low-level issues connected with the server-agents communication nor with the visualization of the game. This permits us to rapidly develop and modify the simulation scenarios and to use different simulations by just loading different configuration files.

During a tournament, a live broadcast of the running simulations as well as the latest results of the tournament are accessible by the participants through an optionally installed web interface. Besides that, for debugging purposes the *MASSim* toolkit also provides a simpler Java tool able to connect to a running *MASSim* tournament server and retrieve the actual simulation status. Finally, the *MASSim* server visualization component records all the simulation runs as a series of SVG drawn snapshots which are afterwards replayable off-line in an SVG-enabled web browser.

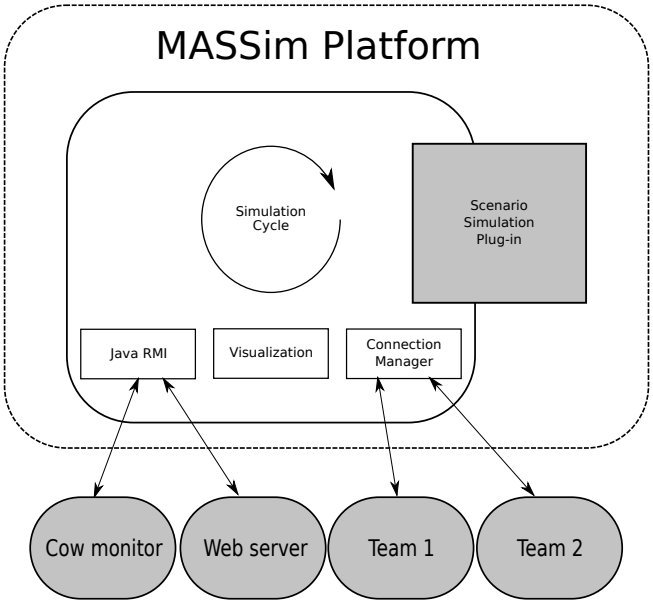


Fig. 3 *MASSim* platform overview.

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 a simple 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 encoding their actions as XML messages and transmitting them to the server.
- **Teams of agents:** agents 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 server:** 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.
- **Cow monitor:** is provided for debugging purposes.

The modularity of the platform allows 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.

Moreover, the scenarios implemented in the Multi-Agent Programming Contest can also be seen as *IT ecosystems* [**?**], i.e., systems composed of a large number of distributed, decentralized, autonomous, interacting, cooperating, organically grown, heterogeneous, and continually evolving subsystems. While the *MASSim* platform facilitates the interaction of different agent teams, each agent team can be described as well as a autonomous system containing smaller systems (the agents), so that, in the end, we get a hierarchy of systems. As the participants are from different universities it's highly decentralized and because of the different multi-agent platforms it is also heterogeneous. In the future we even plan to include cooperation between different teams in order to strengthen the interaction.

5 Multi-Agent Programming Contest 2009: Cows and Cowboys

After presenting the history and the underlying technological framework we proceed with describing the cows and herders scenario implemented for the 2009 Contest edition. To make a contest scenario more accessible for first-timers, we often provide a *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 dairy-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.

To solve this task each team controlled a group of herders trying 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 detailed 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.

5.1 General Description

As always, 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.

As explained in the previous sections, 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.

5.2 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 distance to obstacles. If an 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:

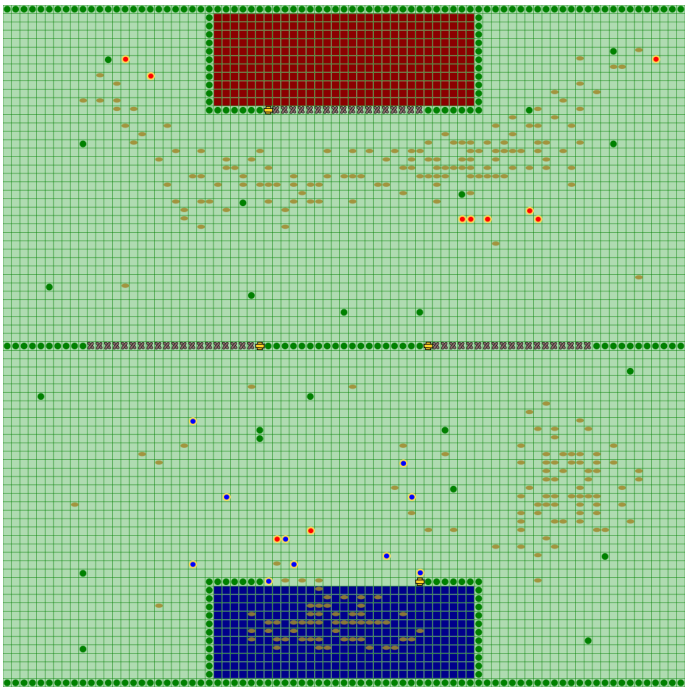


Fig. 4 The environment is a grid-like world. Agents (red and blue) are steered by the participants. Obstacles (green) block cells. Cows (brown ovals) are steered by a cow-algorithm. Fences (x-shapes) can be opened by letting an agent stand on a reachable cell adjacent to the button (slash-shaped). Cows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

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 another friendly agent. Moreover, when fences close, 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.

5.3 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 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.

5.4 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.

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*

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.

5.5 Comparison to Multi-Agent Programming Contest 2008

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.

5.6 Communication Protocol

5.6.1 General Agent-2-Server Communication Principles

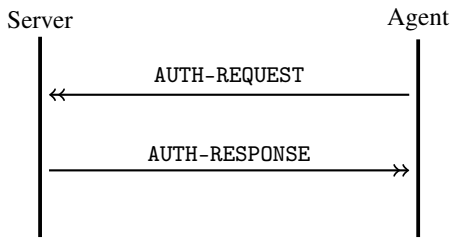
The agents from each participating team are executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, runs on the remote contest simulation server. Agents communicate with the contest server using standard TCP/IP stack 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.

5.6.2 Communication Protocol Overview

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 an 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:

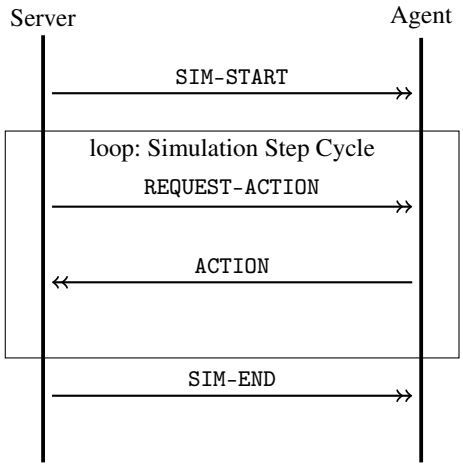
- simulation ID,
- opponent's ID,
- grid size,

- 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:



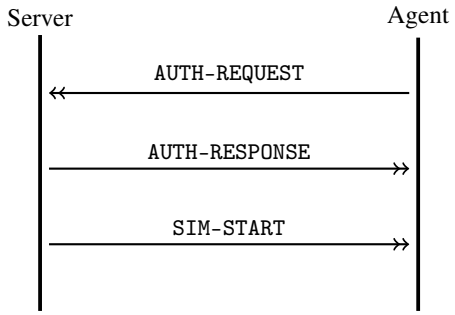
When the simulation is finished, participating agents receive a notification about its end (`SIM-END` message) which includes the information about the number of caught cows, and the information about the result of the simulation (whether the team has won or lost the simulation).

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.

5.6.3 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.



5.7 Participants and Approaches

We will now introduce all the participating teams in an alphabetical order.

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 [**?**]. 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 team considers to move more logic to the agent layer for the next contest edition.

The second team was *Jadex@HAW* formed by Gregor Balthasar, Jan Sudeikat, and Wolfgang Renz from MMLab, HAW Hamburg, Germany [**?**]. 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 [**?**]. 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 events 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. The team 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 from Technical University Berlin, Germany [**?**]. 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 a 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, and easier to deploy).

The fifth team was *microJIAC* formed by Anand Bayarbilig and Erdene-Ochir Tuguldur from DAI-Labor, TU-Berlin, Germany [**?**]. 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. They found debugging self-organization to be quite complicated.

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 [**?**]. 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 which plans should be selected and performed 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 has 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 and number of cows per cowboy).

The seventh team was *smaperteam* formed by Chenguang Zhou from RMIT, Australia [**?**]. 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 did path finding for them using the A* algorithm. 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[12]. The team used the Jazzyk language with three modules. They used Open Agent Architecture for exchanging of messages. The agent team 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*. 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.

5.8 Results

The *JIAC* team, the winner of the Multi-Agent Programming Contest 2007 and 2008, won the tournament again. While in 2007 there was a big gap between the first and the second place, in 2008 the *Jadex* team got quite close to *JIAC*: The *Jadex* team was only three points behind the *JIAC* team.

The following overview shows how many cows were gathered and how many points each team scored. 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 three subgroups: *JIAC* and *Jadex* were dominating the contest while *Roman Farmers* and *Jason DTU* performed well but were not able to keep track of the first two teams. Finally, the matches in the third group resulted in some competitive games between the teams *smaperteam*, *Micro JIAC* and *AFABLE*. But even the last team succeeded to steal one point from the *smaperteam*.

[12] Note, that this team did not participate officially. That is, the team was introduced just as a kind of bot in the tournament.

6 Multi-Agent Programming Contest 2010: Cows and Cowboys

In 2010, we used basically the same scenario and changed just some small rules. Therefore, we only present an overview here and refer for the details to the corresponding section of the Contest 2009 respective the appendix. The fiction of the unknown species of cattle had to serve as background story again. Each team controlled once more a group of cowboys steering cows into their own corral. But this time not the team with the highest number of cows in the corral at the end won the match but that team that caught the most cows in average regarding the number of steps. Also, teams consist of 20 agents instead of 10. Additional information as well as the software (including all environments from the contest) are published at `http://multiagentcontest.org/2010`.

Each team competed against all other teams in a series of matches. A single match between two competing teams consisted of several simulations. Winning a simulation yielded 3 points for the team, a draw was worth 1 point and a loss 0 points. The winner of the whole tournament was evaluated on the basis of the overall number of collected points in all the matches during the tournament.

The environment was a rectangular grid consisting of cells. The simulated environment contained two corrals—one for each team—which served as a location where cows should be directed to. Furthermore there were fences that could be opened using switches. Each cell of the grid could be occupied by exactly one of the following objects: *Agents*, *obstacles*, *cows* and *fences*. Figure 5 depicts a map used in the contest.

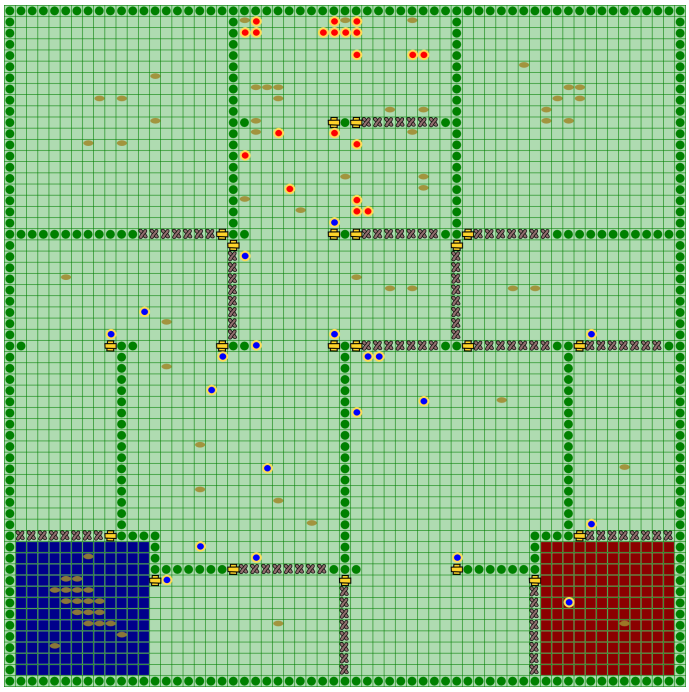


Fig. 5 The environment is a grid-like world. Agents (red and blue) are steered by the participants. Obstacles (green) block cells. Cows (brown ovals) are steered by a cow-algorithm. Fences (x-shapes) can be opened by letting an agent stand on a reachable cell adjacent to the button (slash-shaped). Cows have to be pushed into the corrals (red and blue rectangles).

The Agent perceptions and the actions available were the same as in 2009. Each agent perceived the content of the cells in a radius of 8, i.e., a square of size 17×17 with the agent in the centre. It reacted to the received sensory information by indicating which action (*move* or *skip* action) it wants to perform in the environment. If no reaction was received from the agent within the given time limit, the simulation server assumed that the agent performed the *skip* action. Agents had only a *local view* of their environment, their *perceptions could be incomplete*, and it was possible that *actions fail*, however the server never provided incorrect information.

Concerning the cow movement algorithm we changed some parameters to make cow steering more challenging. The cows were moving in each step and they were choosing direction more randomly.

The communication protocol was only changed in a minor way to cope with the new average score. Details can be found in the protocol description available at our homepage[13]. To sum up, for the Multi-Agent Programming Contest 2010 we slightly changed the environment, so that cows were moving faster, the team-size was increased to 20 agents and the score was calculated differently. Instead of determining the result of a simulation only at the end, we calculated an average score.

6.1 Participants and Approaches

We will now introduce all the participating teams in an alphabetical order. The first team was *Argonauts* formed by A. Löwen, B. Jablkowski, W. Cai, D. Hölzgen, E. Böhmer, F. Bienek, M. Kruse, S. Broszeit, T. Vengels from Technical University Dortmund, Germany[**?**]. They were using a modified *Jason* framework combined with *DLV*. Collaboration was done by the means of the role system *specified by Gaia*. The assignment was steered by the agents and not by a centralized master agent.

The second team was *Brainbug* formed by Fabian Linges, Sönke Sieckmann, Erik Stürmer, Jonathan Ziller, Axel Heßler from Technical University Berlin, Germany[**?**]. The team has been developed in a student course. It was their fourth participation in the Multi-Agent Programming Contest. The *JIAC V* meta-model was the frame for design and implementation. This time they used a centralized approach with a manager agent collecting the perceptions of all agents and thus maintaining one world model for all agents and coordinating the behaviour of the team. This additional agent was responsible for deciding the overall strategy of the normal agents that connect to the *MASSim* platform. To keep some degree of independence in the normal agents they implemented a role system. In each step the manager agent assigned to each agent a role (explorer, switcher, shepherd) and a task. The agent then decided by itself what it has to do in order to accomplish its goal.

The third team was *CowRaiders* formed by Bettina Feldheim, Erik Petzhold, Michael MÃŒller, Verena Klös and Marcel Patzlaff from DAI-Labor, TU-Berlin, Germany and Jonas Hartmann from Universidade Federal do Rio Grande do Sul, Brazil. The team has been developed in a student course. They used a centralised coordination approach by designing an additional coordination agent that awaited perceptions of each normal agent and answered with appropriate instructions. The coordination agent had the complete knowledge of the team and was able to make decisions in time. It sent goals to normal agents which they followed autonomously. However, the system itself was distributed and its composition was based on the agent model of *MicroJIAC* agent framework, which has been developed by

[13] `http://multiagentcontest.org/2010`

the DAI-Labor. The team implemented a role system, i.e., the coordination agent assigned roles (explorer, switcher, shepherd) as well as goals to the normal agents.

The fourth team was *Galoan* formed by Vahid Rafe, Aghil Bayat, Mohammed Rezaei and Yousef Purnaghi from the Computer Engineering Department, Islamic Azad University-Malayer Branch, Malayer, Iran[**?**]. Their system did not follow a multi-agent system approach but simulated the desired behaviour of a human facing the same problem. It was implemented in the object oriented programming language *Java*. It featured two components, a supervisor and a world model. Percepts were reflected in the world model and the supervisor decided what action to choose for each agent depending on the world model.

The fifth team was *Jason-DTU* formed by Jørgen Villadsen, Niklas Skamriis Boss, Andreas Schmidt Jensen and Steen Vester from the Department of Informatics and Mathematical Modelling, Technical University of Denmark[**?**]. They participated in the contest for the first time in 2009. The Prometheus methodology was used as a guideline for development, i.e., they adapted relevant concepts from the methodology, while not following it too strictly. The Jason framework appeared to allow for easy implementation of agents with events and plans. Three kinds of cowboys were developed: herders, a scout and a leader. Communication was primarily between individual agents and the leader. A design with a single leader delegating targets leads to a less autonomous approach. In fact, it leaded to a centralized coordination mechanism.

The sixth team was *PauLo* formed by Christian Loosli and Adrian Pauli from Berne University of Applied Sciences, Switzerland. They did not use an existing multi-agent system methodology. Instead the software was implemented in *Java*. Agents were mapped to threads sending percepts to an centralized organisation that then decided what roles and goal should be assigned to an agent. However, the agent then could act on their own according to their role.

The seventh team was *UCD Bogtrotters* formed by Sean Russell, Rem Collier and Mauro Dragone from University College Dublin, Ireland[**?**]. Their solution architecture consisted of a custom behavioural layer for the implementation of primitive capabilities and *Agent Factory (AF)* as the upper layer. Together with a Teleo-Reactive programming language *AF-TR* available as part of the agent Factory platform the developers implemented a strategy agent that is responsible for coordination by receiving all perceptions, organizing agents into teams and allocating task to individual agents. The agents' low-level behaviours (explore, open fence, etc.) were written in Java.

The eighth and final team was *USP Farmers* formed by Gustavo P. Gouveia, Ricardo H. Pereira, Jaime S. Sichman from University of Sao Paulo, Brazil[**?**]. 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 which plans should be selected and performed by the agents, and 3) artifacts (*CArtAgO*) to externalize actions. Roles were not assigned by a coordinator, instead the agents asked for a role and assigned roles to other agents when needed.

6.2 Results

The *Brainbug* team, the winner of the Multi-Agent Programming Contest 2007, 2008 and 2009, won the tournament again. While in 2009 the gap was quite small the *Brainbug* team managed to increase the gap again to 9 points.

The following overview shows how many points each team scored.

1. Brainbug (57 Points)
2. CowRaiders (48 Points)
3. UCDBogtrotters (46 Points)
4. Galoan (36 Points)
5. Argonauts (29 Points)
6. Jason-DTU (20 Points)
7. PauLo (11 Points)
8. USPFarmers (1 Point)

The *Brainbug* team was dominating the Contest while *CowRaiders* and *UCD-Bogtrotters* performed well but were not able to keep track of the first team. Then the other teams follow with big gaps between them.

7 Conclusion: Experiences and Future Outlook

The initial idea for starting Multi-Agent Programming Contest series 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 very carefully in order to enable more objective evaluation and comparison of the multi-agent programming languages, development tools, techniques 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 its impact to the wider multi-agent programming research community. Nonetheless we have noticed that various prominent research groups in this area are enthusiastic about the contest and participated in various contest editions. After each edition, they provided us with their experiences, feedbacks and suggestions which we used to improve the next edition of the contest. Several participating groups 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.

Besides 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, our 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 motivates 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 some of the participating teams).

In a way, the history and the evolution of the Multi-Agent Programming Contest tournament series follows the maturing of the research community interested in problems of design

and implementation of programming tools and techniques for cognitive BDI agents. Even though initially associated with the CLIMA workshop series, it was mostly the members of the research community around the ProMAS workshop series, who actively participated on the contest and contributed to its further evolution in numerous discussions.

While a lot of effort in this community was dedicated in the past to issues related to development of single-agent systems, gradually we witnessed a shift towards practical problems in multi-agent system, such as cooperation, coordination, negotiation, etc. As the recapitulation of the Contest history in Section 2 highlights, the switch to the cows and cowboys scenario also demonstrates the gradual shift of focus of the contest from scenarios more appropriate for benchmarking of behaviours of individual agents to *enforcing cooperation and coordination in the teams of agents*. Our ambition is to continue in this trend and we are actively discussing possibilities for adaptation and re-design of the future simulation scenarios to involve more features such as teamwork, coordination, cooperation, etc.

While we had the ambition to focus on benchmarking solutions to problems and issues of multi-agent systems such as coordination, cooperation and teamwork early on, it turned out that our belief that coordination in agent teams is beneficial even in simple scenarios, such as the gold-mining game, was wrong. In fact, unless the scenario enforces some kind of cooperative behaviour between agents, (1) it is much more straightforward to come up with solutions involving uncooperative agents and (2) the actual gains in performance of coordinating agent teams did not support the costs and overheads induced by implementation of coordination mechanisms of autonomous agents beyond information sharing.

This observation makes us believe that if we want to see implemented cooperative behaviours, the simulation scenarios should *enforce* coordination techniques, rather than simply *encourage* it. This feature is highlighted in the cows and cowboys scenario which is designed so that success of individual agent-based strategies is limited and sometimes even impossible. A single agent cannot "push" a herd of cows, they would simply disperse in the grid, nor can an agent navigate through the environment on its own because fences have to be actively kept opened by another agent.

As already said above, in the future, we are planning to further improve and develop new tournament scenarios. There are two main vectors along which these developments should proceed.

Firstly, as already mentioned above, we seek to enrich the tournament scenarios with features enforcing and encouraging non-trivial coordination in agent teams. From our experience we realized that scenarios enforcing coordination must involve active non-trivial autonomous entities as a part of the simulated environment featuring complex behaviours opaque to the agents.

Moreover, in order for coordination among the agent team members to yield a substantial benefit, the environment has to behave so that a single agent not-only can succeed, but rather individual actions should be even penalized. Only then, we will be able to receive submissions fairly comparable w.r.t. implemented coordination mechanisms in the agent teams. In particular, we are looking at drawing inspiration from modern strategic computer games. Alternatively, being inspired by approaches such as ORTS, we are also studying a possibility to introduce *adversarial elements* into simulation scenarios. For example, we aim to allow the agent teams to play against each other in a more direct fashion. Up to now, the teams were only competing for a constant amount of otherwise neutral resources.

Secondly, in the past we also realized that one of the factors drawing attention to the Multi-Agent Programming Contest, in particular the attention of students in multi agent systems courses, is the visual and playful attractiveness of the simulation scenarios. We believe that this is a factor which should not be underestimated, especially when the *MASSim*

platform is embedded in teaching. Along this axis, we are exploring several ideas, such as pushing the contest closer to visual attractiveness of computer games by integration with some open game platforms. Alternatively, we are looking into possibilities on how the tournament scenarios could be closer to real-world applications by exploiting already existing technologies in high-fidelity simulations of real-world problems, such as urban, or air-traffic simulations or physical simulations for multi-robot systems.

Finally, what we have not seriously addressed until now is the *autonomy* of an agent. As it turned out, some teams in the last contests used a kind of centralized approach: The agents were steered by the system and the information that the single agents collect has been distributed among all the agents through the agent platform. Thus the agents were not truly autonomous.

Acknowledgements The work of Peter Novák was partially supported by *The Ministry of Education, Youth and Sports* of the *Czech Republic* through its Research Program MSM6840770038.

The work of Jürgen Dix and Michael Köster was funded by the NTH School for IT Ecosystems. NTH (Niedersächsische Technische Hochschule) is a joint university consisting of Technische Universität Braunschweig, Technische Universität Clausthal, and Leibniz Universität Hannover.

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. 6):

```
<?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. 6). 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.

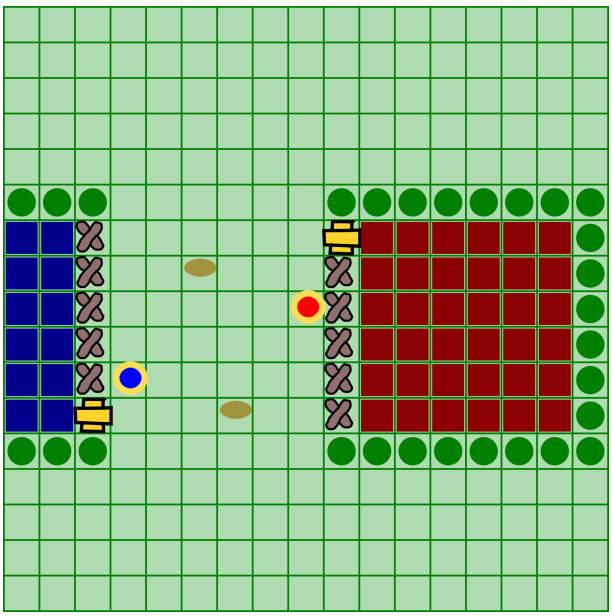


Fig. 6 The view range of the agents. The agent in the center perceives all depicted cells.

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`.

B System Description Template

Preface

In order to compare the systems properly, we asked the participants to follow this template. Additionally, each system description ends with a short compilation of the answers to the questions described hereinafter.

B.1 Introduction

1. Explain the background of your multi-agent system development activities. Are you developing or working on a specific multi-agent system platform in your research group? What are the state of art of your relevant research?
2. What were your motivations for joining this contest? What were your objectives that you aimed to achieve in participating the contest?
3. Explain briefly the preparation and set up of your participation in the contest. Which machines and infrastructures did you use? Which software did you use?

B.2 System Analysis and Specification

1. Describe the requirements that you used to develop you multi-agent system. Please indicate and explain if you have used a specific requirement analysis approach.
2. How is your multi-agent system specified? Please provide a detailed description of the specification of 1) individual agents, 2) the interaction between agents (or their organization), and 3) the interaction between agents and the simulation environment? If available, please provide formal or semi-formal specifications.
3. Did you use any existing multi-agent system methodology such as Prometheus, Gaia or Tropos? If so, please provide a detailed description of generated specifications. If not, explain how did you have specified your system?
4. How are the following features specified: *autonomy*, *role*, *proactiveness* and *communication*, *team working*, and *coordination*?
5. Is your system a truly multi-agent system or rather a centralised system in disguise? It is important to explain if you use a centralized or distributed coordination mechanism.

B.3 System Design and Architecture

1. How is your multi-agent system designed? Please provide a detailed description on the design and architecture of 1) individual agents, 2) the interaction between agents (or their organization), and 3) the interaction between agents and the simulation environment?
2. Did you use any existing multi-agent system design methodology to define your system architecture? If so, please provide a detailed description of generated design documents. If not, explain the architecture of your system?
3. How are the following features reflected in your system architecture: *autonomy*, *role*, *proactiveness* and *communication*, *team working*, and *coordination*?

B.4 Programming Language and Execution Platform

1. Explain the technology you used to develop your multi-agent system. Which programming language and development environment did you use to implement your multi-agent system?
2. Provide an overview of the characteristic syntactic constructs (of the employed programming language) that facilitates the implementation of multi-agent issues such as individual agents, agents' organization and coordination structure, and their interaction with the simulation environment. If possible, you can present the syntax of the used programming language. Explain the semantics of these constructs informally.

3. How is the designed architecture (both multi-agent and individual agent architectures) are mapped to programming codes, i.e., how are specific agent-oriented concepts and designed artifacts implemented?
4. Provide a description of the general structure of your programming code. For example, which packages, classes, objects, agents, or artifacts did you use?
5. Which strategies and algorithms did you use in developing different functionalities of individual agents and their coordination mechanisms?

B.5 Agent team strategy

1. Describe the navigation algorithms, e.g.,:
   - obstacle avoiding,
   - agent navigation,
   - strategy for finding and herding cows
   - opponent blocking,
   - robbing opponents.
2. Describe the team coordination strategy (if any).
3. Does your team strategy use some distributed optimization technique w.r.t. e.g. minimizing distances walked by the agents?
4. Describe and discuss the information exchanged (and shared) in the agent team.
5. Describe the communication strategy in the agent team. Can you estimate the communication complexity in your approach?
6. How could the overall strategy of your team be improved in the future based on your experiences during the contest?

B.6 Technical Details

1. Did your system do some background processing? Under background processing we understand some computation which happened while agents of the team were *idle*, i.e. between sending an action message to the simulation server and receiving a perception message for the subsequent simulation step.
2. If possible, please discuss additional technical details of your system like e.g. failure/crash recovery and alike.
3. How stable was your system in retrospective? How could you improve the stability in the future?

B.7 Discussion and Conclusion

In this section please expand a bit on your experience with the contest organization, the proposed scenario, and the setup of the actual contest. Please indicate what do you see as pros/cons of participating in the Contest with respect to your research in the field.

1. Critical discussion of your approach to the development of the agent team.
2. Did you gain some insights/experiences into developing multi-agent system in the course of participating in the Agent Contest so far? If so, what kind of insights?
3. Describe and discuss possible problems that you face in choosing approach, programming platform or technical infrastructure to participate in this contest?
4. How could the scenario be extended and which insights/results do you expect from these extensions?