Agent Contest Competition
3rd edition

Mehdi Dastani[1], Jürgen Dix[2] and Peter Novák[2]

[1]Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
mehdi@cs.uu.nl
[2]Clausthal University of Technology
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
{dix|peter.novak}@tu-clausthal.de

Abstract. This paper summarises the Agent Contest 2007 which was organized in association with ProMAS'07. The aim of this contest is to stimulate research in the area of multi-agent systems by identifying key problems and collecting suitable benchmarks that can serve as milestones for evaluating new tools, models, and techniques to develop multi-agent systems. The first two editions of this contest were organized in association with CLIMA conference series. Based on the experiences from the previous two editions ([8,9]), the contest scenario has been slightly extended to test the participating multi-agent systems on their abilities to *coordinate*, *cooperate*, and their *team work* and *team strategy* issues in a dynamic environment where teams compete for the same resources. Six groups from Germany, Brazil, England, Australia and The Netherlands did participate in this contest. The actual contest took place prior to the ProMAS'07 workshop and the winner, a group from the technical university of Berlin, was announced during ProMAS'07.

1 Introduction

Multi-agent systems are beginning to play an important role in today's software development. In the field of agent-oriented software engineering, various multi-agent system development methodologies have been proposed. Each methodology focuses on specific stages of the multi-agent system development. For example, Gaia [12] and Prometheus [10] focus on the specification and design stages assuming that other stages such as requirement and implementation are similar to corresponding stages of other software development paradigms. Therefore, software developers using Gaia and Prometheus propose models to specify and design multi-agent systems, while ignoring the implementation models.

Moreover, there is a growing number of agent-oriented programming languages and development platforms that are proposed to facilitate the implementation of multi-agent systems. These programming languages and platforms introduce programming constructs that can facilitate efficient and effective implementation and execution of multi-agent systems. The development of multi-agent

systems requires efficient and effective solutions for different problems which can be divided into three classes: 1) the problems related to the development of individual agents, 2) the problems related to the development of coordination and cooperation mechanisms to manage the interactions between individual agents, and 3) the problems related to the development of the shared environment in which agents perform their actions.

Typical problems related to individual agents are how to specify, design and implement issues such as *autonomy*, *pro-active/reactive behaviour*, *perception and update of information*, *reasoning and deliberation*, and *planning*. Typical problems related to the interaction of individual agents are how to specify, design and implement issues such as *communication*, *coordination*, *cooperation* and *negotiation*. Finally, typical problems related to the development of their environment are how to specify, design and implement issues such as *resources and services*, agents' access to *resources*, active and passive *sensing* of the environment, and realizing the *effects of actions*.
This competition is an attempt to stimulate research in the area of multi-agent systems by

1. *identifying key problems in developing multi-agent systems, and*
2. *evaluating state-of-the-art tools, models, and techniques in the field of multi-agent systems.*

While there already exist several competitions in various areas of artificial intelligence (theorem proving, planning, Robo-Cup, etc.) and, lately, also in specialized areas in agent systems (Trading Agent Competition (TAC) [1] and AgentCities competitions [2]), the emphasis of this contest is on the use of existing tools, models, and techniques that are proposed to develop multi-agent systems ([3,7,4]. In particular, we aim at evaluating existing approaches for the development of multi-agent systems where individual agents has to cooperate with each other to solve a task. In this respect, issues such as team working, team strategy, interaction with dynamic environment, modeling the environment, limited perception, uncertain action effects, reasoning and planning, and learning are essential.

The previous editions of this contest were organized in cooperation with CLIMA workshop series. The scenario from this year can be seen as an extension of the scenario from the CLIMA VII Contest 2006. The main differences include:

- perception now includes also the information about the number of gold items an agent carries,
- number of agents per team is 6, instead of 4 last year,
- agents can push each other,
- agents can collect and carry more gold items,
- each agent has to connect to the server from a separate IP address (this requirement might be a subject of change).

We believe that these extensions lead to a greater competitiveness of the scenario (the fun factor should not be underestimated) and put the participating multi-agent systems under a test w.r.t. coordination and cooperation issues in an environment where teams compete for the same resources.

2 Scenario Description

The competition task consisted of developing a multi-agent system to solve a cooperative task in a dynamically changing environment. The environment of the multi-agent system was a grid-like world where agents could move from one cell to a neighbouring cell. In this environment, gold could appear in the cells. Participating agent teams were expected to explore the environment, avoid obstacles and compete with another agent team for the gold. The agents of each team could coordinate their actions in order to collect as much gold as they could and to deliver it to the depot where the gold can be safely stored. Agents had only a local view on their environment, their perceptions could be incomplete, and their actions could fail. The agents were able to play different roles (such as explorer or collector), communicate and cooperate in order to find and collect gold in an efficient and effective way.

The idea was to divide participating agent teams randomly into groups before the tournament started. Each team from one group should then compete against all other teams in the same group in a series of matches. The winners from these groups should form a new group and each team in a new group should play against each other again. In the case of few participating teams, we had planned to form only one single group consisting of all teams. Because of the number of participants, we decided to form only one group for this edition of the agent contest. Each team competed against all other teams in a series of matches. Each match between two competing teams consisted of five simulations. A simulation between two teams was a competition between them with respect to a certain starting configuration of the environment. Winning a simulation yielded three points for the team, a draw was worth one point and a loss resulted in zero points. The winner of the whole tournament was evaluated on the basis of the overall number of collected points in the matches during the tournament. In the case of equal number of points, the winner should be decided on the basis of the absolute number of collected gold items. Details on the number of simulations per match and the exact structure of the competition was published prior to the Contest on the official Agent Contest 2007 website at `http://cig.in.tu-clausthal.de/AgentContest2007/`.

2.1 Technical Description of the Scenario

In the contest, the agents from each participating team were executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams performed actions, was run on the remote contest simulation server run by the contest organizers. The interaction/communication between agents from one team were managed locally, but the interaction between individual agents and their environment (run on the simulation server) took place via Internet. Participating agents were connected to the simulation server that did provide the information about the environment. Each agent from each team connected to and communicated with the simulation server using TCP protocol and messages in XML format.

During the initial phase[1] agents from all competing teams connected to the simulation server, identified and authenticated themselves and got general match information. Each agent had to connect to the simulation server from a separate IP address. Teams not obeying this rule would have been disqualified and disconnected from the simulation server during the tournament. At the announced start time of the tournament, the simulation server was on-line and the agents from participating teams were able to connect to it. After a successful initial handshake during which agents identified themselves by their IDs and received acknowledgment from the server, they waited for the simulation start. The initial connecting phase took a reasonable amount of time in order to allow agents to be initialised and connected (15 minutes).

The simulation server controlled the competition by selecting the competing teams and managing the matches and simulations. In each simulation, the simulation server, in a cyclic fashion, provided sensory information about the environment to the participating agents and expected their reactions within a given time limit. Each agent reacted to the received sensory information by indicating which action (including the 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 on their environment, their perceptions could be incomplete, and their actions could fail. After a finite number of steps the simulation server stopped the cycle and participating agents received a notification about the end of a simulation. Then the server started a new simulation possibly involving the same teams.

2.2 Team, Match, and Simulation

An agent team consisted of six software agents with distinct IDs. There were no restrictions on the implementation of agents, although we encouraged the use of approaches based on the state-of-the-art tools, methodologies and languages for programming agents and multi-agent systems, as well as the use of computational logic based approaches. The tournament consisted of a number of matches. A match was a sequence of simulations during which two teams of agents competed in several different settings of the environment. For each match, the server 1) picked two teams to play it and, subsequently, 2) started the first simulation of the match. Each simulation in a match started by notifying the agents from the participating teams and sending them the details of the simulation. These included for example the size of the grid, the depot position, the number of steps the simulation will perform, etc. A simulation consisted of a number of simulation steps. Each step consisted of 1) sending a sensory information to agents (one or more) and 2) waiting for their actions. In the case that an agent did not respond within a timeout (specified at the beginning of the simulation) by a valid action, it was considered to perform the skip action in the given simulation step.

[1] The contest organizers contacted participants before the actual tournament and provided them the IDs necessary for identification of their agents for the tournament.

2.3 Environment objects

The (simulated) environment was a rectangular grid consisting of cells. The size of the grid was specified at the start of each simulation and was variable. However, it was always at most 100x100 cells. The [0,0] coordinate of the grid was in the top-left corner (north-west). The simulated environment contained one depot, which served for both teams as a location of delivery of gold items. The environment did contain the following objects in its cells:

- an obstacle (a cell with an obstacle cannot be visited by an agent),
- gold (an item which can be picked from a cell) agent,
- an agent,
- the depot (a cell to which gold items are to be delivered in order to earn a point in a simulation),
- a marker (a string data with a maximum of 5 characters which can be read/written/rewritten/removed by an agent).

There could be only one object in a cell, except that an agent could enter cells containing gold, depot or mark. A gold item could be in a marked cell visited by an agent. At the beginning of a simulation the grid contained obstacles, gold items and agents of both teams. Distribution of obstacles, gold items and initial positions of agents was either hand crafted for the particular scenario, or completely random. During the simulation, gold items were appearing randomly in empty cells of the grid. The frequency and probability of gold generation was simulation specific, however not known to neither agents, nor participants. At the start of each simulation agents got the details of the environment (grid size, depot position, etc.). Agents also received information about their initial position in the perception information of the first simulation step.

Perception Agents were located in the grid and the simulation server provided each agent with the following information:

- the absolute position of the agent in the grid,
- the content of the cells surrounding the agent and the content of the cell in which the agent currently stands in (9 cells in total),
- the number of gold items the agent currently holds.

If two agents were standing in each other's field of view, they were able to recognize whether they are enemies, or whether they belong to the same team. However an agent was not able to recognise whether the other agent carries a gold item or not. If there was a mark in a cell, within the agent's field of view, the agent also received the information about its content.

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

- **skip:** The execution of the skip action had no influence on the local state of the environment around the agent (under the assumption that other agents did not change it). When an agent did not respond to a perception information provided by the simulation server within the given time limit, the agent was considered as performing the skip action.
- **movements (right, up, left, down):** An agent could move in four directions in the grid. These movement actions were specified as follows. The execution of move actions up, down, left and right changes the position of the agent one cell to the up, down, left, and right, respectively. A movement action succeeds only when the cell to which an agent is about to move does not contain an obstacle. In the case two agents stand in adjacent cells and one of them tries to step into the cell the second agent stands in while the second agent performs e.g. skip action, the second agent can be pushed away. The resulting local change of the environment amounts to the same situation as if the pushed agent performed a move action in the same direction as the pushing agent. The same constraints as for regular move actions apply, i.e. there cannot be another obstacle, or an agent standing in the way of the pushed agent. Only one agent can be pushed in one move. In the case both agents standing in the adjacent cells try to push each other, one of them will be randomly determined (with probability of 50%) as the pushing and the other as the pushed agent. A detailed specification of the action execution algorithm later in this paper describes further details of push action and its consequences. Moving to and from the depot cell were regulated by additional rules described later in this description.
- **pick, drop:** An agent could carry up to maximum of three gold items which it successfully picked up before. An agent could pick up a gold item if 1) the cell in which the agent stands in contains gold, and 2) the agent carries less than 3 gold items. An agent could drop gold item it carried only into the empty cell it stood in. The result of a successful pick action is that in the next simulation step the acting agent will be considered to carry one more gold item than before performing the pick action and the cell, it stands in, will not contain the gold item any more. The result of a drop action is that the acting agent is carrying one gold item less than before performing the drop action (given that the agent was carrying at least one gold item in that simulation step) and that the cell it stands in will contain the gold item in the next simulation step. Drop action performed in the depot cell results in dropping all the gold items the agent carries at once and increases the score of the agent's team by a number of points equal to the number of gold items the agent dropped in the depot cell. The depot cell never contains a gold item that can be picked by an agent.
- **mark, unmark:** An agent was allowed to mark a cell it stood in by a string data with a maximum of 5 characters. The result of a mark action is that the cell in which an agent is located, will contain a string in the next simulation step. The depot cell, and cells containing an obstacle cannot be marked. By marking a previously marked cell, the old mark is removed and replaced by the new one. If the cell in which an agent is located, contains a mark,

then the agent receives the string in the perception information from the simulation server. An agent was allowed to unmark the marked cell it stood in. The result of an unmark action is that the cell will not contain a mark in the next simulation step. Agents do not get immediate feedback on their actions, but can learn about the effects of their actions (and the actions of other agents) from the perception information that will be sent to them in the next simulation step.

Action Execution Algorithm After the simulation engine collected the actions that the agents chose to execute in the next simulation step (or the simulation step timeout for agent's reaction elapsed), the next state of the environment was determined as follows:

1. all the agents' impossible actions were replaced by skip actions. An impossible action is:
   - the move action when the agent tries to step into an obstacle, or out of the grid boundary, or
   - the drop action when the cell already contains gold, or
   - the pick action when there's no gold contained in the cell, or
   - the unmark action when the cell does not contain a mark;
2. the simulation engine determined actions which will fail because of Fatigue (see description below) and replaces them with a skip action;
3. for each cell not containing an agent, or an obstacle, such that there's at least one agent indicating an intention to move into it, one of these agents was selected and moved to this cell. Actions of all the other considered agents were replaced with a skip action;
4. for each agent which can be pushed by more than one pushing agent (an agent can be pushed iff it is about to perform a skip action [after applying steps 1-3], the cell it is going to be pushed into is within the grid boundary and does not contain an agent, or an obstacle), one such pushing agent was selected, and both pushed and pushing agents were moved in the direction of the move of the pushing agent;
5. all other move actions which were not executed in steps 3 and 4 were replaced by skip action;
6. all the non-move actions were executed;
7. further internal changes and calculations of the environment, like e.g. gold generation, took place.

Depot cell Strong conditions were imposed on the depot cell:

1. an agent not carrying a gold item was unable to enter the depot cell (the result of such an action is the same as if the depot was an obstacle);
2. agent which entered the depot cell should drop the gold item as the very next action it executed;
3. after dropping the gold item in a cell, an agent had to leave the cell in the first subsequent simulation step when it was able to move (i.e. when there was an empty cell at the time of agent's move action).

If an agent did not leave the depot in the first possible opportunity, or did not drop the gold item as the very next action after entering the depot, the simulation server punished it by "teleporting" it away (it was moved to a random cell not containing another agent, or obstacle in the grid by the environment simulator).

Timeout The agents had to inform the simulation server which action they wanted to perform within a timeout specified at the beginning of the simulation. Timeouts were set reasonably high, so that even participants with a slow network connection and complex deliberation algorithms were able to communicate with the server in an efficient way. Simulation timeouts were not lower than two seconds and higher than 10 seconds per one simulation step.

A ping interface was provided by the server in order to allow participating agents to test the speed of their connection during the initial phase of the tournament. Note, that only a limited number of ping requests were processed from one agent in a certain time interval.

Fatigue (Information Distortion/Action Failure) Agents received incomplete information about the environment from the simulation server. The simulation server could omit information about particular environment cells, however, the server never provided incorrect information. Also, agent's action could fail. In such a case the simulation server evaluated the agent's action in the simulation step as a skip action.

Both the probability of sending an agent incomplete information (P_{inf}) and the probability of agent's action failure (P_{fail}) were constant and specific for each simulation, however not higher than 20%. Moreover, both probabilities increase in a linear fashion with respect to the number of gold items currently carried by the agent up to at most 50%. The equation regulating this relation was as follows:

$$p = P_{sim} + \frac{P_{max} - P_{sim}}{N_{itMax}} \times N_{it}$$

Here, P stands for the actual probability of action failure, or information distortion w.r.t. number of items the agent currently carries, P_{sim} is the probability of action failure/information distortion set as default for the current simulation (it is equal to the corresponding probability when agent does not carry a gold item). P_{max} and N_{itMax} are the maximal value of failure/information distortion probability (at most 50%) and maximal number of gold items the agent is allowed to carry (3 as specified above) respectively. These values, together with P_{sim} (at most 20%) are parameters of each current simulation. Finally N_{it} stands for the number of gold items the agent currently carries.

Below we list examples of two simulation settings together with tables of resulting probabilities for agent carrying 0, 1, 2 and 3 gold items:

$P_{sim} = 10\%$	$P_{sim} = 5\%$
$P_{max} = 50\%$	$P_{max} = 40\%$
$N_{itMax} = 3$	$N_{itMax} = 3$

N_{it} - P	N_{it} - P
0 - 10.0%	0 - 5.0%
1 - 23.3%	1 - 16.6%
2 - 36.6%	2 - 28.3%
3 - 50.0%	3 - 40.0%

Simulation parameters P_{sim}, P_{max} are not known neither to agent team designers, nor to the agents during the simulation. As already mentioned above, N_{itMax} is a constant set to 3 for all simulations in the tournament.

Final Phase In the final phase, the simulation server sent a message to each agent allowing them to disconnect from the server. By this, the tournament was over.

2.4 General Agent-2-Server Communication Principles

In this contest, the agents from each participating team were executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams performed actions, was run on the remote contest simulation server. Agents communicated with the contest server using standard TCP/IP stack with socket session interface. The Internet coordinates (IP address and port) of the contest server (and a dedicated test server) were announced later via the official Contest mailing list. Agents communicated with the server by exchanging XML messages. Messages were well-formed XML documents, described later in this document. We recommended using standard XML parsers available for many programming languages for generation and processing of these XML messages.

Communication Protocol The tournament consisted of a number of matches. A match is a sequence of simulations during which two teams of agents compete in several different settings of the environment. However, from the agent's point of view, the tournament consisted of a number of simulations in different environment settings and against different opponents.

The tournament was divided into three phases. During the initial phase, agents connected to the simulation server and identified themselves by username and password (AUTH-REQUEST message). Credentials for each agent were distributed in advance via e-mail. As a response, agents received the result of their authentication request (AUTH-RESPONSE message) which either succeeded, or failed. After successful authentication, agents waited until the first simulation of the tournament started.

At the beginning of each simulation, agents of the two participating teams were notified (SIM-START message) and received simulation specific informa-

tion: simulation ID, opponent's ID, grid size, number of steps the simulation will last and the depot position.

In each simulation step an agent received a perception about its environment (REQUEST-ACTION message) and it responded by performing an action (ACTION message). Each request-action message contained information about nine neighboring cells around the agent (including the one agent stands on), its absolute position in the grid, simulation step number, number of gold items the agent carries and deadline for its response. The agent had to answer within the given deadline. The action message contained the identifier of the action, agent wants to perform, and action parameters, if required.

When the simulation was finished, participating agents received the notification about it (SIM-END message) which included the information about the number of gold items collected by the team agent belongs to and the information about the result of the simulation (whether the team won, or lost the simulation).

All agents which currently did not participate in a simulation had to wait until the simulation server notified 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 received the notification (BYE message). Subsequently the simulation server terminated the connection to the agent.

Reconnection When an agent lost connection to the simulation server, the tournament proceeded without disruption, only all the actions of the disconnected agent were considered to be empty (skip). Agents were responsible for maintaining the connection to the simulation server and in a case of connection disruption, they were allowed to reconnect.

An agent reconnected by performing the same sequence of steps as at the beginning of the tournament. After establishing the connection to the simulation server, it sent AUTH-REQUEST message and received AUTH-RESPONSE. After successful authentication, the server sent SIM-START message to an agent. If an agent participated in a currently running simulation, the SIM-START message was delivered immediately after AUTH-RESPONSE. Otherwise an agent had to wait until the next simulation in which it participates. In the next step when the agent was picked to perform an action, it received the standard REQUEST-ACTION message containing the perception of the agent at the current simulation step and simulation proceeded in a normal mode.

Ping Interface The simulation server provided a ping interface in order to allow agents to test their connection to the simulation server. An agent can send a PING message containing a payload data (ASCII string up to 100 characters) and it received a PONG message with the same payload. As all messages contained a timestamp (see description of the message envelope below), an agent could also use the ping interface to synchronize its local time with the server.

XML Messages Description XML messages exchanged between server and agents were zero terminated UTF-8 strings. Each XML message exchanged between the simulation server and agent consisted 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 was `<message>`. It has attributes: the timestamp and a message type identifier.
- Message separator: Each message is a UTF-8 zero terminated string. Messages are separated by null byte.

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 1, 1970 UTC). Message type identifier was 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 contained all attributes of the root element. However, the timestamp attribute could be omitted in messages sent from an agent to the server. In the case it was included, server silently ignored 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>
```

According to the message type, the root element `<message>` can contain simulation specific data. These simulation data are described and explained in the official contest webpage[2].

3 Submission

The participation in this contest consisted of two parts. Participants first submitted the description of analysis, design and implementation of a multi-agent system for the above application. Existing multi-agent system methodologies such as Gaia, Prometheus or Tropos can be used to describe the system. For the description of the implementation, it should be explained how the design is

[2] `http://cig.in.tu-clausthal.de/fileadmin/user_upload/_temp_/ac07-protocol.txt`

implemented. This can be done by explaining, for example, which programming language, platform, tools, and techniques are used to implement the multi-agent system. These submissions are included in this volume.

The second part of the contest is the actual participation in the tournament by means of an (executable) implementation of a multi-agent system. The agents from each participating systems (agent teams) were executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, was run on the remote contest simulation server. Interaction/communication between agents from one team has been managed locally, but the interaction between individual agents and their environment (run on the simulation server) was via Internet. Participating agents connected to the simulation server that provided the information about the environment. Each agent from each team connected and communicated to the simulation server using a TCP connection.

3.1 Received Submissions

We have received seven submissions for this edition of the contest from which one withdrew just before the start of the actual contest tournament. From the received submissions, which are included in this volume, three submissions used existing multi-agent development methodologies to specify and design their multi-agent systems. Other submissions used their own developed agent platforms and corresponding customised development methodologies. The withdrawn submission intended to use GoLog, a knowledge representation language based on logic. Unfortunately, one competitor in the contest could not provide a description of their team in this volume.

The submission by J.F. Hübner and R.H. Bordini was a collaboration between Durham University, UK, and Universidade Regional de Blumenau, Brazil. Like their submission to the previous edition of this contest, they use Prometheus [10] as the multi-agent system development methodology to specify and design their multi-agent system. Using this methodology, the multi-agent system is designed by means of a system overview Diagram that describes the interaction between miner and leader agents. Miners are the agents that interact with the contest simulator and the leader helps the coordination of some activities. These agents are subsequently specified and designed in terms agent overview diagrams describing their specific knowledge, goals and plans. Their designed system is then implemented in Jason [5], which is an interpreter of an extension of the agent programming language AgentSpeak [11]. As it was required by the contest, their multi-agent system consisted of six miner agents operating in the simulated environment. These agents follow a general strategy according to which each agent is responsible for one quadrant of the grid environment. The leader helps the miners to coordinate themselves in two ways. First, it allocates miners to quadrants, and second it coordinate the negotiation process that is started when a miner sees a piece of gold and is not able to collect it (because its container is full).

The second submission that uses existing multi-agent system development methodologies to specify and design their system is by L. Astefanoaei, C.P. Mol, M.P. Sindlar, and N.A.M. Tinnemeier from Utrecht University, Netherlands. They use a combination of Tropos and Moise$^+$ methodologies. They use Tropos to specify the multi-agent system in terms of system goal and subgoals and Moise$^+$ to specify the roles the agent cap play and the interaction between roles. In their system specification, an agent can play three different roles: leader, scout, and miner. Moreover, their system can have at most one leader, and zero to six players that can play the scout or miner role. The leader communicates with the scouts and the miners. The leader coordinate the behavior of scouts and miners by means of task ordering: first the scouts explore the wilderness, then the miners can gold-enrich the team. They use the 2APL programming language and its corresponding multi-agent platform to implement and execute their multi-agent system.

The third submission is by E. Tuguldur and M. Patzlaff from the DAI-Labor, Technische Universität Berlin, Germany. They develop a multi-agent system based on microJIAC agent definition and its corresponding Maven plug-in that supports the compilation and packaging process of agents. According to this definition, an agent consists of three components: connector, perceptor, and monitor. The connector maintains the connection to the contest server, the perceptor updates the agent's world model, and the monitor which provides a graphical user interface to display the world model of the agent (mainly for debug purposes). The microJIAC is a lightweight agent architecture targeted at devices with different capabilities. Each agent can be in either explorer or transporter mode. An agent in the explorer mode aims at collecting gold items up to the maximum amount of gold items that it can carry. When an agent changes its role to transporter mode, it aims at reaching the depot to drop all gold items. After dropping its gold items the transporter agent becomes an explorer again.

The fourth submission was by A. Hessler, B. Hirsch, and J. Keiser, also from the DAI-Labor, Technische Universität Berlin, Germany. They used the JIAC IV (Java Intelligent Agent Componentware) methodology and its corresponding framework to develop their multi-agent system participating in the agent contest. The JIAC methodology is based on the JIAC meta-model that has explicit notions of goal, rule, plan, service and protocol. The JIAC development process starts with collecting, structuring and prioritising domain vocabulary and requirements. Based on the requirements with the highest priority a multi-agent system architecture is designed by listing the agents. Plans, services and protocols are then implemented and plugged into agents. The application is then evaluated and, if necessary, the cycle is started until the desired quality of the multi-agent system is achieved. In their multi-agent system implementation agents cooperate by sharing their perceptions, states, and intentions as they may go for the same unknown field or to pick the same gold items. In their approach agent communication and cooperation is fully decentralised. There is neither a message broker nor a central instance which coordinates agents. Every agent builds its own world model from what it is told by the server and the other

agents. Every agent also plans for itself, taking the states and intentions of other agents into account.

The fifth system, by Sebastian Sardina and Dave Scerri, from RMIT University, Australia, was mostly designed using the Prometheus [10] multi-agent system development methodology and implemented in the JACK BDI agent-oriented programming language [6] using its JDE development environment. In this system, there are two type of agents: player agents and one coordinator agent. The player agents are the ones that are able to interact with the game simulator; whereas the coordinator agent acts as an (information) proxy among the player agents, and instructs the players on some activities. At any point in time, a player agent can play either a "collector" role or an "explorer" role. These roles are assigned by the coordinator agent. As a collector, a player agent's main objective is to collect gold pieces and bring them to the depot location. In contrast, as an explorer, the objective of a player is to gather information about unknown areas of the world, and communicate such information to the coordinator. Collector players also have a set of quantitative parameters that influence the way it would behave. For example, an "exploration attitude" parameter determines how bias a collector agent is towards exploration. In that way, a collector can be bias to explore (or to avoid exploring) unknown areas of the grid while traveling to the depot location for gold deposition. Unfortunately, the JACK system was not able to sustain its participation throughout the whole contest, as the system communication infrastructure was not sufficiently robust and, as a result, the agents would very often lose communication with the contest simulator.

The final submission is by S. Schiffel, M. Thielscher, and D. Thu Trang from Dresden University of Technology, Germany. Like their submission in the previous edition of this contest, they do not use any specific multi-agent system development methodology. Instead, they use FLUX agents to design and implement their multi-agent system. Each FLUX agent is a logic program consisting of three modules: the fundamental reasoning facilities based on the fluent calculus, the specification of the effects of actions, and the strategy. Their implemented multi-agent system consists of six agents and a leader. The leader coordinates the behaviors of other agents by helping them to share their information about the environment. The action of an agent depends on the current intentions of that agent and the current state of the world. After an agent decides on its next action it sends its new information and its current intention to the leader. The leader sends information gathered by the other agents to the agent and might request the agent to change its intentions for coordinating the agents of the team. Conflicts between agents are resolved in two ways. First, the leader assigns areas to the agents for exploration. Second, small conflicts such as when several agents try to get into the same cell, are resolved using fixed priorities of the agents.

4 Technical infrastructure

In the third edition of this Agent Contest, we re-used the technical infrastructure we developed for the second edition. Briefly, the server's architecture consists of

1. *simulation plug-in*: A replaceable module providing the logics of the environment simulation,
2. *agent session manager*: Responsible for holding the sessions between the server and individual agents and en/de-coding of XML messages of the protocol,
3. *visualization library*: It produced the SVG records from each time frame of the simulation environment state,
4. *contest webinterface*: Providing a public view and interface to the MASSim server, and
5. MASSim core module: Managing the tournament scheme and providing the connection between the simulation plug-in, agent session manager and web-interface.

A more detailed description of the system can be found in the report on the second edition of the Agent Contest [9]. The system is published on the official Contest website: `http://cig.in.tu-clausthal.de/AgentContest/`.

4.1 Contest preparation

As in previous editions, before the tournament itself, the Contest organization went through several preparatory stages. We released the communication protocol for the 2007 Contest simulation scenario in February 2nd 2007 together with a template for system description submissions. The first protocol release contained a requirement that each agent has to run from a distinct IP address, however after a discussion with potential participants, we dropped this requirement later (February 23rd). The main reason was the variety of network infrastructures on the participants' side like e.g. NAT and various other IP masquerading technologies which render this requirement not enforceable.

Shortly before the system description submission deadline on March 10th 2007, we published the first release of the testing suite on March 6th, which was later followed by a precise description of the algorithm for calculating agent movements regarding various configurations of situations when agents push each other. The testing suite contained a testing version of the MASSim server configured to run the 2007 Contest simulation, together with a simple debugging tool (MASSim Debug Monitor) and vanilla agents compatible with the Contest scenario.

The Agent Contest 2007 testing phase was launched on March 27th 2007 and ran until the very Contest tournament launch on May 2nd 2007. During this period, which lasted more than one month, the participants could freely connect to the testing server and test their agents in a simulated match against our dummy *Bot* agent team. We did not allow different teams to compete against

each other as this should happen only during the tournament itself. During the testing phase, few minor bugs in the scenario implementation were discovered and quickly fixed.

4.2 Tournament

The Agent Contest 2007 tournament itself was launched on Wednesday, May 2nd 2007 at 15:00 CEST (UTC/GMT+2). A few days in advance, the participants received the Internet coordinates of the tournament server together with credentials for their agents. The Contest was served on the tournament server `agentmaster.in.tu-clausthal.de` and it could be observed via a web-interface at the address `http://agentmaster.in.tu-clausthal.de/`. We provided also a chat space for participants, what in the course of the tournament itself turned out to be a vital and efficient communication tool.

The teams competed sequentially against each other so that the order of teams was fixed (decided randomly at the beginning of the tournament) and then 1st played against the 2nd, then 3rd, 4th, etc. and finally against the last in the row. Then the 2nd team played against the 3rd, 4th, etc. The participation order was:

1. microJiacteam,
2. FLUXteam,
3. JiacIVteam,
4. AC07bot,
5. JACKteam,
6. APLteam,
7. GOLOGteam,
8. Jasonteam.

Unfortunately, this approach caused the last team in a row (Jasonteam) to compete only at the end of every "cycle", which was also a reason for complaints raised by this team during the tournament.

The tournament itself ran for several days and officially finished only on Monday, May 7th 2007 in the early morning. However, its execution was disrupted by a simulation server failure on Saturday, May 5th in the late evening. The failure lasted for several hours and in the early morning on Sunday May 6th, the tournament was restarted and the remaining simulations were run to the end.

During the tournament, on Friday May 4th, because of technical and performance difficulties, the GOLOG team decided to withdraw from the tournament. The team was disconnected and to keep the tournament run consistently, replaced with a dummy bot team.

The tournament lasted for approximately 4 and a half days. The long tournament execution time was caused by the setup of the simulation scenarios and our own desire not to handicap deliberating approaches.

For illustration, for 8 participating teams, and 5 simulations, we get $7 \times 8 = 56$ matches, i.e. $56 \times 5 = 280$ simulations. Simulations had approximately 800 steps.

Provided a timeout of approximately 4 seconds per simulation step, in the worst case (when each team fully uses the timeout for deliberation), we could have a tournament running for 248 hours, i.e. approximately 10 days. Therefore, in the next editions of the Contest we plan to approach this issue by a parallel execution of several simulations simultaneously.

All results, together with the SVG recordings of all the matches and the official DVD ISO image with a mirror-copy of the whole tournament website can be downloaded from `http://agentmaster.in.tu-clausthal.de/`.

4.3 Simulation instances

The teams competed in matches each consisting of 5 different grid simulations with identifiers *Park*, *Meadow*, *Semiramis*, *Fence* and *Overkill* (Figure 1). The first two simulation scenarios *Park* and *Meadow* are randomly generated and differ only in the amount of gold items and trees. While the first features more trees and sparse gold, *Meadow* is configured to feature the opposite. Scenarios *Semiramis* and *Fence* are handcrafted labyrinths to challenge agent teams obstacle avoiding and communication approaches. Finally *Overkill* is a variation on the most difficult maze from the previous Contest 2006. The details of configuration properties of the scenarios is listed in Table 1.

simulation ID:	*Park*	*Meadow*	*Semiramis*	*Fence*	*Overkill*
grid size:	51x51	40x40	40x40	51x51	30x30
depot position:	random	random	(19,34)	(29,34)	(20,20)
number of obstacles:	250	95	175	235	76
initial number of gold items:	100	175	85	155	66
information distortion probability:	10%	10%	20%	5%	5%
action failure probability:	10-25%	5-33%	10-50%	10-50%	5-33%
gold generation frequency:	20 steps	10 steps	20 steps	20 steps	30 steps
number of generated gold items:	2	3	4	3	5
number of simulation steps:	1000	800	800	1000	700

Table 1. Simulation scenario configurations

5 Contest results

The winner of the ProMAS'07 Agent Contest was the JIAC IV team from the DAI-Labor, Technische Universität Berlin, Germany. They gained the highest number of points: 63. The second team was microJIAC team, from the same institute, with 54 points followed by the Jason team with 49 points. The summary of the whole tournament is summarized in the Table 2.

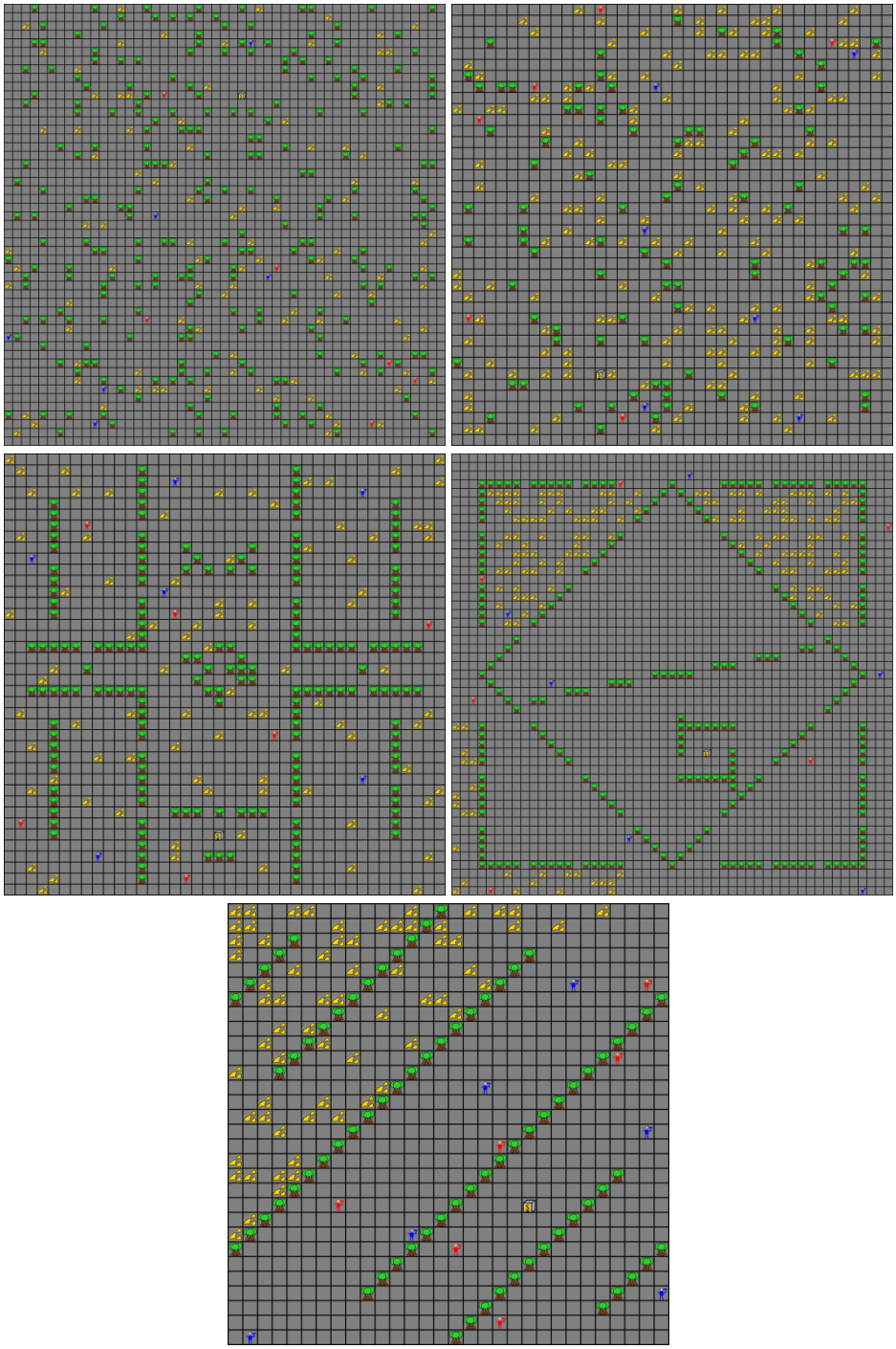

Fig. 1. Initial simulation scenarios *Park*, *Meadow*, *Semiramis*, *Fence* and *Overkill* (left → right, up → down)

Rank	Team	GoldScore	Points
1.	JIAC IV team	2824 : 1759	63
2.	microJiacteam	2680 : 1598	54
3.	Jasonteam	2563 : 1988	49
4.	FLUXteam	2514 : 1816	43
5.	APLteam	1246 : 2585	12

Table 2. Final tournament results.

6 Conclusion

The main motivations behind this Agent Contest are the following:

- to foster the research and development of practically oriented approaches to programming multi-agent systems,
- to evaluate the state-of-the-art techniques in the field, and
- to identify key problems using these techniques.

The three editions of the Agent Contest have convinced us about its impact to the research in multi-agent system development. One important contribution is the great opportunity for the related research groups to participate in this contest in order to test and evaluate their developed agent development approaches. Participating in this contest helps them to discover bugs in their developed tools and technologies (e.g., multi-agent system methodology, agent programming language and their interpreters, agent platforms, etc.). The result is an improvement in the overall quality of the existing multi-agent system development approaches. Moreover, we notice that this contest helps research groups to deepen the understanding of practical aspects of using their approaches.

Another contribution of this contest is that different complementary multi-agent system development approaches are combined and aligned to develop multi-agent systems. For example, both Jason and 2APL teams use existing multi-agent system development methodologies to specify and design systems, which are subsequently implemented in their developed programming languages. Moreover, the implemented systems are then executed by their developed execution platforms. In this way, they can have a better understanding of problems related to the integration of different complementary approaches.

From the last three editions of the Contest we learned that the current scenario scheme does not enforce coordination and cooperation among the agent teams too much. Therefore, for the next edition of the Agent Contest we are planning to rethink the simulation scenarios so that participating agent teams will be required to have more advanced coordination mechanism. In particular this means that we need to introduce a higher level of dependency between the agents. I.e. 1) a single agent alone shouldn't be able to achieve a team goal, and 2) the environment itself has to have its own dynamics as if playing against the agent team.

As we already mentioned above, because of the extensive duration of the Contest tournament, we plan to modify the simulation server so, that it will be

able to run multiple simulations simultaneously. To this end and to improve the simulation server reliability, we plan to migrate the existing software infrastructure to a computer with a higher processing power.

Of course, we also plan to improve the contest management, especially with respect to managing the contest infrastructure, mailing lists and contest schedule planning and announcements. The participating agent teams in the last edition of the contest advise us to have a scenario where there are few pieces of gold, so that good strategies to search for (scarce) gold can be evaluated. They also advised us to have more depots to avoid queues to deliver the gold. We are planning to organize the next edition of the Agent Contest again in association with the ProMAS workshop.

7 Acknowledgements

We are very thankful to the students for the Department of Informatics of Clausthal University of Technology. They worked hard in order to meet all the deadlines and deliver high-quality code. In particular, our thanks go this year to

- *Xavier Queralt Mateu* for the tournament server deployment, administration and maintenance, and
- *Slawomir Deren* for the simulation engine and scenarios development.

And of course we are thankful to *Bernd Fuhrmann*, *Michael Köster*, *David Mainzer* and *Dominik Steinborn* for the support when problems with the technical infrastructure occurred. We also thank all the contest participants who contributed to its success.

References

1. `http://www.sics.se/tac`.
2. `http://www.agentcities.org/EUNET/Competition`.
3. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Multi-Agent Programming: Languages, Platforms, and Applications*. Number 15 in MASA. Springer, Berlin, 2005.
4. R. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3346. LNAI, Springer Verlag, 2005.
5. R. H. Bordini and J. F. Hübner. BDI Agent Programming in AgentSpeak Using *Jason* (Tutorial Paper). In F. Toni and P. Torroni, editors, *CLIMA VI*, volume 3900 of *Lecture Notes in Computer Science*, pages 143–164. Springer, 2005.
6. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents: Components for intelligent agents in Java. AgentLink News Letter, Jan 1999.
7. M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Programming Multi-Agent Systems*, volume 3067. LNAI, Springer Verlag, 2004.
8. M. Dastani, J. Dix, and P. Novák. The First Contest on Multi-Agent Systems based on Computational Logic. In F. Toni and P. Torroni, editors, *Proceedings of CLIMA '05, London, UK*, volume 3900 of *Lecture Notes in Artificial Intelligence*, pages 373–384. Springer, Berlin, 2006.

9. M. Dastani, J. Dix, and P. Novák. The second contest on multi-agent systems based on computational logic. In K. Inoue, K. Satoh, and F. Toni, editors, *CLIMA VII*, volume 4371 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2006.
10. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Agent-Oriented Software Engineering III: Third International Workshop (AOSE'02)*. Springer, LNAI 2585, 2003.
11. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. W. Perram, editors, *MAAMAW*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer, 1996.
12. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.