The Multi-Agent Programming Contest 2012

Michael Köster[1], Federico Schlesinger[1],
and Jürgen Dix[1]

Department of Informatics, Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
{dix, michael.koester, federico.schlesinger}@tu-clausthal.de

Abstract. The Multi-Agent Programming Contest, MAPC, is an annual, community-serving competition that attracts groups from all over the world. Its aim is to facilitate advances in programming multiagent systems (MAS) by (1) developing benchmark problems, (2) enabling head-to-head comparison of MAS's and (3) supporting educational efforts in the design and implementation of MAS's. We report about its eighth edition and give a detailed overview of the participants strategies and the overall contest.

1 Introduction

This paper serves as an introduction to the subsequent papers in this proceedings volume, each of which describes a team that participated in this years edition. We give a comprehensive overview of the Multi-Agent Programming Contest[1] 2012, an annual international event that has started in 2005 as an attempt to stimulate research in the field of programming multi-agent system by 1) identifying key problems, 2) collecting suitable benchmarks, and 3) gathering test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. In 2012 the competition was organised and held for the eighth time.

Research communities in general benefit from competitions that attempt to evaluate different aspects of the systems under consideration and furthermore allow for comparing state of the art systems, act as a driver and catalyst for developments and pose challenging research problems.

In this paper we (1) briefly introduce the Contest and its infrastructure, (2) elaborate on the 2012 scenario and its differences with the 2011 edition, (3) introduce the seven teams that took part in the tournament, and (4) present results and findings acquired before, during and after the tournament.

More detailed information about the strategies of the teams are to be found in the remaining six papers in this volume.

[1] http://multiagentcontest.org

1.1 Related Work

The Multi-Agent Programming Contest has generated quite a few publications over the years [9,10,11,3,4,1,8]. For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to [1,8,5,6]. A quick non-technical overview appears in [2].

Similar contests, competitions and challenges have taken place in the past few years. Among them we mention *Google's AI challenge*[2], the *AI-MAS Winter Olympics*[3], the *Starcraft AI Competition*[4], the *Mario AI Championship*[5], the *ORTS competition*[6], and the *Planning Competition*[7]. Every such competition rests in its own research niche. Originally, our Contest has been designed for problem solving approaches that are based on formal approaches and computational logics. But this is not a requirement to enter the competition.

1.2 The contest from 2005–2012

From 2005 to 2007 we used a classical gold miners scenario [10] and introduced the *MASSim* platform: A platform for executing the Contest tournaments.

From 2008 to 2010 we developed the cows and cowboys scenario which has been designed to enforce cooperative behavior among agents [4]. The topology of the environment was represented by a grid that contained, besides various obstacles, a population of simulated cows. The goal was to arrange agents in a manner that scared cows into special areas, called corrals, in order to get points. While still maintaining the core tasks of environment exploration and path planning, we also made the use of cooperative strategies an obligation.

The agents on Mars scenario, used during the 2012 edition and discussed in this paper, was firstly introduced in 2011 [5]. In short, we have generalized the environment topology to a weighted graph. Agents were expected to cooperatively establish a graph covering while standing their ground in an adversarial setting and reaching achievements.

2 MAPC 2012: Agents on Mars

In this section we give a detailed overview of the 2012 agents on Mars scenario and point out differences to the scenario from 2011.

2.1 The Scenario

It is now a tradition to accompany the technical description of each scenario with a motivating little story:

[2] http://aichallenge.org/
[3] http://www.aiolympics.ro/
[4] http://eis.ucsc.edu/StarCraftAICompetition
[5] http://www.marioai.org/
[6] http://skatgame.net/mburo/orts/
[7] http://ipc.icaps-conference.org/

In the year 2033 mankind finally populates Mars. While in the beginning the settlers received food and water from transport ships sent from earth shortly afterwards – because of the outer space pirates – sending these ships became too dangerous and expensive. Also, there were rumors going around that somebody actually found water on Mars below the surface. Soon the settlers started to develop autonomous intelligent agents, so-called All Terrain Planetary Vehicles (ATPV), to search for water wells. The World Emperor – enervated by the pirates – decided to strengthen the search for water wells by paying money for certain achievements. Sadly, this resulted in sabotage among the different groups of settlers.

Now, the task of your agents is to find the best water wells and occupy the best zones of Mars. Sometimes they have to sabotage their rivals to achieve their goal (while the opponents will most probably do the same) or to defend themselves. Of course the agents' vehicle pool contains specific vehicles. Some of them have special sensors, some are faster and some have sabotage devices on board.

Last but not least, your team also contains special experts, e.g. the repairer agents, that are capable of fixing agents that are disabled. In general, each agent has special expert knowledge and is thus the only one being able to perform a certain action. So your agents have to find ways to cooperate and coordinate among them.

The environment's topology is constituted by a weighted graph. Each vertex has a unique identifier and a number that indicates its value. Each edge has a number that represents the costs of moving from one of its vertices to the other. These vertex-values are crucial for calculating the values of zones. A zone is a subgraph that is covered by a team of agents according to a coloring algorithm that is based on a domination principle.

Several agents can stand on a single vertex. If a set of agents dominates such a vertex, the vertex gets the color of the dominating team. A previously uncolored vertex that has a majority of neighbors (at least 2) with a specific color, inherits this color as well. Finally, if the overall graph contains a colored subgraph that constitutes a frontier or border, all the nodes that are inside this border are colored as well. This means that agents can color or cover a subgraph that has more vertices than the overall number of agents. Figure 1 shows a screenshot of a relatively small map, depicting, amongst other things, the graph coloring.

Before elaborating on the agent roles we have to specify the effectoric capabilities of the agents. Each agent, or vehicle, has a state that is defined by its position on the map, its current energy available for executing actions and its current health. On top of that, each team has a budget for equipping the vehicles during the simulation. These actions[8] are defined by the scenario:

– `skip` is the noop-action, which does not change the state of the environment,

[8] Of course, all the actions that cost energy will fail if the vehicle under consideration does not have enough energy.

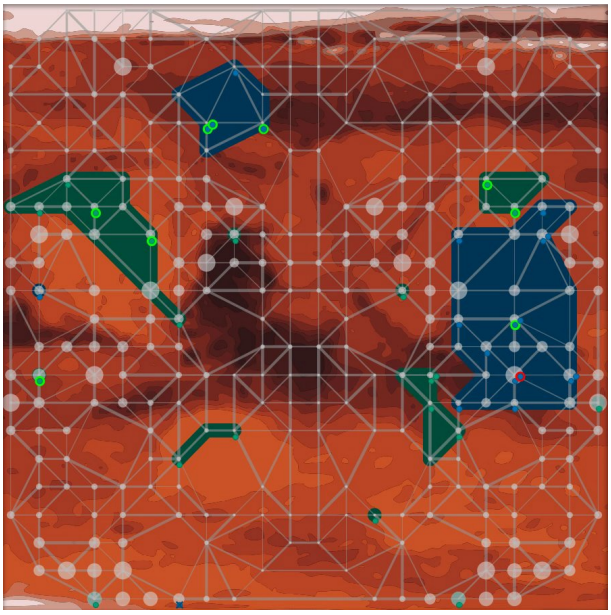

Fig. 1. A screenshot of the agents on Mars scenario.

- `recharge` increases the current energy of a vehicle by a fixed factor and can be performed at any time without costs,
- `attack` decreases the health of an opponent, standing on the same vertex, if successfully executed and decreases the current energy of the attacker,
- `parry` parries an attack and decreases the energy of the defending agent,
- `goto` moves the vehicle to a neighboring vertex while decreasing its energy by the weight of the traversed edge,
- `probe` yields the exact value of the vertex the vehicle is standing on and decreases the probing vehicle's energy,
- `survey` yields the exact weights of visible edges while decreasing the energy,
- `inspect` costs energy and yields the internals of all visible opponents,
- `buy` equips the vehicle with new components, which increase its performance, and cost money, and
- `repair` repairs a teammate, which again costs energy.

We have defined five different roles. Each team consists of four vehicles for each role, that is a total of twenty vehicles per team. This number increased from the 2011 edition, where teams were composed by 2 vehicles for each role, totaling 10 vehicles. Each role defines the vehicle's internals and its capabilities. The roles differ with respect to energy, health, strength and visibility range. The effectoric capabilities are as follows:

- **explorer** can skip, move to a vertex, probe a vertex, survey visible edges, buy equipment and recharge its energy,

- **repairer** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment, repair a teammate and recharge its energy,
- **saboteur** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment, attack an opponent and recharge its energy,
- **sentinel** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment and recharge its energy,
- **inspector** can skip, move to a vertex, inspect visible opponents, survey visible edges, buy equipment and recharge its energy.

Achievements are tasks that, if fulfilled, contribute to the teams' budgets. We have defined a set of achievements that includes having zones with fixed values, inspecting a specific number of vehicles, probing a number of vertices, surveying a fixed number of edges and successfully performing and parrying a number of attacks.

In each step, each vehicle is provided with its currently available percepts:

- the state of the simulation, i.e. the current step,
- the state of the team, i.e. the current scores and money,
- the state of itself, i.e. its internals,
- all visible vertices, i.e. identifier and team,
- all visible edges, i.e. their vertices' identifiers,
- all visible vehicles, i.e. their identifier, vertices and team,
- probed vertices, i.e. their identifier and values,
- surveyed edges, i.e. their vertices' identifiers and weights, and
- inspected vehicles, i.e. their identifiers, vertices, teams and internals.

After sending percepts, the server grants some time for deliberation. After that the new state is computed. The simulation state transition is as follows:

1. collect all actions from the agents,
2. let each action fail with a specific probability,
3. execute all remaining `attack` and `parry` actions,
4. determine disabled agents,
5. execute all remaining actions,
6. prepare percepts,
7. deliver the percepts.

The introduction of the agents on Mars scenario was also accompanied by the release of an environment interface that has been developed to be compatible with the *environment interface standard* [7]. This standard allows Java based problem solving approaches to make use of a jar-file provided by the organizers that facilitated connecting to and communicating with the *MASSim* server. This is done my mapping the whole communication to Java-method invocations and callbacks.

2.2 Changes and Modifications to the Scenario from 2011

As already mentioned, we increased the number of agents to 20 and provided them with more energy. This results in less recharging and gives them more freedom: in 2011, recharge was by far the most used action.

The visualisation was improved a lot (zones as well as high-valued vertices are highlighted, costs of the edges are depicted by their thickness. The last action from an agent at each vertex is illustrated: (1) green circle: successful sense action (probe, survey, inspect), (2) red circle: last action failed, (3) yellow star: successful attack, (4) indigo star: successful parry, (5) pink star: successful repair, and (6) crossed out: disabled.

Agents are now getting feedback as to why their actions failed (if they did). The (automatic) generation of maps has been improved (a map contains now several centers).

3 The Tournament

During past editions of the Contest, stability (i.e., the capacity to send actions to the *MASSim* server in time) was a big problem for some teams. It also affected the overall quality of the Contest and the possibility to draw conclusions about the strategies by looking at the results. To address this, we decided for the 2012 edition to implement a *qualification round*, in which teams were required to show that they were able to maintain good stability (i.e. timeout-rates below 5%) during a round of test matches. Only then they were allowed to take part in the tournament.

3.1 Participants and Results

Nine teams from all around the world registered for the Contest. Seven of them were able to pass the qualification round and took part in the tournament (see Table 1). Full introductions of the teams can be found in [12] and in the papers included in this volume.

Team	**Affiliation**	**Platform/Language**
AiWYX	Sun Yat-Sen University, China	C++
PGIM	Islamic Azad University of Malayer, Iran	Prometheus, JACK
LTI-USP	University of Sao Paulo, Brazil	Jason, CArtAgO, Moise
SMADAS-UFSC	Federal University of Santa Catarina, Brazil	Jason
Python-DTU	Technical University of Denmark	Python
Streett	- , USA	Java
TUB	TU Berlin, Germany	JIAC

Table 1. Participants of the 2012 edition.

Team AiWYX was a single-developer team from Sun Yat-Sen Univerity, China. The agents were developed in C++, using no agent-specific technologies. The approach used is centralized, where one agent gets all the percepts from the other agents and makes the decisions for the whole team.

Team PGIM comes from the Islamic Azad University of Malayer, Iran. The 3 developers used agent-specific technologies for developing their team: Prometheus, JACK. Nevertheless the team organization is not distributed, and agents broadcast their percepts.

Team LTI-USP from University of Sao Paulo, Brazil had three developers. Agents were implemented using Jason, CArtAgO and Moise. There is one agent that determines the best strategy, but each agent has its own thread, with its own beliefs, desires and intentions. Agents broadcast new percepts, but communication load decreases over time.

Team SMADAS-UFSC is from Federal University of Santa Catarina, Brazil. It had six team members. The language of choice for agent development was Jason. Besides normal agent-communication provided by Jason, agents shared a common data-structure (blackboard) for storing the graph topology.

Team Python-DTU from the Technical University of Denmark is a regular contender of the Multi-Agent Programming Contest. For this edition it registered 6 members. As team's name suggest, Python was the language of choice. The agents follow a decentralized approach, where coordination is achieved through distributed algorithms, e.g. for auction-based agreement.

Team Streett was composed by a single independent developer from the USA. Agents were developed in Java, based on the sample agents provided with the *MASSim* platform. Agents shared only vital information and coordination was achieved by sharing location data.

Team TUB, TU Berlin, Germany, is another regular contender of the Multi-Agent Programming Contest, that presented for this edition as a single-developer team. The agents are developed in the JIAC platform (which won the contest several times in previous years).

The tournament took place from 10th to 12th September 2012. Each day each team played against two other teams so that in the end all teams played against all others. We started the tournament each morning at 10 am and finished at around 3 pm. A match between two teams consisted of 3 simulations only differing in the size of the graph. For a win the team got 3 points and for a draw 1 point. The results of this year's Contest are shown in Table 2.

Pos.	**Team**	**Score**	**Difference**	**Points**
1	SMADAS-UFSC	2778057 : 1043023	1735034	51
2	Python-DTU	2738397 : 1095251	1643146	48
3	TUB	2090849 : 1600914	489935	30
4	LTI-USP	1627177 : 1845601	-218424	27
5	AiWYX	2301358 : 1526768	774590	24
6	PGIM	1130432 : 2047735	-917303	9
7	Streett	192694 : 3699672	-3506978	0

Table 2. Results.

Two teams, *SMADAS-UFSC* and *Python-DTU*, stood out from the rest and the tournament winner was decided by the match that confronted them, during the second day of the competition. *SMADAS-UFSC* won two of three simulations

of that match and was crowned champion, leaving *Python-DTU* as runner-up for the second consecutive year. Both teams won all the matches they played against the rest of the teams without losing any simulations. The mid-table teams *TUB*, *LTI-USP* and *AiWYX* where relatively close while playing against each other. They could not catch up with the first two teams but clearly differentiated from the last two.

Thanks to the qualification round (as well as the optional test matches offered before it), there were no stability issues during the Contest. This was a great improvement compared to previous editions. Although some of the teams experimented a few crashes from time to time, the promptness of the developers to restart their agents ensured that the results of the simulation were not affected by these isolated events.

3.2 Overview of the Teams' Strategies

In this section we collect a few facts about the participating teams. For more detailed information we refer to the articles in these proceedings.

SMADAS: The winner of this years contest, from Brazil, used Jason, a dedicated MAS programming language. For some algorithms, Java was used to implement them, rather than Jason. The development needed 500 person hours distributed among 6 people. They used 7900 lines of code, 2400 of which were written in Java. Communication with the server was done through the EISMASSIM interface.

The system is decentralized. Agents were executed on the same machine to use shared memory (blackboard programming). But updating the blackboard was computationally difficult and thus could only be done every 3 steps.

The strategy was first to explore the map, find the best potential zones (high values) and then to conquer and defend them. An interesting idea was to make the opponents spend their money using a special agent: Hulk. If the team detects that there is no particular buying strategy, then the Hulk agent changes its behaviour.

They claim that the good performance is based on the various strategies that make the team very flexible against different opponents. Defending of the zones can still be improved.

Python-DTU: The danish team ended as runner-up for the second time in a row. The team did not use a dedicated platform or MAS programming language. They choose Python for efficiency and to have complete control over all features in the implementation. However, the team used the organizational model of *Moise*.

The solution they implemented is decentralized and heavily based on communications between the agents and on an auction-based agreement algorithm. They invested 300 person hours distributed among 6 people. 1500 lines of codes were written.

The strategy is based on dividing the game in three phases: randomly trying for achievements in the first phase, taking control of high valued areas and sending out explorers in the second phase, and trying to expand in the third phase.

The team claims that their buying algorithm has been detected in the qualification phase and a clever counter strategy was developed by another team that eventually led to the defeat.

TUB: The german team TUB, winner of several contests in the past, entered the contest for the 4th time (but with different team members). They use a centralized approach where agents share all their perceptions and intentions. It required 640 person hours (and 8000 lines of code)

First the agents probe and survey the whole graph. Explorers, attackers, repairers and inspectors only contribute to the zoning algorithm, if they have done their dedicated tasks. The team tries to find a balance between zoning and achievements points.

The team claims that they did not foresee very aggressive playing methods and that this led to several lost games.

LTI-USP: The motivation of the second brazilian team, (one professor and 2 students without previous experience in this scenario, was to test the JaCaMo framework (CArtAgO, Jason and Moise). They used a centralized approach for coordinating the agents and communication via speech-acts. 300 person-hours were invested and 3000 lines of code (a third in AgentSpeak, the rest in Java) were written.

The strategy was not to divide the game into phases but the agents into three subgroups: two for occupying zones and one for sabotaging the enemy. Communication with the server was through the EISMASSIM interface. The repairer agents stay where they are and wait until damaged agents come and see them. The sentinels always parry when an opponent saboteur is there and the own saboteurs always attack opponents in the same vertex.

No defense strategy has been implemented and the team claims that this was responsible for not doing better in the contest (zones were instable).

AiWYX: The chinese team consisted of just one person, a bachelor of science. He has a background in knowledge representation, game theories and distributed algorithms and used just plain C++. He invested ca. 250 person-hours and wrote 10000 lines of code. No agent programming technology was used at all, the system was centralized, all agents share their knowledge to build the map.

The strategy is to first go for areas where nobody else is and trying to expand them. If enemies attack, the agents draw back and look for better zones rather than attacking the enemies. Agents can dynamically change their behaviour at run-time. A big problem was that the agents did not attack the enemy team and that attacks from the enemy were not parried in a suitable way which resulted in instability of the zones.

PGIM: The iranian team consisted of one scientist and three students. They invested 8000 person-hours in total, using 7000 lines of code, to develop a decentralized system. After careful evaluation they chose Prometheus and

Jack. Due to licensing problems, they could not use Jack and had to redo all in Java. Due to some misunderstanding of the scenario, they chose to first attack and destroy the opponents repairer agent, then to attack other agents and only in the third place to consider building zones.
Instability of the zones and not being able to conquer zones of some value were the main drawbacks.

Streett: This team consisted of an american student who, unfortunately, did not provide us with any information about his team.

4 Interesting Simulations

In this section we analyse three of the most interesting games using our newly developed statistics module. This involves analysing the following charts: (1) summed-up scores, (2) zone scores and achievement scores, (3) zone stabilities.

The summed-up score consists of the achievement-score plus the zone-score. Note that the achievement score decreases, when the buy action is executed.

summed-up scores: This chart depicts the summed-up score of each team in each step of the current simulation.

zone scores and achievement scores: This chart combines the charts for the step-score (zone-scores + achievement-scores) and the achievement-scores. The zone-score derives from the number and value of the currently dominated nodes, while the achievement score sums up (across all categories) all the achievements so far.

zone stabilities: This chart depicts the zone stabilities of each team in each step of the current simulation. The zone stability increases for one team, if the team can hold all conquered nodes over a longer period of time. If nodes are lost, the value decreases. The exact computation is as follows: For each node that is dominated by a team in a certain step the counter is increased by one. If the team does not dominate the node anymore the counter is reset. The overall zone stability is then the sum of all node counter values.

4.1 SMADAS-UFSC vs. Python-DTU – Simulation 1

The first simulation of the match between SMADAS-UFSC and Python-DTU was a close victory for the winners of the contest, by 127.546 to 121.312. The complete visualization of the simulation can be downloaded from our webpage [9]. Both teams started even, with a very small edge to Python-DTU in the first few steps. Then, SMADAS-UFSC took over from step 35 until step 259. Python-DTU managed to recover the lead at that point for around 50 steps but with no considerable difference. Finally, SMADAS-UFSC took over again from step 309 until the end of the simulation, with a tendency to further increase the score difference. Figure 2, which shows the summed scores at each step, presents this visually.

[9] `http://www.multiagentcontest.org/downloads?func=fileinfo&id=1133`

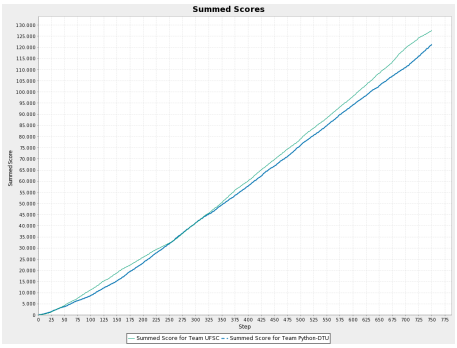

Fig. 2. SMADAS-UFSC vs. Python-DTU (Sim 1): Summed scores.

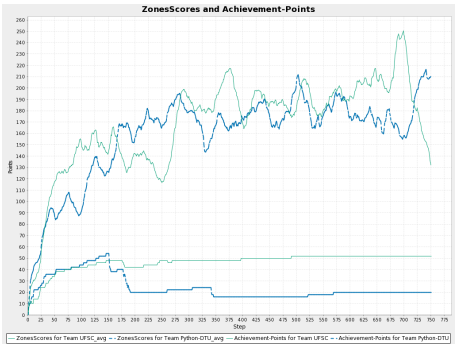

Fig. 3. SMADAS-UFSC vs. Python-DTU (Sim 1): Step-scores and Achievement points.

Figure 3 shows the *step-score* at each step (i.e., the value of the zone plus the unused achievement points at each step). To better display how the score is composed, also the unused achievement points at each step are displayed in the figure. Changes in step-score suggest that both teams attempted to conquer differentiated overlapping zones, as both teams maintained their zone value always above a relatively high minimum, but at several points in the graph the increase in the score for one team is correlated with a decrease in the opponent's score.

Achievements and Buying Strategy Also from Figure 3 it becomes clear that the difference in achievement points is much more significant than the difference in the total score. Even though Python-DTU had more valuable zones during most steps of the simulation, SMADAS-UFSC earned more points per step because of achievement points. The buying strategy proved to be crucial: the clever strategy implemented by SMADAS-UFSC, which consisted in buying improvements for only one of their saboteurs in an attempt to drive the other teams to spend more achievement points in more agents, worked perfectly in this case. Both teams earned the same number of achievements points: 68. But Python-DTU spent 48 of those points improving the saboteurs, whereas SMADAS-UFSC only used 16 for improving one of theirs. This meant a difference that at the end of the match was of 32 extra points per step for SMADAS-UFSC with little variations after step 350, which was not easily compensated by the zone-score. A point to remark here is that doubling the number of agents per team with regards to the previous edition of the Multi-Agent Programming Contest increased the efficacy of this strategy.

It is worth noticing that, while SMADAS-UFSC attempted to start their buying strategy as early as possible (and also to earn as many achievements as early as possible), Python-DTU's approach was to compensate for the aggressive buying strategy by delaying the first round of buys until step 150. Half of the 16 achievement points spent by SMADAS-UFSC were spent before step 10.

Their strategy also attempted to detect whether the other time was buying improvements to limit their own buys, and that explains the later buys at step 175.

Nevertheless, even when in general the buying strategy played in favor of UFSC-SMADAS, there seems to be a correlation between the first bulk of buys for Python-DTU at step 150 and an increase in their step scores. On the other hand, at that point of the simulation both teams were still scattered on the map and had not yet committed to defend a certain area.

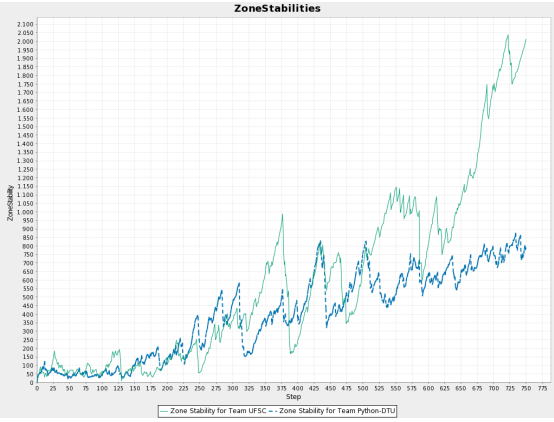

Fig. 4. SMADAS-UFSC vs. Python-DTU (Sim 1): Zones' Stability.

Zone Stability The *zone-stability*[10] graph in Figure 4 reaffirms the idea of overlapping but differentiated zones. Both teams' zone-stability have a clear tendency towards increasing, which means that a number of nodes remain unchallenged. At the same time, none of the zone-stability lines is smooth, which means that several nodes were being lost and recovered during simulation.

Two examples of area domination, one for each team, are presented in Figures 5 and 6. In Figure 5, at step 338 the value of the zone for Python-DTU was 223 and 140 for SMADAS-UFSC. In Figure 6, at step 417 those were respectively 160 and 219.

Actions per Role

[10] The *zone- stability* is a measure that increases when a team keeps dominance of a node, without taking into account the values of the nodes. It was designed for post-match analysis only, as it is not used for computing the scores.

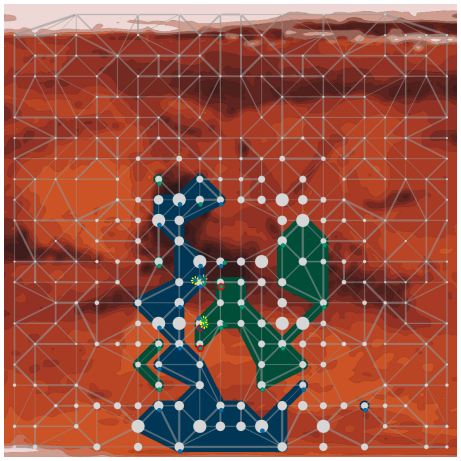

Fig. 5. SMADAS-UFSC vs. Python-DTU (Sim 1): Simulation after 338 steps.

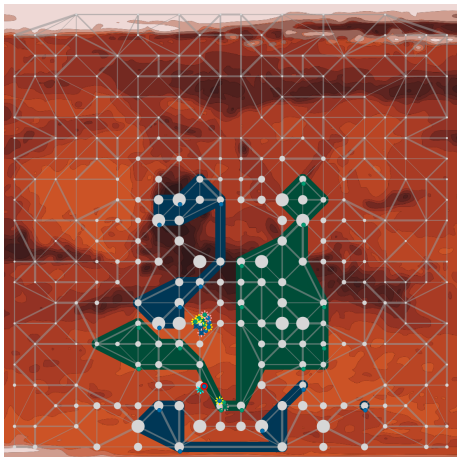

Fig. 6. SMADAS-UFSC vs. Python-DTU (Sim 1): Simulation after 417 steps.

SMADAS-UFSC. SMADAS-UFSC's *Explorers* used the `recharge` action the most, 55 percent of the times, followed by the `goto` action (35 percent). The `probe` action was used 303 times (10 percent), 302 of which were successful even though the map had only 300 vertices. The `survey` action was only used 16 times (less than 1 percent). The *Sentinels* executed the `recharge` action half of the times, followed by the `goto` action (38 percent). They also used the `parry` action 10 percent of the times and the `survey` action only 2 percent. The *Saboteurs* were quite aggressive, using the `attack` action in 51 percent of all cases (85 percent of the attacks were successful). The `recharge` action was used 32 percent of the times, And the `goto` action in only 16 percent of the cases, meaning they were somehow static. The `survey` action was also only used in less than 1 percent of the times (18) and the `buy` action, as mentioned before, was used 8 times. The *Repairers* executed `goto`, `recharge` and `repair` close to a third of the times each (39 percent, 30 percent, and 28 percent respectively). They also chose the `survey` action and the `parry` action around 1 percent of the times each. Finally, the *Inspectors* used mainly the `recharge` action (58 percent) followed by the `goto` action (38 percent). The `survey` action was used only 63 times (2 percent) and the `inspect` action even less, 33 times (1 percent).

Python-DTU. The *Explorers* from Python-DTU used the `recharge` action extensively, 75 percent of the times. The `goto` action, in contrast, was used 15 percent of the times. The `probe` action was used on 305 occasions (10 percent), of which 300 were successful (the number of vertices on the map). The `survey` action was used only in two occasions. The *Sentinels* also used the `recharge` action 75 percent the times. It was followed by the `parry` action, 13 percent of the times, although less than half of the parries were successful. They used the `goto` action even less than the Explorers, only 8 percent of the times. They also

used the `survey` action 5 percent of the times. The *Saboteurs* used the `attack` action 38 percent of the times (76 percent of the attacks were successful). The `recharge` and `goto` actions were used 30 percent of the times each. The `buy` action was used 24 times. They used the `survey` action only once. The *Repairers* executed the `goto` action 35 percent of the times and the `repair` action 34 percent. The third choice was the `recharge` action, 26 percent of the times. They opted for the `parry` action 83 times (3 percent, less than half of the parries were successful) and for the `survey` action 36 times (1 percent). Finally, the *Inspectors* used the `recharge` action the most (67 percent). They used the `inspect` action much more than they rivals (24 percent) and the `goto` action much less (9 percent). They only surveyed in 4 occasions.

4.2 SMADAS-UFSC vs. Python-DTU – Simulation 2

The second simulation of the match between the winners and runner-ups of the contest was won by the latter, by an even closer score of 120.450 to 115.076. Thus Python-DTU maintained the lead during the whole simulation, although SMADAS-UFSC reduced that difference to just 2.474 points at step 578. This is shown in Figure 7. The complete visualization of the simulation can be downloaded at our webpage [11].

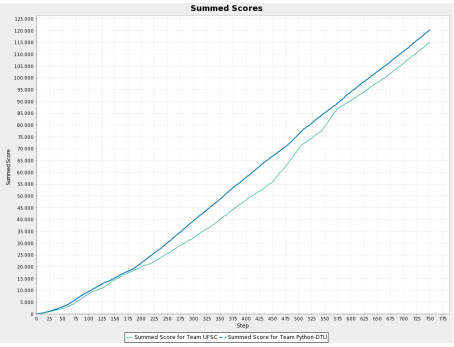

Fig. 7. SMADAS-UFSC vs. Python-DTU (Sim 2): Summed scores.

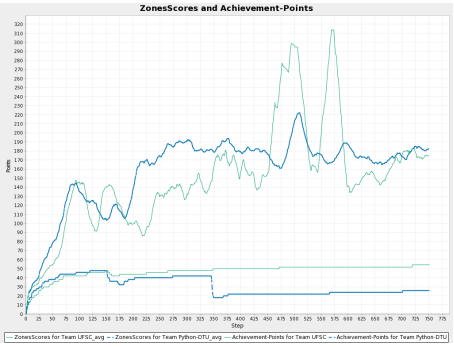

Fig. 8. SMADAS-UFSC vs. Python-DTU (Sim 2): Step-scores and Achievement points.

Zone Scores and Stability Figure 8 presents the Step-scores and achievement points at each step of simulation 2. In spite of the two high peaks in the score for SMADAS-UFSC, the advantage for Python-DTU was clear during most of the simulation.

[11] `http://www.multiagentcontest.org/downloads?func=fileinfo&id=1120`

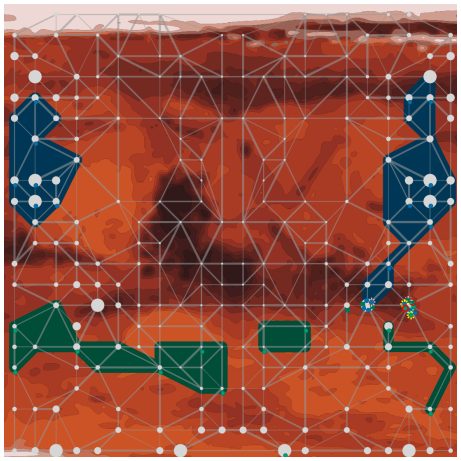

Fig. 9. SMADAS-UFSC vs. Python-DTU (Sim 2): Simulation after 362 steps.

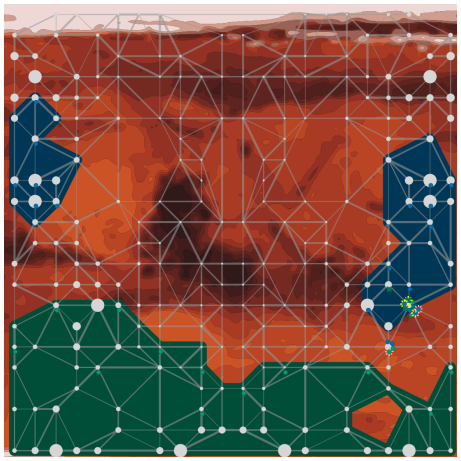

Fig. 10. SMADAS-UFSC vs. Python-DTU (Sim 2): Simulation after 481 steps.

The map in this simulation has different characteristics compared to the first simulation: The most valuable nodes were scattered towards the outer edges of the graph. A clear pattern of which zones each team would attempt to dominate and keep, did not emerge until around step 250. Two different moments during the simulation are presented in Figure 9, at step 362, where the value of the zone for Python-DTU was 176 and 64 for SMADAS-UFSC; and in Figure 10, at step 481, where the values were 172 and 243 respectively. Both figures exemplify what happened during the game, once the teams settled for a region of the map: Python-DTU conquered two zones far away from each other, and although those zones were not very big, they were very stable: In fact, one of the two remained practically unchanged during most of the simulation.

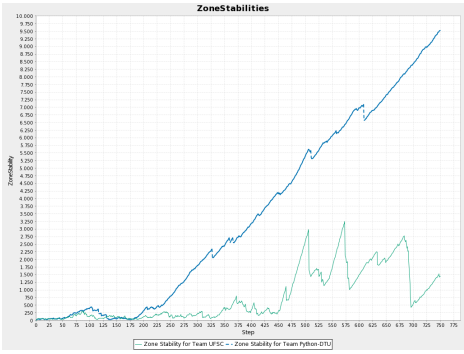

Fig. 11. SMADAS-UFSC vs. Python-DTU (Sim 2): Zones' Stability.

SMADAS-UFSC, on the other hand, managed to build the biggest and most valuable zone by isolating the bottom of the map. However, this was an unstable zone that they were not able to keep for a very long time. Furthermore, SMADAS-UFSC's agents were not standing on the most valuable nodes of that zone, so whenever the zone collapsed, those nodes were lost and thus the zone-score decreased significantly.

Figure 11 shows this difference with respect to zone-stability for each team. As zone-stability takes into account the number of nodes in the zones, the two peaks in the zone-score of SMADAS-UFSC are also slightly reflected in the zone-stability graph. Nevertheless, zone-stability for Python-DTU is still much higher.

Achievements and buying strategy During the second simulation, the buying strategy applied was the same as during the first one. This time, SMADAS-UFSC earned 68 achievement points and spent 14, whereas Python-DTU earned 66 and used 40. Nonetheless, as it can be seen in Figure 8, during this simulation the difference in achievement-points was not enough to compensate the difference in the zone-scores.

Actions per Role

SMADAS-UFSC. The *Explorers* of team SMADAS-UFSC used the `recharge` action in 61 percent of all cases, followed by `goto` (31 percent) and `probe` (8 percent). The `survey` action was only executed 10 times and the `buy` action was not used at all. Also, the *Sentinels* spend a lot of their time for recharging, i.e., the `recharge` action was used in 60 percent of all cases. Additionally, the main actions for this role were the `goto` action (31 percent) and the `parry` action (7 percent / 5 percent successful). Although the intended main purpose of the sentinel was to be used for surveying the edges the `survey` action was just used in 2 percent of the cases. Probably, because of the high visibility range of this role together with the information of the other roles these few executions were still enough. Finally, this type of agent did not `buy` anything. The behaviour of the *Saboteurs* was implemented in the following way. The `attack` command was executed 1302 times, i.e., in 43 percent of all cases, and was almost always successful (1123 times or 37 percent). The `recharge` action (37 percent) and the `goto` action (19 percent) were the second and third most used actions. The `survey` (25 times) and `buy` (7 times) action were only used sometimes, however the `buy` action was only used by this particular role. The main purpose of the *Repairers* was to go to some agents and repair them, therefore the `goto` (37 percent), the `recharge` (34 percent), and the `repair` (26 percent) action were used most often. The `survey` action was executed 42 times and the `parry` action 37 times (out of that 21 were successful). This is a huge difference to the Python-DTU Repairer that parried just one attack. Lastly, the *Inspectors* used mainly the `recharge` (72 percent) and `goto` action (25 percent). The `survey` action was used 53 times and `inspect` 20 times.

Python-DTU. The *Explorers* of team Python-DTU however used the `recharge` action extensively (more than 75 percent of all cases), followed by the `goto` action (14 percent) and `probe` action (8 percent). The `survey` and `buy` action were never used. The *Sentinels* executed the `recharge` action quite often (62 percent), followed by the `parry` (18 percent in total, but only 6 percent successful) and the `goto` action (12 percent). The `survey` action (7 percent) was only used seldom. The `buy` action was not used at all. The *Saboteurs* used the `attack` action in 39 percent of the cases. 33 percent were successful. A little bit less was the `recharge` action executed (33 percent in total / 30 percent successful). The `goto` action was applied in 27 out of hundred times. Additionally, this agent was the only one using the `buy` action. The action was used exactly 20 times, i.e., in 0.67 percent of the cases. Finally, the agent did not use the `survey` action once. The *Repairers* executed `goto` in 38 of the cases, followed by the `repair` (28 percent) and `recharge` action (33 percent in total / 31 percent successful). The `survey` action was used 17 times, the `parry` action just three times (out of that only one was successful) and the `buy` action was never executed. Finally, the *Inspectors* used mainly the `recharge` action (83 percent), followed by `inspect` (11 percent) and `goto` (5 percent). The `survey` action was executed 5 times and `buy` was never used.

4.3 PGIM vs. AiWYX – Simulation 1

The team *AiWYX* clearly won all simulations against *PGIM*. While the first simulation ended 81562 to 212016, the second resulted in 68748 to 107600 and the last in 75846 to 112466. The final position of *AiWYX* was 5 and *PGIM* got the 6th place.

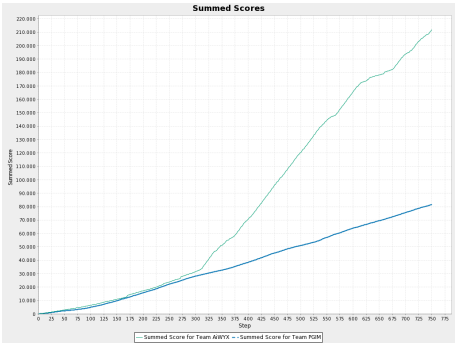

Fig. 12. PGIM vs. AiWYX (Sim 1): Summed scores.

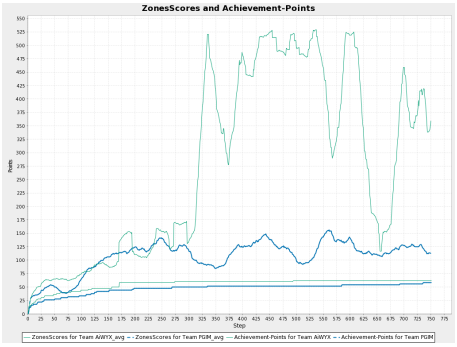

Fig. 13. PGIM vs. AiWYX (Sim 1): Step-scores and Achievement points.

During the beginning of the match both teams were at the same level. At step 170 *AiWYX* conquered an area of more than 640 nodes but was not able to

keep it for a longer period (cf. Figure 14). At step 312 *AiWYX* finally stabilized its zone(s) (cf. Figure 16 and 15). The team *PGIM*, however, was not able to conquer zones larger as 160 nodes and got therefore only the achievement for holding 80 nodes at the same time.

AiWYX used a novel strategy (not seen in the competition so far) for building zones: Instead of trying to conquer a small zone, probing the nodes in order to increase the value of the zone and finally defending, the team was positioning itself around an opponent's zone and thereby isolating the opponents zone from the rest of the graph. Figure 14 shows such a zone. At step 312 *AiWYX* finally stabilized its zone(s) (cf. Figure 15 and 16). As one can see this resulted in very large zones, basically containing all nodes the opponents did not conquer.

Nevertheless due to the lack of probing all conquered nodes the team *AiWYX* did not score all possible points but only a small subset. Additionally, the strategy was highly depending on the size of the map and more effective on larger maps. That is probably the reason why the team *AiWYX* scored the most points per simulation but did not reach a better place in the competition.

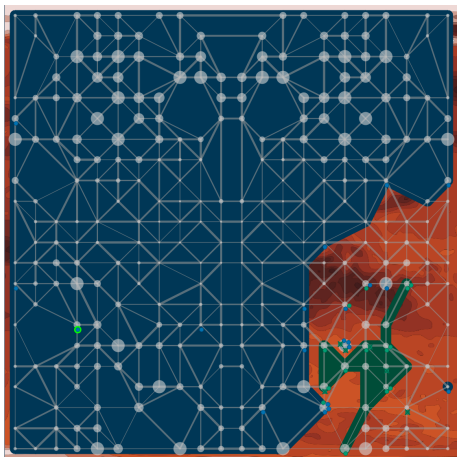

Fig. 14. Simulation after 170 steps.

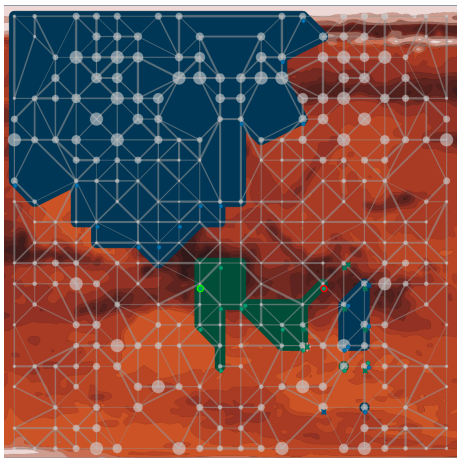

Fig. 15. Simulation after 312 steps.

The complete visualization of the simulation can be downloaded from our webpage [12]. In the following, we will discuss this simulation in more detail.

Scores The evolution of the zone scores and achievement points are depicted in Figure 13. While the development of the achievement points is similar (both teams did not invest the points for agent improvements), the flows of the zone scores are different. From step 0 to 300 it was a head to head competition but

[12] `http://www.multiagentcontest.org/downloads?func=fileinfo&id=1148`

after step 312 *AiWYX* was able to occupy a large zone and *PGIM* was not able to increase its zone score anymore.

Zone Stability The zone stability of team *PGIM* was low, i.e., under 500 points per step. In contrast, the zone stability of *AiWYX* was quite good and was almost always higher than that for *PGIM*. This is one reason why the team *AiWYX* won the match.

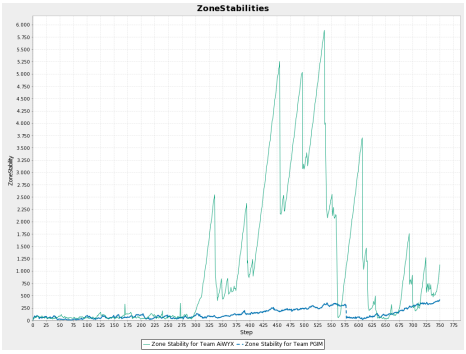

Fig. 16. PGIM vs. AiWYX (Sim 1): Zones' Stability

Achievements The team *AiWYX* conquered a zone with an impressive value of 640 points, attacked 640 times the opponents successfully, probed 160 nodes, and surveyed 640 edges. Additionally, It inspected 20 times an opponent. An interesting fact is that the agents did not try to parry an attack.

The team *PGIM* made the following highest achievements: It conquered an area of 80 nodes, attacked 320 successfully, probed 80 nodes and surveyed 640 edges. It inspected 10 times an opponent and parried 40 times attacks successfully.

Actions per Role

AiWYX. The *Explorers* of team *AiWYX* used the `recharge` action extensively (more than 50 percent of all cases), followed by the `goto` action (35 percent) and `probe` action (10 percent). The `survey` action was just used in just 1.7 percent. The *Sentinels* executed the `recharge` action quite often (53 percent), followed by the `goto` action (32 percent) and the `survey` action (4 percent). The *Saboteurs* used the `goto` action in 42 percent of the cases, followed by the `attack` (35 percent) and `recharge` action (22 percent). The *Repairers* executed `goto` in 54

the cases, followed by the `repair` (26 percent) and `recharge` action (18 percent). Finally, the *Inspectors* used mainly the `goto` (41 percent) and `recharge` action (56 percent). The `inspect` was just used 18 times (0.6 percent). `survey` was executed in 1.73 percent of the cases.

PGIM. The *Explorers* of team *PGIM* however used the `goto` action in 56 percent of all cases. 19 percent of the time they executed the `skip` action which does not have an effect. It would be more efficient to use the `recharge` action instead. This action was used in 11 percent of the cases. Finally, `probe` and `survey` were executed 8 and 5 percent of the times. The behaviour of the *Sentinels* was not optimal. The `skip` action was the most often used action (49 percent) followed by a `goto` command (37 percent). `parry` (2 percent), `survey` (4 percent), and `recharge` (8 percent) were just used seldom. Also the behaviour of the *Saboteurs* was not implemented in a good way. The `skip` action was used 1304 times, i.e., 43 percent of all cases although a `recharge` (13 percent) would be more efficient. The `goto` action was executed in 27 percent of all cases, followed by `survey` (3 percent) and `attack` (14 percent). For the *Repairers* the `goto` action was the main action (48 percent). This was followed by the `repair` (18 percent) and `recharge` action (21 percent). The `skip` action was executed 296 times, that corresponds to 10 percent. `survey` was used 84 times, i.e., 2,8 percent. The *Inspectors* used mainly the `goto` action (55 percent), followed by `skip` (26 percent) while `recharge` (14 percent) would be the better option. `survey` was used in 4 percent of the cases and `inspect` just 21 times (0,7 percent).

5 Summary, Conclusion and Future of the Contest

This paper provides an overview of the most recent edition of the Multi-Agent Programming Contest. We have introduced the Contest in general, and we elaborated on the current scenario in a more detailed way. We have also introduced the teams that took part and evaluated their performance. We compared three of the more interesting matches using our new visualisation and statistics modules.

This is our third newly designed scenario that we will also use, with some modifications and lessons learned from the 2012 edition, for the Contest in 2013. It is time to lean back and consider what we have achieved so far. *What conclusions (if any) can we draw from the "Agents on Mars" scenario? Can we observe some trends in the quality of the teams? What is the impact on the Pro-MAS community?* While these are critical and difficult questions that might be answered differently by different people, we collect a few observations that we consider relevant.

- Both times a dedicated Multi-Agent Programming Language/Platform won, but runner-up was Python-DTU, which did not use a dedicated platform, but was inspired by MAS technology.
  Nevertheless, other examples (e.g., the teams ranked 5–7 in this years edition) show that ad hoc implementations seem to perform worse than MAS inspired systems.

- The introduction of a qualification round increased the stability of the teams and therefore the whole contest a lot. This feature will be kept.
- Teams performing for the second time usually perform better. But the winners were both first time participants.
- The contest helped a lot to find bugs in the used platforms. This is an observation we made throughout the history of the contest. So it seems the scenario is demanding and most features of the used platform/language are indeed used (so that potential bugs surface). One team participated exactly because of this reason (testing their platform).
- We usually end up with as few as 7 to 9 teams that seriously want to participate. We believe this number could be much higher and does not really show a great impact on our community. On the other hand we have quite a variation: it is not always the same participants. Over the last 3 years, we had 20 different teams participating.
- The overall performance of the teams improved a lot with each new contest, although we increased the complexity considerably (size of the map, number of agents, difficulty of the task).
- Compared with the *cows and cowboys* scenario, we see much more cooperation among the agents, more dynamic behaviour, and a lot more interaction with the opposing team. In addition, the data to be handled (observing the environment, messages between the agents) has also increased a lot. While we have not yet excluded centralized approaches, the sheer amount of data makes it difficult for the systems to provide each agent with the central memory of the whole system.
  Also, in the current scenario, the computational costs of Dijkstra's algorithm is high so that it is not feasible for all agents to execute it at the same time.
- In the current scenario, there are indications that buying health and strength is much more important than investing the money for other reasons. Thus it may pay off to find a more balanced scenario that allows for more diverse strategies of the teams. This point makes us reconsider the precise values of the different parameter we have in our scenario.

The amount of work that went into implementing a team varied from one person with 250 person-hours to 6 people with 800 person hours and from 1500 to 10000 lines of code (the latter because no dedicated technology was used, interestingly, that was done by one single person).

It would be interesting to assess if it would be beneficial to steer the Contest into a more specialized direction in order to strengthen its niche in the research ecology. This includes but is not limited to focusing on the planning aspect of the competition, leaving behind path planning as the main facet of agent deliberation.

We could also focus on using a massive number of agents: lots of agents with different roles and thus different capabilities. This would allow us to take into account the scalability of agent-oriented programming platforms.

Additionally it would be worthwhile to focus on agent communication and to evaluate that aspect of the tournament by routing agent-messages through the *MASSim* server for proper evaluation.

Last but not least, the most important part of the contest are the contestants: We hope to attract more teams in the future — the contest is an excellent opportunity for a student project on Bachelor or Master level.

References

1. T. Behrens, M. Dastani, J. Dix, M. Köster, and P. Novák, editors. *Special Issue about Multi-Agent-Contest*, volume 59 of *Annals of Mathematics and Artificial Intelligence*. Springer, Netherlands, 2010.
2. Tristan Behrens, Mehdi Dastani, Jürgen Dix, Jomi Hübner, Michael Köster, Peter Novk, and Federico Schlesinger. The multi-agent programming contest. *AI Magazine*, to appear, 2013.
3. Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition - 4th edition. In *Proceedings of Sixth international Workshop on Programming Multi-Agent Systems, ProMAS'08*, volume 5442 of *LNAI*. Springer, 2008.
4. Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition: 4th edition. In Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors, *Programming Multi-Agent Systems, 6th International Workshop (ProMAS 2008)*, volume 5442 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2009.
5. Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. MAPC 2011 Documentation. Technical Report IfI-12-01, Clausthal University of Technology, December 2012.
6. Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. MAPC 2011 Evaluation and Team Descriptions. Technical Report IfI-12-02, Clausthal University of Technology, December 2012.
7. Tristan Behrens, Koen Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61:3–38, 2011.
8. Tristan Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner.
9. Mehdi Dastani, Jürgen Dix, and Peter Novák. The first contest on multi-agent systems based on computational logic. In Francesca Toni and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI*, volume 3900 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2005.
10. Mehdi Dastani, Jürgen Dix, and Peter Novák. The second contest on multi-agent systems based on computational logic. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems, 7th International Workshop, CLIMA VII*, volume 4371 of *Lecture Notes on Computer Science*, pages 266–283. Springer, 2006.
11. Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition - 3rd edition. In M. Dastani, A. Ricci, A. El Fallah Seghrouchni, and M. Winikoff, editors, *Proceedings of ProMAS '07, Revised Selected and Invited Papers*, number 4908 in Lecture Notes in Artificial Intelligence, Honululu, US, 2008. Springer.
12. Michael Köster, Federico Schlesinger, and Jürgen Dix. MAPC 2012 Evaluation and Team Descriptions. Technical Report IfI-13-01, Clausthal University of Technology, jan 2013.