The Second Contest on Multi-Agent Systems based on Computational Logic

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@tu-clausthal.de, peter.novak@in.tu-clausthal.de`

Abstract. The second edition of the contest on *Multi-Agent Systems based on computational logic* was held in conjunction with the CLIMA '06 workshop in Hakodate, Japan. Based on the experiences from the first edition of this contest ([8]), we decided to improve the setting of the first edition. In particular, we built a server to simulate the multi-agent system environment in which the agents from different groups can sense the environment and perform their actions. In this way, different multi-agent systems can compete with each other for the same resources. This allows for much more objective evaluation criteria to decide the winner. Three groups from Brazil, Spain and Germany did participate in this contest. The actual contest took place prior to the CLIMA workshop and the winner, the group from Brazil, was announced during CLIMA '06.

1 Introduction

Multi-agent systems are a promising paradigm in software engineering. Various multi-agent system development methodologies have been proposed each of which focuses on specific stages of the software development process. For example, Gaia focuses 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, the designers of Gaia propose models to specify and design multi-agent systems, but they ignore the implementation models.

Moreover, there is a growing number of agent-oriented programming languages that are proposed to facilitate the implementation of multi-agent systems. These programming languages introduce programming constructs that can facilitate efficient and effective implementation of multi-agent systems. On the other hand, many aspects involved in multi-agent systems require logical representation and reasoning: to represent agent's knowledge and actions and to reason about them. In the last decades, research on computational logic has resulted in numerous implementable methods and techniques that can be used to model such aspects.

The development of multi-agent systems requires efficient and effective solutions for different problems which can be divided into three classes: the problems related to

1. the development of individual agents,
2. the development of mechanisms to manage the interactions between individual agents, and
3. 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 during MAS development, and*
2. *evaluating state-of-the-art techniques and approaches from both computational logic and 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 *computational logic* in the development of multi-agent systems. We believe that approaches and techniques of computational logic are essential for the development of multi-agent systems ([3,7,4] for at least two reasons:

1. logical approaches have proven to be very useful for specifying and modeling multi-agent systems in a precise manner, and
2. the specification and models can be executed.

We tried to encourage participants to use existing methods and techniques from computational logic research, as well as existing development methodologies and programming languages for multi-agent systems. However, in order to evaluate how computational logic based implementations will perform in a head-to-head competition with other systems, we decided to open the contest also to non-logic based submissions.

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 if there was no agent or obstacle already in that 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.

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.

2.1 Technical Description of the Scenario

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 their actions, was run on the remote contest simulation server. The interaction/communication between agents from one team had to be managed locally, but the interaction between individual agents and their environment (run on the simulation server) was done via Internet. Participating agents had to connect to the simulation server that provided the information about the environment. Each agent from each team did connect and communicate to the simulation server using one TCP connection.

After the initial phase[1], during which agents from all competing teams connected to the simulation server, identified themselves and got a general match information, the competition started. The simulation server controlled the competition by selecting the competing teams and managing the matches and simulations. In each simulation, the simulation server provided in a cyclic fashion sensory information about the environment to the participating agents and expected agent's reaction within a given time limit. Each agent had to react to the received sensory information by indicating which action (including the skip action) it wanted to perform in the environment. If no reaction was received from the agent within the given time limit, the simulation server assumed that

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

the agent has performed the skip action. After a finite number of steps the simulation server stopped the cycle and participating agents received a notification about the end of a match.

2.2 Team, Match, and Simulation

An agent team consisted of four software agents with distinct IDs. There were no restrictions on the implementation of agents, although we encouraged the use of computational logic based approaches. The tournament consisted of three matches. Each match was a sequel of five 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 2) started the first simulation of the match. Each simulation in a match started by notifying the agents from the participating teams and distributing them the details of the simulation. These included for example the size of the grid, depot position, and the number of steps performed by the simulation. 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 had not responded 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.

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 (not more than 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 could contain the following objects in its cells:

- obstacle (a cell with an obstacle could not be visited by an agent),
- gold (an item which could be picked from a cell),
- agent,
- depot (a cell to which gold items were to be delivered in order to earn a point in a simulation), or
- mark (a string data with a maximum of 5 characters which could 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, and 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 could be either hand crafted for the particular scenario, or completely random. During the simulation, gold items could appear randomly (with a uniform distribution) in empty cells of the grid. The frequency and probability of gold generation was simulation specific, however not known to neither agents, nor participants.

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

– absolute position of the agent in the grid,
– the content of the cells surrounding the agent and the content of the cell in which the agent was currently standing in (9 cells in total).

If two agents were standing in each other's field of view, they could recognize whether they were an enemy, or whether they belong to the same team. However an agent was not able to recognize whether the other agent carries a gold item or not. If there was a mark in a cell, which was in an agent's field of view, it received also 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 does not change the state of the environment (under the assumption that other agents do not change it).
– **movements (east, north, west, south):** An agent can move in four directions in the grid. The execution of move east, move north, move west, and move south changes the position of the agent one cell to the left, up, right, and down, respectively. A movement action succeeds only when the cell to which an agent is about to move does not contain another agent or obstacle. Moving to and from the depot cell is regulated by additional rules described later in this description.
– **pick, drop:** An agent can carry only one gold item which it successfully picked up before. An agent can pick up a gold item if 1) the cell in which an agent stands in contains gold, and 2) the agent is not carrying another gold item. An agent can drop the gold item it is carrying only in the cell it is standing in. The result of a successful pick action is that in the next simulation step the acting agent was considered to carry a gold item and the cell, it is standing in, does not contain the gold item any more. The result of a drop action is that the acting agent is not carrying the gold item anymore and that the cell it is standing in contains the gold item in the next simulation step. Dropping a gold item to a depot cell does increase the score of the agent's team by one point. The depot cell does never contain a gold item that can be picked by an agent.
– **mark, unmark:** An agent is allowed to mark a cell it is standing in by a string data with a maximum of 5 characters. The result of a mark action was that the cell in which an agent was located, contained a string in the next simulation step. The depot cell, and cells containing an obstacle could not be marked. By marking a previously marked cell, the old mark was removed and replaced by the new one. If the cell in which an agent was located, contains a mark, then the agent received the string in the perception information from the simulation server. An agent is also allowed to unmark the marked cell it is standing in. The result of an unmark action is that the cell does 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 is sent to them in the next simulation step.

All actions, except the skip action, could fail. The result of a failed action was the same as the result of the skip action. An action could fail either because the conditions for its successful execution were not fulfilled, or because of the information distortion (described later in this text).

Depot cell There are strong conditions imposed on the depot cell:

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

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

Timeout, Information Distortion, and Final Phase The agents had to inform the simulation server which actions they want to perform within a timeout specified at the beginning of the simulation. The contest organizers did not take any responsibility for the speed of the Internet connection between the server and participating agents. Timeouts were set reasonably high, so that even participants with a slow network connection, or using a time demanding deliberation, were able to communicate with the server in an efficient way.

A ping interface was provided by the server in order to allow participating agents to test the speed of their connection during the whole duration of the tournament.

Agents could receive incomplete information about the environment from the simulation server. The simulation server could omit information about particular environment cells. However, this happened only with a certain probability, guaranteed to be not higher than 20 percent and fixed for each simulation.

In the final phase, the simulation server sent a message to each agent allowing them to disconnect from the server ending the tournament.

2.4 General Agent-2-Server Communication Principles

Agents communicated with the contest server by exchanging XML messages and using standard TCP/IP stack with socket session interface. Messages were XML documents that could be analyzed by standard XML parsers available for many programming languages. The Internet coordinates (IP address and port) of the contest server (and a dedicated test server) were communicated to the participants via the official CLIMA VII Contest mailing list.

Communication Protocol The tournament was divided into three phases. During the initial phase, agents connected to the simulation server and identify themselves by user-name 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 could either succeed or fail. After successful authentication, agents had to wait 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 information: simulation ID, opponent's ID, grid size, number of steps the simulation last and the depot position.

In each simulation step an agent received a perception about its environment (REQUEST-ACTION message) and it had to respond by performing an action (ACTION message). Each request-action message contained information about 9 neighbouring cells around agent (including the one agent stands on), its absolute position in the grid, simulation step number and a deadline for its response. Agent had to deliver its response within the given deadline. The action message had to contain the identifier of the action 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 and the information about the result of the simulation (whether the team won, or lost the simulation).

All agents that 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 its connection to the simulation server, the tournament proceeded without disruption, only all the actions of a disconnected agent were considered to be empty. Agents were themselves responsible for maintaining the connection to the simulation server and in a case of connection disruption, they were allowed to reconnect.

Agents reconnected by performing the same sequence of steps as at the beginning of the tournament. After establishing the connection to the simulation server, it could send AUTH-REQUEST message and receive AUTH-RESPONSE. After successful authentication, the server could send a 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 a next simulation in which it participates started. In the next subsequent 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. Agents could send a PING message containing a payload data (ASCII string up to 100 characters) and receive PONG message with the same payload. As all messages contained a timestamp, agents could also use ping interface to synchronize their time with the server.

Protocol Sequence Diagram (UML like notation)

```
* Initial phase
       Server                         Agent
        |         AUTH-REQUEST            |
        |<<-------------------------------|
        |                                 |
        |         AUTH-RESPONSE           |
        |------------------------------->>|


* Simulation
       Server                         Agent
        |         SIM-START               |
        |------------------------------->>|
        |                                 |
+----------------------------------------------+
| loop: Simulation Step Cycle             |    |
|       |                                 |    |
|       |         REQUEST-ACTION          |    |
|       |------------------------------->>|    |
|       |                                 |    |
|       |         ACTION                  |    |
|       |<<-------------------------------|    |
|       |                                 |    |
+----------------------------------------------+
        |                                 |
        |         SIM-END                 |
        |------------------------------->>|
```

```
* Reconnect
       Server                         Agent
        |         AUTH-REQUEST            |
        |<<-------------------------------|
        |                                 |
        |         AUTH-RESPONSE           |
        |------------------------------->>|
        |                                 |
        |         SIM-START               |
        |------------------------------->>|
                     +------------------------+
                     | ref:                   |
                     | Simulation Step Cycle  |
                     +------------------------+


* Ping
       Server                         Agent
        |         PING                    |
        |<<-------------------------------|
        |                                 |
        |         PONG                    |
        |------------------------------->>|

* Final phase
       Server                         Agent
        |         BYE                     |
        |------------------------------->>|
```

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: Contained the standard XML document header
  `<?xml version="1.0" encoding="UTF-8"?>`
– Message envelope: The root element of all XML messages was `<message>`. It had 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 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 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 CLIMA VII webpage[2].

3 Submission

A submission consisted of two parts. The first part was a description of analysis, design and implementation of a MAS for the above application. We have encouraged submissions that specify, design and implement multi-agent systems using existing agent-oriented methodologies such as Gaia [12], Prometheus [9] and Tropos [6]. For the description of the implementation, the authors were asked to explain how their design is implemented. In particular, they were asked to explain which computational logic techniques (e.g. logic programming, formal calculi, etc.) are used to implement aspects of the multi-agent system such as mental states (e.g., goals, beliefs, plans, and roles) of individual agents, communication, coordination, negotiation, and dialogue. Although we emphasized the use of computational logic techniques, we did not exclude submissions that were based on other approaches (e.g. based on machine learning, neural nets, etc.) and programming paradigms.

The second part was the participation in the contest tournament by means of an (executable) implementation of a multi-agent system. The source code together with instructions on how to install it, including precise instructions on software and hardware requirements, had to be submitted just before the competition started.

[2] `http://cig.in.tu-clausthal.de/fileadmin/user_upload/_temp_/c7c-protocol.txt`

3.1 Received Submissions

We have received three submissions for this edition of the CLIMA contest. From the received submissions, two submissions used an existing multi-agent development methodology to specify, design, and implement a running multi-agent system. The use of computational logic techniques was explicitly discussed in two of the submissions. The use of computational logic in the third submission emerged mostly from using Prolog as the programming language for implementing the multi-agent system.

The submission from R.H. Bordini, J.F. Hübner, and D.M. Tralamazza uses Prometheus [9] as the multi-agent 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 (searching) and courier agents. These agents are subsequently specified and designed in terms of Goal Overview Diagram and 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-oriented programming language AgentSpeak [10]. As it was required by the contest, their multi-agent system consisted of four individual agents. These agents follow a general strategy according to which each agent is responsible for one quadrant of the grid environment. Each agent can then play two roles: carrying gold or searching for gold. The agents from the team can communicate to help each other. For example, a searching agent that finds some gold can ask a courier agent to transport the gold to the gold depot. The use of computational logic techniques in this submission emerges mostly from the use of computational logic techniques in AgentSpeak language and its Jason interpreter.

The submission from C. Cares, X. Franch, and E. Mayol uses a goal-oriented development methodology which is based on a combination of antimodels (a requirement engineering technique) and an extension of Tropos [6]. This methodology is used to specify and design a multi-agent system for the Gold mining scenario of the contest. They specify the goals of the multi-agent system, analyze possible attacks on these goals (e.g., the goal of the competitor team), and propose adequate responses to such attacks. Based on the resulting specification, they identify possible involved agents and design their strategies. Their extension of Tropos enables them to generate a Prolog implementation of the identified agents and their designed strategies. The fact that the implementation is based on Prolog seems to be the only use of computational logic in their system.

The final submission from S. Schiffel and M. Thielscher does not use any specific multi-agent system development methodology. The focus on this submission is rather on the use of computational logic techniques in implementing the Gold mining scenario. In particular, they implement their multi-agent system in FLUX [11], which is a programming language based on constraint logic programming. The agent implemented in FLUX can reason logically about their sense information and actions in the presence of incomplete knowledge. Each agent builds a mental model of the environment by sensing the environment and performing actions. A FLUX agent program consists of a kernel (the reasoning capability of the agent), a domain-specific background theory (the model of the environment), and a strategy (the behaviour of the agent). The mental model of the agents is defined in terms of fluents that represent the position of agents, the position of the gold depot, the position of the obstacles, and the fact that an agent carries gold. The strategies of the agents are defined in terms of the actions that are proposed in the contest description. In this system, all FLUX agents share the same background theory, and each agent acts according to its individual strategy and the role it plays

4 Technical infrastructure

In order to run the competition, we developed a multi-agent system environment simulation server MASSim. 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* - which 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 webinterface.

Most of the software components were implemented in Java 1.5.0. The webinterface module, running as a set of Java servlets under Apache webserver with Tomcat application server, was loosely connected to the core simulation server via Java RMI (Remote Method Invocation) protocol so, that if a need arose due to high CPU load, we could run the webinterface and the core simulation server on different computers.

The whole MASSim server architecture was designed with the following requirements in mind:

1. *high versatility*—the core system (MASSim simulation server) should depend on the most standard software today so that contest participants are able to download and install the system without a hassle. It should be easily deployable on standard configurations using Linux OS, Apache webserver and Tomcat application server;
2. *open and reusable* - we designed the server so that it can accommodate vast range of discrete-time simulation scenarios which can be easily connected to the core server API;

3. *component based design* - each MASSim simulation server component communicates with other components using a well-defined API such that we are able to replace it quickly on the ground of changing requirements (e.g. different tournament structure, different network communication protocol, or a visualization technology)

The system, including a documentation which is still partly a work in progress, is published on the official Contest website: `http://cig.in.tu-clausthal.de/CLIMAContest/`.

4.1 Contest preparation

Several days before the start of the competition, the contest organizers contacted participants via e-mail with details on time and Internet coordinates (IP addresses/ports) of the simulation server. Participants received also agent IDs necessary for identification of their agents for the tournament. Agents had to communicate with the simulation server using TCP protocol and by means of messages in XML format. The details about communication protocol and message format was specified and provided to participants long enough before the actual competition.

The MASSim simulation server system proved to be a reliable and successful platform for the CLIMA VII Contest. During the first testing phase (05.02.2006–15.03.2006), we published the system on the Contest website. The participants could download it and use it for development and debugging of their multi-agent systems. Together with the simulation server, we published also a sample implementation of an agent team. The contest participants could copy our full-fledged working contest protocol implementation written in Java. This should speed up the agent team development and participants could focus more on the scenario strategy rather than the technical issues.

The main testing phase of the contest was run between March 15th and April 27th 2006. During this period we ran the MASSim simulation server on our network infrastructure with a slightly modified tournament structure implementation. Participants had to subscribe for a testing account and after receiving valid credentials for each of their agents they could start using the test server. Agents could connect to our test MASSim server running at `agentmaster.in.tu-clausthal.de` port 12300 and participate in a test match against our *CLIMABot* agent team. We did not allow agents of different teams compete against each other as this should happen exclusively during the tournament itself.

For completeness, we give a short description of our own *CLIMABot* team. The agents are completely on their own, there is no communication, no lobal map building. An agent can be in two modes:

Mode 1: the agent moves randomly in its environment and looks for gold. It remembers all gold positions and stores them in a list. This list is updated while it is wandering around.

Mode 2: if an agent wants to get to a target at the position (X,Y) it tries the shortest way to get there. If there are more shortest ways (as the grid is rectangular and in the case when there are no obstacles directly around it) it chooses the shortest path randomly.
If on the direct way to the target it bumps into an obstacle so that it cannot proceed further in the desired direction it pushes the current direction it wants to go to the stack, then it turns clockwise (to the right) and tries to go in the new direction each step checking whether it is not possible to turn into the desired direction (top of the stack). If on this way the agent bumps into another obstacle it just pushes the current direction onto the stack, again turns clockwise and proceeds. It does this thing until the stack is empty. Otherwise the stack is just filled with stuff forever.

If, being in mode 1, an agent finds a piece of gold, it picks it and switches to mode 2. In the case it finds another piece of gold on its way to the depot it remembers its position in a list. Once it delivered the piece of gold, it picks the nearest known piece of gold from the list and goes to it applying the algorithm in mode 2. If the list is empty, it switches to mode 1.

4.2 Tournament

The CLIMA VII Contest tournament was scheduled for Thursday, April 27th 2006, 15:00 CEST (GMT+2). However, because of last-minute technical difficulties of the Spanish team, we had to postpone the start of the tournament for a couple of minutes so the tournament finally started at 15:26 the same day. First, we let all the participating teams to compete against our *CLIMABot* agent team and then real tournament matches followed in the following order Germany:Spain, Brazil:Spain and Brazil:Germany. The tournament finished on April 27th at 21:58 after 9 hours and 32 minutes. Matches took from 27 minutes (Germany:CLIMABot) to 1 hour 48 minutes (Spain:CLIMABot).

During the tournament itself, the current tournament status could be watched in real-time using our webinterface at `http://agentmaster.in.tu-clausthal.de/`. Throughout the whole tournament we have not observed any technical problems. All the 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

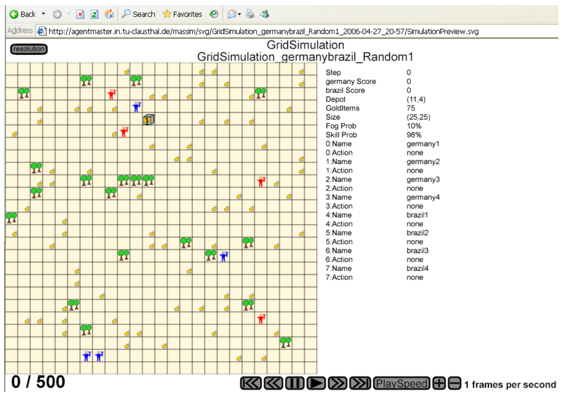
The teams competed in matches each consisting of 5 simulations with identifiers *Random1*, *Random2*, *Labyrinth1*, *Labyrinth2* and *Labyrinth*. As the names suggest, the first two simulation instances *Random1* (Figure 1) and *Random2* were randomly generated simulations differing in the parameters while the last three were handcrafted mazes.

Labyrinth1 (Figure 2) was a maze of rows of obstacles with gates at the very ends. This simulation instance proved to be the most difficult for the participating agent teams, because a random exploration of the grid did not lead to satisfactory results. In order to collect gold items in this simulation instance agents had to implement a systematic search of the environment.

Labyrinth2 and *Labyrinth* (Figure 3) were inverted versions (w.r.t. initial position of agent teams) of the same simulation instance consisting of a simple rectangular maze with some holes in the structure and the depot in the middle. In order to succeed in this simulation instance, agents had to develop an internal representation of the environment, but a systematic exploration of the environment was not a hard requirement.

The following is a detailed description of simulation instances:

simulation ID:	*Random1*	*Random2*	*Labyrinth1*	*Labyrinth2*	*Labyrinth*
grid size:	25x25	25x25	35x35	30x30	30x30
depot position:	random	random	(10,17)	(14,14)	(14,14)
number of obstacles:	20	40	305	126	126
initial number of gold items:	75	20	60	45	45
information distortion probability:	10%	10%	3%	1%	1%
action failure probability:	2%	2%	2%	2%	2%
gold generation frequency:	3sec	3 sec	3 sec	3 sec	3 sec
number of generated gold items:	1	1	1	1	1
number of simulation steps:	500	500	500	500	500


Back
Search
Favorites
Address
http://agentmaster.in.tu-clausthal.de/massim/svg/GridSimulation_germanybrazil_Random1_2006-04-27_20-57/SimulationPreview.svg
resolution
GridSimulation
GridSimulation_germanybrazil_Random1
0 / 500
PlaySpeed
1 frames per second

```
Step            0
germany Score   0
brazil Score    0
Depot           (11,4)
GoldItems       75
Size            (25,25)
Fog Prob        10%
Skill Prob      98%
0:Name          germany1
0:Action        none
1:Name          germany2
1:Action        none
2:Name          germany3
2:Action        none
3:Name          germany4
3:Action        none
4:Name          brazil1
4:Action        none
5:Name          brazil2
5:Action        none
6:Name          brazil3
6:Action        none
7:Name          brazil4
7:Action        none
```

Fig. 1. Simulation instance *Random1* with a screenshot of the visualization interface.

5 Contest results

The winner of the CLIMA VII Contest was the team from Brazil with the highest number of points: 25. The second team was from Spain with 14 points followed by the team from Germany with 4 points. The summary of the whole tournament

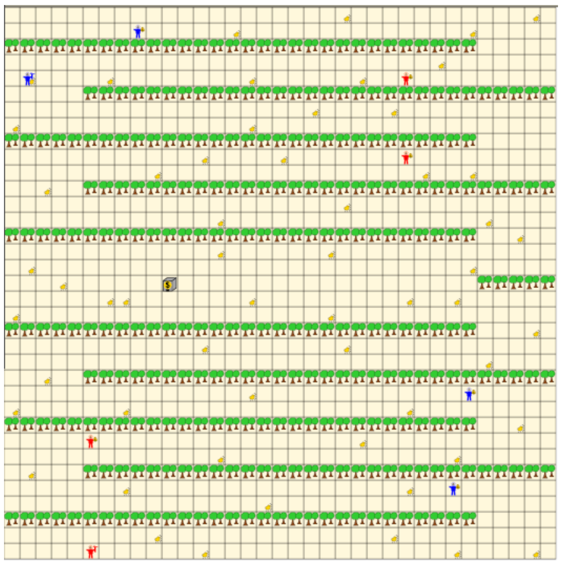


Fig. 2. Simulation instance *Labyrinth1*.

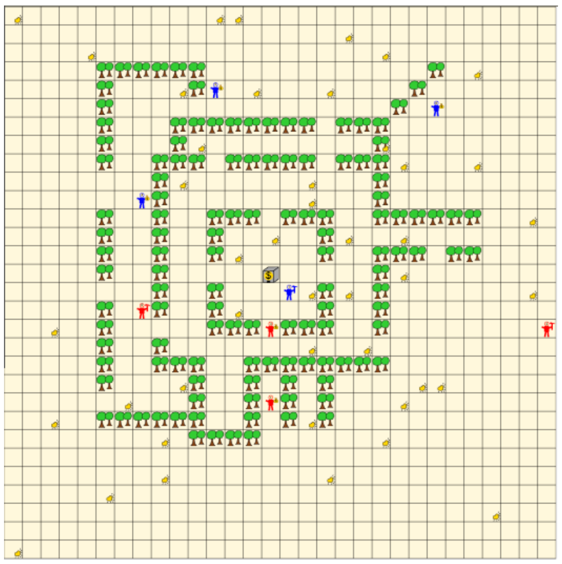


Fig. 3. Simulation instance *Labyrinth2*.

is summarized in the Table 1. All the partial summaries of the matches can be found in Tables 2, 3 and 4.

We did not include the results of our *CLIMABot* team. In fact, had our team attended the contest, it would have been the winner. This is interesting because it shows that not a lot of logic or strategies are needed to win the whole contest. Our team has been written by our students in just 2 days. We are still not sure why such a simple strategy turned out to be superior to all other contestants.

Rank	Team	Score	Points
1.	brazil	265 : 125	25
2.	spain	225 : 176	14
3.	germany	79 : 268	4

Table 1. Final tournament results.

Simulation/Score	germany	spain
Random1	36	33
Random2	6	10
Labyrinth	2	36
Labyrinth1	0	0
Labyrinth2	3	53

Table 2. Summary of the match between team Germany and the team Spain.

Simulation/Score	brazil	spain
Random1	33	33
Random2	3	4
Labyrinth	41	22
Labyrinth1	15	2
Labyrinth2	37	32

Table 3. Summary of the match between team Brazil and the team Spain.

6 Conclusion

The major motivations behind organizing the CLIMA Contest are the following:

– to foster the research and development of practically oriented approaches to programming multi-agent systems,

Simulation/Score	brazil	germany
Random1	23	18
Random2	11	9
Labyrinth	50	2
Labyrinth1	11	0
Labyrinth2	41	3

Table 4. Summary of the match between team Brazil and the team Germany.

– to evaluate the state-of-the-art techniques in the field, and
– to identify key problems using these techniques.

After successfuly organizing two editions of the CLIMA Contest we still cannot give a full account of the impact of our Contest to the research in MAS. However we can briefly summarize what we believe are the major contributions of the Contest.

Finding bugs: Apart from growing attention, we learned from participants of the CLIMA Contest that they gathered a lot of practical experience in programming agent teams for the contest. We consider this already a major success: It helps to deepen the understanding of practical aspects of using tools, approaches and languages. Participating in the CLIMA Contest already helped to discover several bugs in prominent agent programming language interpreters and thus to improve the overall quality of the tool.

Classroom use: In the meantime, the technical infrastructure we developed for the CLIMA Contest, is already used in teaching processes of at least two universities in Germany and Australia. The advantage of using our simulation server in MAS lectures is its versatility and simplicity of the Contest scenario. Practical experience of students with programming simple multi-agent systems using our simulation server also helps to raise general awareness about multi agent research and applications of it.

Objective evaluation: A difficulty with the previous edition of this contest was the lack of an objective evaluation criterion by means of which the winner of the contest could be decided. In fact, the evaluation of the last contest edition was partially based on the performance of the agent teams in grid-like environments that were built by the participating groups themselves. As a consequence, different agent teams could not compete with each other in one and the same shared environment. Therefore, we decided for this contest edition to build an environment server in order to create equal conditions for the participating agent teams. Moreover, this environment server enables agent teams to focus on computational techniques and methods to implement gold mining strategies (using local communication technologies and hardware) by taking off the burden of implementing the simulation environment from the participants' shoulders.

As the whole software infrastructure is already developed and the feedback from the CLIMA VII Contest was in favour of keeping the current competition scenario, we are only planning to slightly modify and improve our scenario and simulation instances. As far as the contest management concerns, we learned a lot during the CLIMA VII Contest, especially with respect to managing the contest infrastructure, mailing lists and contest schedule planning and announcements. As the size of the last tournament in terms of number of participants was rather low, we did not consider to divide participating teams into groups and execute the tournament in several eliminating rounds.

We are currently finalizing and improving the documentation of our software packages. They will be freely made available and can be used by interested colleagues.

7 Acknowledgements

We are very thankful to our students in 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 to

– *Bernd Fuhrmann* for the core server component, contribution to the overall architecture and the code repository management,
– *Michael Köster* for the CLIMA VII Contest simulation plug-in, deployment scripts and the run-time server configuration,
– *David Mainzer* for the SVG visualization component, and
– *Dominik Steinborn* for the webinterface, its connection with the core server and his care for the testing and run-time server deployment.

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. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the TROPOS project. *Information Systems*, 27:365–389, 2002.
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. Novak. 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. 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.
10. 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.
11. M. Thielscher. FLUX: A logic programming method for reasoning agents. *CoRR*, cs.AI/0408044, 2004.
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.