Multi-Agent Programming Contest 2013

Tobias Ahlbrecht, Jürgen Dix, Michael Köster, and Federico Schlesinger

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

Abstract. This is about the ninth edition of the Multi-Agent Programming Contest[1], an annual, community-serving competition that attracts groups from all over the world. Our contest enables head-to-head comparison of multi-agent systems and supports educational efforts in the design and implementation of such systems. This year we have generated a multitude of statistical data for each match and give a detailed interpretation of them.

1 Introduction

In this paper we (1) briefly introduce the Contest, (2) elaborate on the 2013 scenario and its differences with the 2012 edition, (3) introduce the five teams that took part in the tournament, and (4) present many statistical data to interpret the matches and the performance of the teams.

The Multi-Agent Programming Contest [1] (MAPC) is 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 2013 the competition was organized and held for the ninth time.

More detailed information about the strategies of the teams are to be found in the subsequent five papers in this volume. In addition, we compiled a companian paper [1] that contains short answers from each team to more than 50 questions that allows the reader to easily compare the teams.

1.1 Related Work

For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to [2,7,5,6,10]. A quick non-technical overview appeared in [3].

[1] http://multiagentcontest.org

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], the *Planning Competition*[7], and the *General Game Playing*[8]. 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 2005–2013

Through the history of the Contest, changes to the scenarios were introduced with every new edition, with three major redesigns.

From 2005 to 2007, a classical *gold miners* scenario was used [8]. We introduced the *MASSim* platform: A platform for executing the Contest tournaments.

From 2008 to 2010 we developed the *cows and cowboys* scenario, which was 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, the use of cooperative strategies was a requirement of this scenario.

In 2011, the *agents on Mars* scenario [5] was newly introduced. In short, the environment topology was generalized to a weighted graph. Agents were expected to cooperatively establish a graph covering while standing their ground in an adversarial setting and reaching certain achievements. The basics of the *agents on Mars* scenario remained until the 2013 edition discussed in this paper, although several modifications were introduced to keep the Contest challenging.

2 MAPC 2013: Agents on Mars, Third Edition

For the 2013 edition of the Contest, a few significant modifications were made to the agents on Mars scenario used in 2012, in order to keep the challenge up to date. This section focuses on these modification; a more detailed description of the scenario can be found in Appendix A.

The number of agents in each team was increased again this year, to a total of 28 agents: 6 Explorers, 6 Repairers, 6 Sentinels, 6 Inspectors and only 4 Saboteurs. The 2012 edition comprised instead 4 agents of each role per team,

[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/
[8] http://games.stanford.edu/

whereas in the 2011 edition there were only 2 vehicles for each role, totaling 10 vehicles per team.

A big addition to the 2013 edition was the introduction of *ranged actions*. Agents could now act at a distance, i.e., having a target node that is different than the one where the agent stands (`probe`), or having a target agent that stands on a different node (`inspect`, `attack` and `repair`); these, as long as the target is within the visibility range. The successful execution of ranged actions depends on a probability factor that is based on both the distance to the target and the visibility range of the executor.

A slightly more subtle change was made to the map-generating algorithm, to get different (parametrized) levels of connectivity between the nodes that the teams should adapt to.

On a more general level, not concerning directly with the playability of the scenario, a lot of effort was invested in easing the development process to the participants, by means of improving the visualization tools, as well as the feedback sent to the agents. The new visualization lets the viewer distinguish at glance the roles of the agents, the last actions executed by each agent and their success/failure, the executor and target of ranged actions, and the nodes that were already probed by each team, all directly from the map.

3 The Tournament

Following the mode implemented in 2012, a *qualification round* was held prior to the tournament, 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 were they allowed to take part in the tournament.

3.1 Participants and Results

Five teams from around the world registered for the Contest and were able to pass the qualification round, thus taking part in the tournament (see Table 1).

Team	**Affiliation**	**Platform/Language**
AiWXX	Sun Yat-Sen University, China	C++
GOAL-DTU	Technical University of Denmark	GOAL
LTI-USP	University of Sao Paulo, Brazil	Jason, CArtAgO, Moise
SMADAS-UFSC	Federal University of Santa Catarina, Brazil	Jason, CArtAgO, Moise
TUB	TU Berlin, Germany	JIAC

Table 1. Participants of the 2013 Edition.

AiWXX: The team AiWXX [11] from Sun Yat-Sen University, China, took part in the contest for the second time, slightly changing its name (formerly AiWYX), and incorporating a second developer. 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.

LTI-USP: The team LTI-USP [9] from University of São Paulo, Brazil, also competed for the second time; this time with two developers, one less than in 2012. 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.

SMADAS-UFSC: The team SMADAS-UFSC [14] from Federal University of Santa Catarina, Brazil, was the winner of the 2012 edition. It had 7 team members (one more than in 2012). The language of choice for agent development was Jason combined with CArtAgO and Moise. Besides normal agent-communication provided by Jason, agents shared a common data-structure (blackboard) for storing the graph topology.

GOAL-DTU: The team GOAL-DTU [12] from the Technical University of Denmark is a regular contender of the Multi-Agent Programming Contest. This incarnation counted with 7 team members. The language of choice (as well as the team name) changed to GOAL for this edition, after having used a Python-based system for the previous two editions. The agents follow a decentralized approach, where coordination is achieved through distributed algorithms, e.g. for auction-based agreement.

TUB: The team TUB [13], Technical University Berlin, Germany, is another regular contender of the Multi-Agent Programming Contest, presenting this time a team with 12 members (originally working as two separate groups). The agents are developed in the JIAC V platform (which won the contest several times in previous years).

The tournament took place on the 9th and 10th of September, 2013. 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 12 pm and finished at around 6 pm. A match between two teams consisted of 3 simulations differing in the size and connectivity level of the graph: the first simulation was always 550 nodes with a *thinning factor*[9] of 10%, the second one 580 nodes with a thinning factor of 20%, and the third one 600 nodes with a thinning factor of 30%. Teams got 3 points for winning a simulation and 1 point in case of a draw. The results of this year's Contest are shown in Table 2.

All the participating teams of the 2013 edition had also participated in the 2012 edition (with a few different members in some cases), and the final results remained very similar, in spite of the modifications to the scenario and the new strategies implemented. SMADAS-UFSC was crowned champion for the second consecutive time, improving their previous year's performance and winning in every single simulation they took part in. GOAL-DTU was again a clear second,

[9]The *thinning factor* is a configuration parameter that is inversely proportional to the connectivity level of the graph.

Pos.	Team	Score	Difference	Points
1	SMADAS-UFSC	2702948 : 1455163	1247785	36
2	GOAL-DTU	2284575 : 1614711	669864	27
3	LTI-USP	2117299 : 2083105	34194	15
4	TUB	1412702 : 2238820	-826118	6
5	AiWXX	1516760 : 2642485	-1125725	6

Table 2. Results.

after winning every simulation except when they faced the Contest winners. LTI-USP obtained a respectable third place surpassing TUB and this was the only modification in the ranking of the five teams with respect to the 2012 edition. Both TUB and AiWXX got six points, so no team ended the Contest empty handed, but the difference in the simulation scores favoured the former to secure the fourth place.

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 team description articles[11,12,9,14,13] and to the joint paper[1] in these proceedings.

SMADAS-UFSC: The strategy can be divided into two phases: In the first phase the agents explore the map to obtain achievement points and to find good zones as early as possible. In this phase the agents try to build one big zone. If occupying such a zone is not possible in the first 130 steps the second phase is activated: The agents conquer the best nodes and try to protect several small zones. Additionally, the developers specified some special algorithms for building zones when the map has only a few connections.
While implementing the team the developers defined five different strategies and tested them with different maps against their team from last year and decided for the particular one right before the contest.
They claim that occupying several small zones was one of the main reasons why the performance was so good.

GOAL-DTU: The overall strategy was as follows: After around 70 steps one Explorer computed the best positions for the Sentinels and Inspectors to build a zone. After 150 steps the Explorer agents joined them. Saboteurs and Repairers were responsible for destroying the opponent's zones.
The team claims that their agents had two strong points: the ability to control a zone and the preemptive repairing, i.e., the Repairers anticipate an attack on a teammate and start repairing the agent right in that moment. One of the weak points was that the Saboteurs had an unresolved bug.

LTI-USP: The main strategy was basically the same as last year, namely, to divide the agents into three subgroups: two for occupying zones and one for sabotaging the enemy. However, the team organization was implemented in a different way. Instead of using the roles (like Explorer, Sentinel, etc.) from the scenario, additional roles with different strategies were defined. An agent

then could adopt a particular role and execute the associated strategies to fulfill her mission.

The team believes that a strong point of their implementation was a defensive strategy, resulting in more stable zones. The weak point was the size of the zones.

TUB: The strategy was twofold: Each agent followed its own strategy for collecting achievement points. Second, the team had a coordinator agent for computing and building zones. The agents had various roles they can take on, thus they could decide to help building a zone or to disturb the zone building of the opponent.

The authors claim that a strong point of their implementation was that the agents' strategies could be easily replaced due to the modular implementation. One of the main weak points was the zone building strategy.

AiWXX: The main strategy of the team was to probe the whole map first and then occupy several stable and valuable zones.

The team claims that one of the strong points was the computational speed of their pure C++ implementation. However, a weak point was that they did not take the actions of the opponent into account while developing the agents and therefore did not specify a defense or counter-strategy.

4 Overview of Teams' Performance

We collected a lot of data throughout the matches concerning the score and the zones (discussed in Section 4.1), the achievements (cf. Section 4.2) and the overall stability and reliability of the teams (see Section 4.3). Additionally, in Section 4.4 we analyse the behaviour of the agents regarding their roles. The underlying data can be downloaded from our web page[10].

4.1 Score, Zone Values, and Zone Stability

In this section we analyse for each team the overall performance (summed scores), the development of the achievement points and the zone stability.

All these values somehow depend on each other. The curves for the achievement points are usually quite flat but monotonically increasing: They could also, due to buying actions, decrease, but this does not really show in our curves. The reason is that this effect of buying is too small (i.e. the teams did not use it extensively) to have a visible effect.

To interpret the curves for zone stability, one has to take into account that *monotonically increasing* parts show that the zones are stable: the steeper it is, the more stable it is. Dually, if parts of the curve are *monotonically decreasing* this means that zones are attacked by the opponent and are unstable. So the derivative of these curves gives a better picture.

This behaviour follows from the computation of the the values in the chart: each node gets a counter that is initialised to 0 once the node belongs to a

[10] `http://multiagentcontest.org/downloads/Multi-Agent-Programming-Contest-2013`

zone. In each step, the counter is incremented if it still belongs to the zone. The counter is set to 0 if the zone does not exist anymore. The values depicted in the chart are the sum over all counters of all nodes.

SMADAS-UFSC The winner of our contest, won all matches and scored perfectly. GOAL-DTU came closest and the following figure shows the first simulation which was almost a draw.

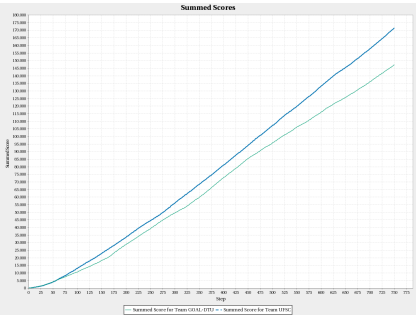

Fig. 1. UFSC-SMADAS vs. GOAL-DTU Simulation 1: Summed scores.

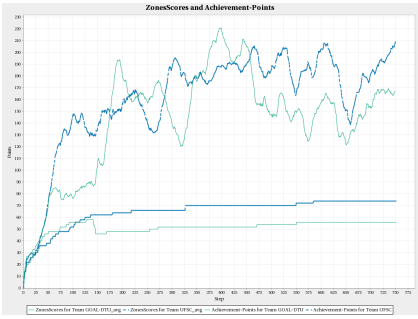

Fig. 2. UFSC-SMADAS vs. GOAL-DTU Simulation 1: Step-scores and Achievement points.

Fig. 3 shows that both teams were good in defending their zones and building up more (in the last third, SMADAS-UFSC was better in achieving this). The step scores were also narrow and oscillating between the two teams. The achievement points from SMADAS-UFSC were consistently better from early on in the match.

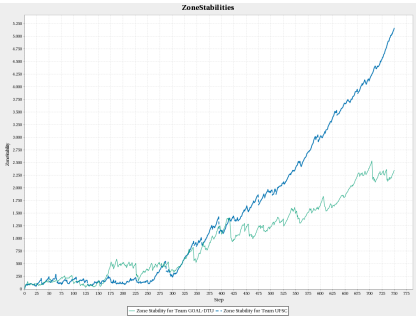

Fig. 3. UFSC-SMADAS vs. GOAL-DTU Simulation 1: Stability.

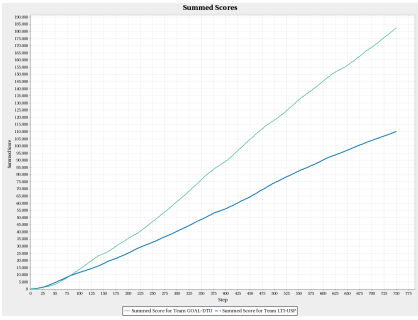

Fig. 4. GOAL-DTU vs. LTI-USP Simulation 2: Summed scores.

GOAL-DTU The runner up (for the third consecutive time) played very well and this also shows in the charts. As a typical example we chose the second simulation against LTI-USP. Fig 5 shows that green (GOAL-DTU) performed consistently better than LTI-USP, both in the achievements as well as in the step scores. This is even more immediate in the zone stability.

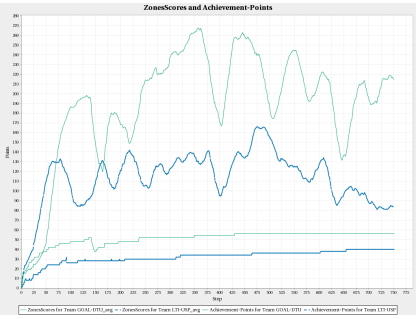

Fig. 5. GOAL-DTU vs. LTI-USP Simulation 2: Step-scores and Achievement points.

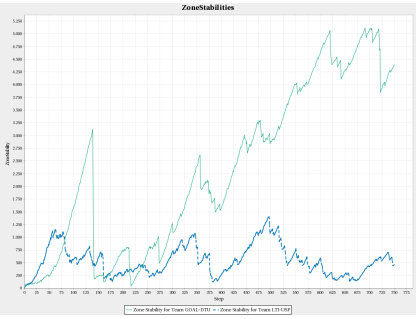

Fig. 6. GOAL-DTU vs. LTI-USP Simulation 2: Stability.

In the paragraph above about SMADAS-UFSC, we discussed the first simulation with GOAL-DTU which was a close match. Interestingly, GOAL-DTU did not as well in the other two simulations. There, its zone stability was not on par with its competitor.

LTI-USP The team came third but showed some strong playing.

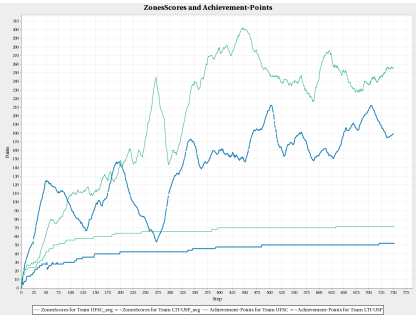

Fig. 7. LTI-USP vs. SMADAS-UFSC Simulation 3: Step-scores and Achievement points.

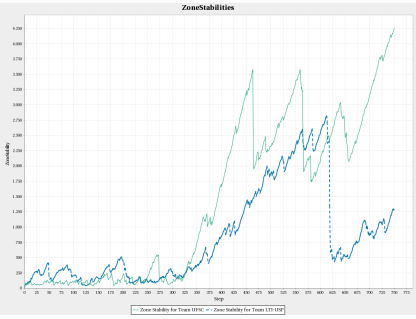

Fig. 8. LTI-USP vs. SMADAS-UFSC Simulation 3: Stability.

Figure 8 shows that zone stability worked well from the middle of the game to almost the end (LTI-USP is blue). Also the scores are well over the achievement points. A good result, only the opponent was a bit better in this match.

TUB The german team did usually quite poor on zone stability. The following figures show the first simulation of TUB against LTI-USP.

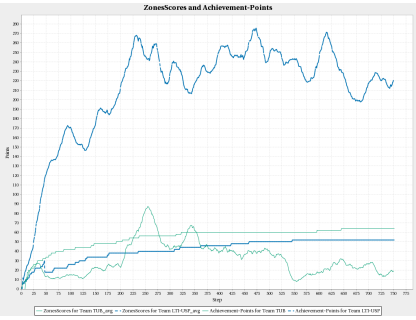

Fig. 9. TUB vs. LTI-USP Simulation 1: Step-scores and Achievement points.

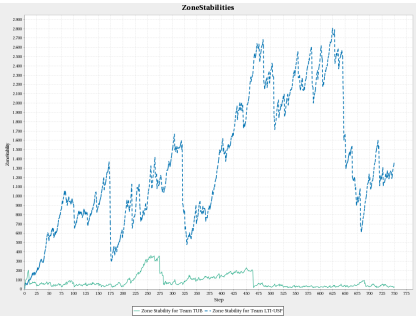

Fig. 10. TUB vs. LTI-USP Simulation 1: Stability.

In Fig. 9 one notices that the score of the green team (TUB), consists mainly of achievement points (the green curve is oscillating around or even under the achievements line). The poor performance on zone stability is of course clearly shown in Fig. 10.

However, TUB was doing much better in the third simulation against LTI-USP, where it almost drawed: So it did much better on the bigger scenario (which is usually more difficult to handle).

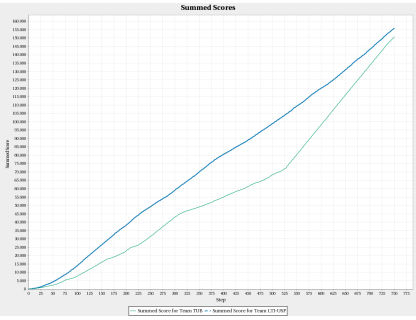

Fig. 11. TUB vs. LTI-USP Simulation 3: Summed scores.

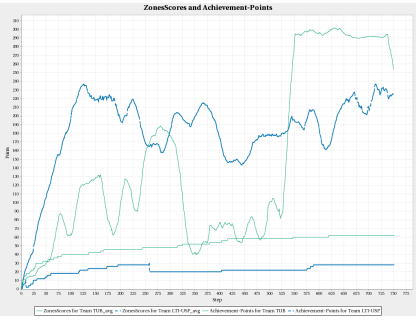

Fig. 12. TUB vs. LTI-USP Simulation 3: Step-scores and Achievement points.

AiWXX Although AiWXX came last, a few games were very close. The next figures show such a close match. The step scores in Fig. 13 show clearly that AiWXX was performing on par until the middle of the game, when the scores went quite dramatically down, and then up and down without stabilizing. The same is showing in Fig. 14: the team was not able to defend their zones.

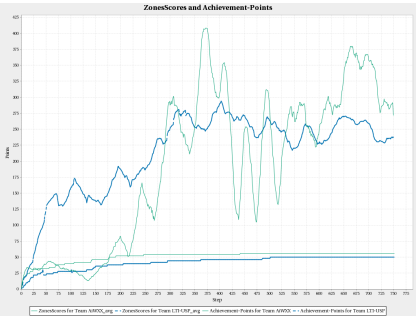

Fig. 13. AiWXX vs. LTI-USP Simulation 2: Step-scores and Achievement points.

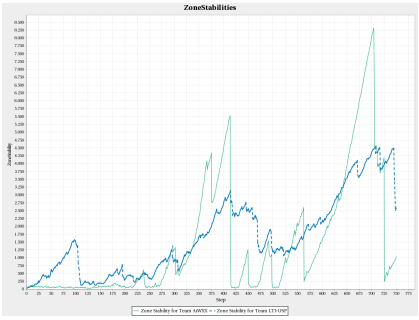

Fig. 14. AiWXX vs. LTI-USP Simulation 2: Stability.

Conclusion This year we saw some very interesting matches. Some of them were close to a draw. For some teams the building of zones did not work well when the opponent was too agressive. It worked better against other teams.

4.2 Achievements

In general, all teams gave more priority to achievement points *as part of the score* than as a resource to improve agents' attributes. In fact, only two of the five teams made use of the `buy` action at all, and they did it in a planned, limited, and consistent manner: GOAL-DTU spent 14 achievement points per simulation, and LTI-USP spent 20 points per simulation.

The average number of achievement points earned by each team, without taking the buys into consideration, is consistent with the final ranking of the contest: 71 points for SMADAS-UFSC, 70,5 for GOAL-DTU (56,5 after buys), 67,5 for LTI-USP (47,5 after buys), 64,5 for TUB and finally 56 for AiWXX. Interestingly, these numbers varied only a little for each team in the different simulations: they were not in any relation to the map sizes.

Figure 15 corresponds to Simulation 1 of the match between GOAL-DTU and LTI-USP, and reflects how the different buying strategies of these two teams affect the evolution of the achievement points. GOAL-DTU realizes all buys in a single phase that starts at around step 140, whereas 4 peaks[11] can be seen in

[11]The number of achievement point can only decrease through the buying action.

the graph for LTI-USP, the first one at the very beginning of the simulation and the last one prior to step 90. Even though at the end of this simulation LTI-USP manages to earn more achievement points than GOAL-DTU, it is clear that the achievement points of GOAL-DTU had more effect on the final score.

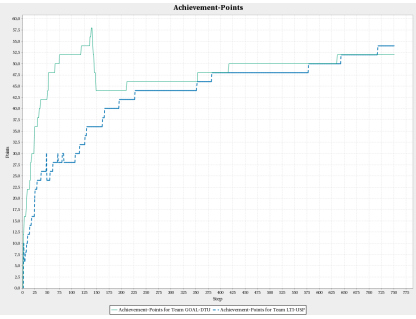

Fig. 15. GOAL-DTU vs. LTI-USP Simulation 1: Achievement Points.

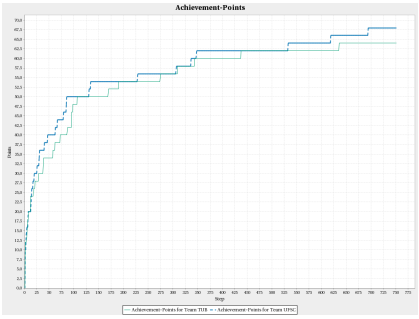

Fig. 16. UFSC vs. TUB Simulation 3: Achievement Points.

Figure 16 on the other hand, shows the typical evolution of the achievement points for teams that did not use the buying action. In these case, SMADAS-UFSC and TUB are almost on par, with a slight margin in favor of the former.

Regarding the composition of the achievements, it is worth noting the following:

- The `survey` action was the fastest to pay-off in terms of achievements, providing up to 10 achievement points in the very first step (up to `surveyed160`).
- The area achievements varied from simulation to simulation, but as expected, the best-performing teams were the ones who earned these achievements earlier.
- LTI-USP was markedly slower than the rest of the teams in probing more than 320 nodes (`probed320` achievement). The rest of the teams reached this number in general at around step 140, almost always before step 200. LTI-USP, on the other hand, never obtained this achievement before step 300, and often after step 400.
- The attack-related achievements were a major source of differences in the final count of achievement points earned. Aggressive teams, for example SMADAS-UFSC, performed better in this respect.
- Attack-related achievements were also the main source of achievement points in the end of each simulation.
- AiWXX's agents never used the `parry` action, and therefore never got any parry achievements.

Conclusion Differently from the previous edition, this year there were no simulations in which the achievement points played a significant role in the final

score. Although some teams performed better than others, the differences were much smaller than the differences in zone-score.

Whether it really pays off to implement a buying strategy, is not clear. This year's winners proved that a team can do very well without one. At the same time, the second and third ranked teams spent some of their achievement points in improvements, and clearly outperformed the fourth and fifth, even though the average achievement points remaining at the end of the simulations was better for the latter. For the two teams that did use the buying action, however, the strategy was rather conservative, and they kept most of the achievement points for scoring.

4.3 Agents' Reliability and Stability

In this section we analyse the success and failure of executing actions by the agents. The set of failure codes can be divided into three classes: a random failure, a technical failure, and a failure with rerspect to the semantics of the simulation.

While the first failure is introduced by the scenario[12] to ensure a certain degree of stability of the agents' perceive-think-act cycle (i.e. the agents are able to detect a failure and act accordingly), the second one is directly connected to the stability of the platform respectively to the agent program. If an agent is not able to send her action in a reasonable time slot then it can be only because of two reasons: the network communication was too slow or the agent had some problems due to a crash or some computational issues. Indeed, in this year's competition the participants did not have any network problems but some agents crashed during a run and had to be restarted and/or were using too much time for their computations[13].

The last class of failures is directly related to the game logic of the scenario. An `attack`-action can fail when the attacked agent executes the `parry`-action. A ranged action or `goto`-action can fail because the node or opponent is out of range. Even if the agent is in range, it can fail with a certain probability (determined by the visibility). Additionally, it fails in case of lack of resources, when the agent got successfully attacked or the status or role does not allow to execute a particular action. For the complete description of all actions and failure codes we refer to the scenario description in Appendix A.

For the reliability and stability of the agent we will focus on the following failure codes: We will look at all technical failures because they allow us to directly deduce some stability properties. On the semantical level we will consider the *out of range failures*, the *unreachable failures*, the *status* and *role failures* as well as the *resources failures*. These failures show that the agent did not respect her internal status or made some wrong conclusions regarding the environment and allow us therefore to speak about the reliability of the agent.

Finally, we will mention the other failures only if their occurrence is much higher than the average.

[12] For this year we let 1% of the actions fail randomly.

[13] The time limit was set to almost 4 seconds.

SMADAS-UFSC Concerning the stability we can conclude from the data that the SMADAS-UFSC agents were very stable. In total, only 12 actions were not sent in time. Interestingly, it was one action per simulation. More precisely, it was always the very same Inspector that did not send an action in the last step.

When it comes to reliability there are only very few failures because of lack of resources. Thus, we can say the agents were very reliable. However, one reason might be that the UFSC team did not use the ranged actions—a potential source of error—a lot.

Typical results for SMADAS-UFSC are shown in Figure 17 and Figure 19.

Reason	SMADAS-UFSC	%	LTI-USP	%
parried	812	3,87	514	2,45
out of range			69	0,33
random	206	0,98	226	1,08
resources	1	0	13	0,06
attacked	285	1,36	191	0,91
no action received	1	0		
status			1	0
in range			1187	5,65

Fig. 17. LTI-USP vs. SMADAS-UFSC Simulation 1: Failed Actions.

GOAL-DTU The GOAL-DTU agents were also very stable. Around 0.5 percent of the actions got lost due to computational issues and the agents did not crash at all. From the scenario perspective the GOAL-DTU team made more mistakes than the SMADAS-UFSC team. One reason for this was caused by the use of ranged actions. Nevertheless the team was robust and did not try to execute an action forbidden by the role or the current status. Figure 18 contains some exemplary data of one simulation.

Reason	TUB	%	GOAL-DTU	%
parried	286	1,36	3	0,01
out of range	8	0,04	4	0,02
random	203	0,97	231	1,1
resources	4	0,02		
unreachable	573	2,73		
attacked	138	0,66	259	1,23
no action received			19	0,09
status	17	0,08		
in range	52	0,25	118	0,56

Fig. 18. TUB vs. GOAL-DTU Simulation 2: Failed Actions.

LTI-USP The stability of LTI-USP was in between the first two teams. While SMADAS-UFSC was the most stable team in the field the LTI-USP was following closely afterwards. In only two simulations (Simulation 3 against AiWXX and Simulation 3 against GOAL-DTU) LTI-USP had some stability issues. Less than 0.1% and 0.8% respectively of actions failed due to that.

Regarding the failures depending on the scenario we can say that the number of failures due to the lack of resources and the out of range actions was comparable with the ones from GOAL-DTU, however the number of actions that failed in range was significantly higher (around 5 Percent). Buying more visibility range for the agents would have decreased that value.

Figure 17 shows an example.

TUB The TUB team's stability was similar to that from GOAL-DTU. In some simulations the agents did not loose one action in others they lost some (but always not more than 1%). Thus the agents were stable and answering normally in time.

When it comes to the reliability of the agents' code we noticed some differences to the first three teams. The agents often tried to go to a node that was not reachable from their position. This was especially the case in Simulation 1 against GOAL-DTU. More than 6% of the actions returned that failure. Also, some actions failed due to the status. Concerning the ranged actions and the resources the results are comparable with GOAL-DTU.

Figure 18 depicts a typical result for TUB.

AiWXX Finally, the stability of AiWXX (Example shown in Figure 19) was the worst in the contest although (except for one simulation against LTI-USP where the computer or the agents crashed) it was still in the range of 2 to 5 percent and therefore quite good.

The reliability was as for the other teams. One thing we noticed was that quite some actions failed due to an attack of the opponent. So it might be that a better strategy for parrying or avoiding attacks would have helped to get a better position in the final ranking.

Reason	AiWXX	%	SMADAS-UFSC	%
parried	52	0,25		
random	237	1,13	201	0,96
unreachable	3	0,01		
attacked	261	1,24	39	0,19
no action received	1120	5,33	1	0
status	4	0,02		
in range	7	0,03		

Fig. 19. AiWXX vs. SMADAS-UFSC Simulation 1: Failed Actions.

Conclusion In summary, we can say that this year all teams were stable and reasonable reliable. This was expectable since we only slightly changed the scenario in the last two years and all teams from this year were participating last year as well.

4.4 Actions per Role

In this section we take a look at the frequency at which actions are executed per agent role and team. For a description of the agents' roles and their respective available actions we refer to Appendix A. Sometimes, we shall mention the percentage of failed actions on a *per role* and a *per team* basis. For a more general perspective of failed actions per team only, we refer to Section 4.3.

Explorer The Explorer role's inherent task is to scout the map and `probe` nodes to get information about their value.

Comparing all teams, the actions `goto` and `recharge` are dominant over all others. Most of the teams (all except AiWXX) execute a similar number of `probe` actions in all simulations. Although maps of different sizes are played (550, 580 and 600 nodes each), the number of executed `probe` actions does not increase proportionally and at times even decreases for those teams. Also, no Explorer used the `buy` action and thus nobody was able to execute a ranged probing.

SMADAS-UFSC: This teams' Explorers did not use the `survey` action at all. Apart from this, the amount of probing was in line with most of the other teams and settling down around 13% in each simulation. Most actions were `goto` and `recharge`, however, neither one dominates the other in all simulations.

GOAL-DTU: This team's Explorers used the least `probe` actions (directly followed by LTI-USP), peaking below 10%. The amount of `survey` actions is negligible and the most used action was `recharge` at 75% to 80%. From this we can deduce that these Explorers seemingly always explored an equally sized portion of the map. Since only a small percentage is left for the `goto` action, we can further assume -also based on the overall outcome- that suitable zones were found swiftly and could be held for a long time. A characteristic performance of these Explorers is given in Figure 20. Each bar represents one action that is available to the role. They allow for analyzing how often the respective actions were used by the agents of the current role and team. The green colored part indicates how many actions were successful while the red part represents the failed actions. Above each bar are a couple of numbers. The blue ones describe the total amount (in absolute and relative numbers) of usages of the action. Accordingly, the green numbers below describe the successful actions.

LTI-USP: Everything said in the previous paragraph also applies for these Explorers. The only difference to GOAL-DTU is the amount of `goto` actions which ranges from 17% to 31% for LTI-USP. The behaviour of this team was also very uniform over all simulations.

TUB: The TUB Explorers were the only ones to use an observable amount of `skip` actions (which also holds for every other role of TUB). Usage of the

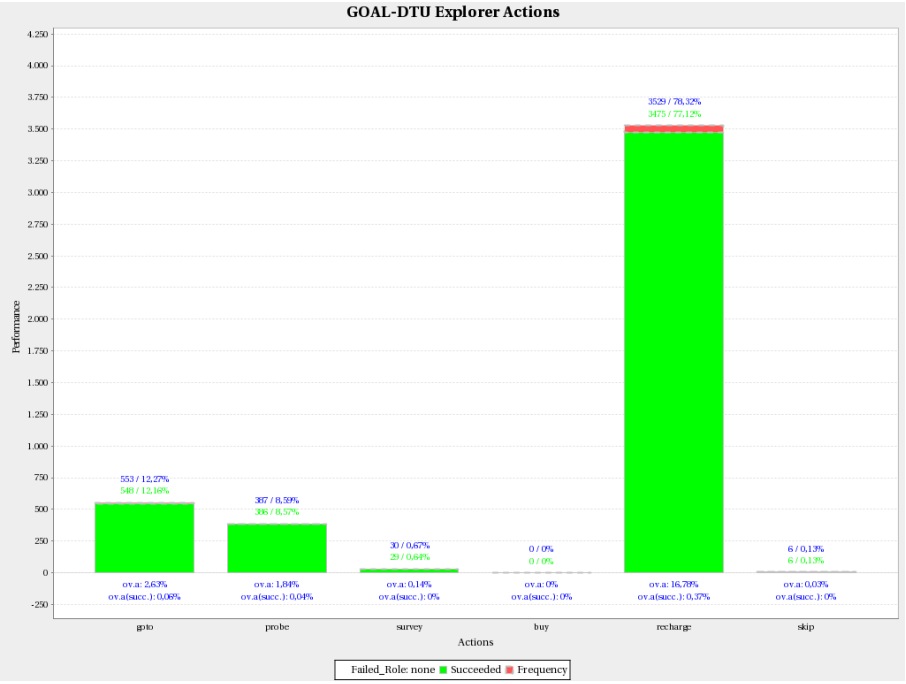

Fig. 20. AiWXX vs. GOAL-DTU Simulation 1: GOAL-DTU Explorer Actions.

`recharge` action might have proven to be a better alternative, however, there was no case of unexceptionally many 'failed resources' failures for TUB. The relative number of `probe` and `survey` actions was uniform for all simulations. However, one simulation showed a large number of failed `survey` actions. Nevertheless, the number of successful `survey` actions in this simulation is comparable to that of the other simulations.

AiWXX: Their Explorers used the `probe` action to a varying degree ranging from 6% to as much as 30% which is the peak percentage of all teams. Besides some simulations, in which they used the `survey` action more than every other team, the majority of actions falls upon `goto` and `recharge`. However, there is no clear favorite between these two actions regarding all simulations. This points to a varied degree of mobility that is neither dependent on the opponent nor the size of the map.

Inspector The Inspector is the only role that is able to inspect, that is to gain information about agents of the other team aside from their observable properties.

The teams used the `inspect` action to a varying degree. However, SMADAS-UFSC, GOAL-DTU and LTI-USP show a similar performance (of actions) over all simulations.

SMADAS-UFSC, GOAL-DTU, LTI-USP:.These Inspectors used the `survey` and `inspect` actions a negligible amount of times. Of these `inspect` actions, only those of SMADAS-UFSC are mostly succeeding while those of the others fail in approximately 2 out of 3 cases. The remaining actions are divided between `goto` and `recharge` with `recharge` clearly dominating. From this we can derive that the Inspectors were mainly used to occupy zones neglecting their special feature. As an example, we refer to Figure 21, which looks quite similar to all other simulations of these three teams.

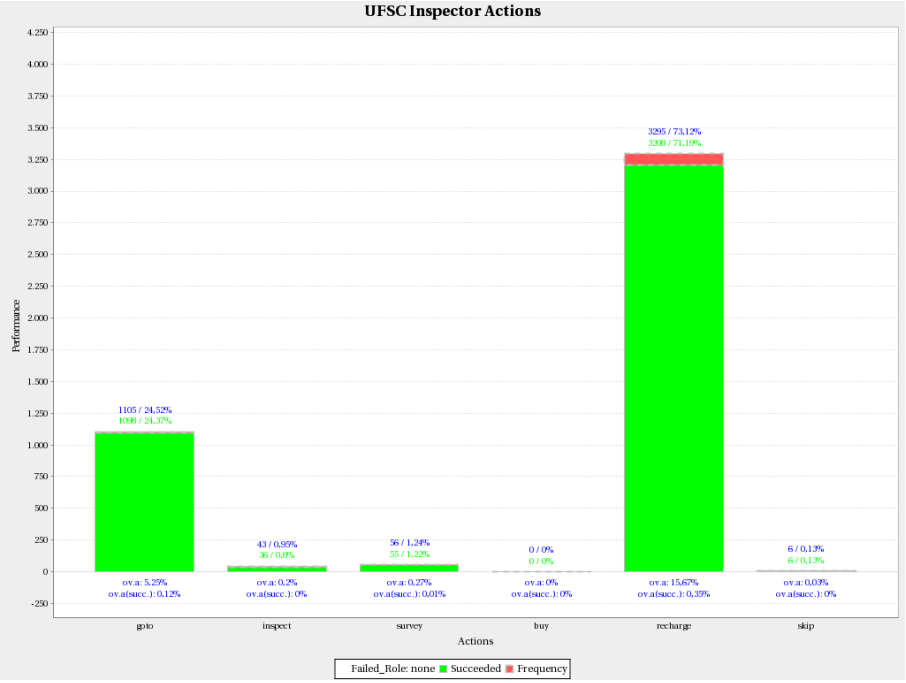

Fig. 21. LTI-USP vs. SMADAS-UFSC Simulation 2: SMADAS-UFSC Inspector Actions.

TUB: The TUB Inspectors used the `inspect` action a lot more, ranging from 15% to 55%. In addition, these were mostly successful (i.e. more than 75% in the worst case). Another distinction is the amount of `goto` actions dwarfing the number of `recharge` actions. However, only these agents had a tendency to fail at using this `goto` action making up for the increased usage.

AiWXX: These Inspectors used the `inspect` action only at 1-3% of times, thus falling in line with every other team but TUB. The `survey` action was used in 1-15% of steps and the remaining numbers of `goto` and `recharge` actions were alternating over simulations, which differs from all other teams.

Repairer The Repairer is able to enable agents which have been disabled by attacks from other teams. As this strongly depends on the performance of the competitor, there is no uniform behavior over all simulations.

SMADAS-UFSC: This team was the one to use the `repair` action the least. Aside from this, the `survey` and `parry` actions were used a few times leaving the `goto` and `recharge` actions again with the greatest number of executions. The latter actions were mostly used equally with no action dwarfing the other. These Repairers showed a uniform performance over all simulations.

GOAL-DTU: The Repairers used the `survey` action more than the average. The agents also parried the most. However, most of the `parry` actions failed. The repairing ranged from 5% to 30% and most `repair` actions were successful.

LTI-USP: Their Repairers used 6 `buy` actions per simulation on average. The `recharge` action was used at varying amounts, in one simulation even peaking at 85%. The `repair` action was mostly used a lot, however, less than 50% of these uses were successful. If the agents repaired more, the `recharge` action was used less (probably only being the default action).

TUB: These Repairers did not parry at all. The `repair` action was used in 5% up to 40% of steps and mostly succeeded. The `repair` and `recharge` actions were alternating similar to LTI-USP.

AiWXX: The AiWXX Repairers used the `repair` action the most. At times it was used in more than 60% of steps and mostly successful. These agents also did not parry at all (so, of course requiring more repairs). An example can be seen in Figure 22.

Saboteur The Saboteur is opposite to the Repairer, being able to disable other agents if they do not parry.

Three of the teams did not make use of the `buy` action. However, those who did were not affected by a higher percentage of failures in general.

SMADAS-UFSC: This team did not buy anything for the Saboteurs. The `attack` action was used 25% to 50% and the success percentage depended on the respective opponent. It was used more often than the `recharge` action.

GOAL-DTU: These agents used the `buy` action 7 times on average. An effect of this is not reflected in the charts. The `attack` action was used in 20% to 50% of steps and again failed according to the respective competitor. The `survey` and `parry` actions were ignored.

LTI-USP: These Saboteurs used 4 to 5 `buy` actions per simulation. The `attack` action was used in 15% to 55% of steps and failed quite often, except in one single simulation. The independence of the opponent is possibly due to ranged attacks that were not used by many other teams. Buying more visibility range would have increased the number of successful attacks.

TUB: The TUB team did not use the `buy` action. The `attack` action was used in 5% to 50% of steps. Slightly distinctive, the percentage of failures did not vary per opponent but per simulation. An example can be seen in Figure 23.

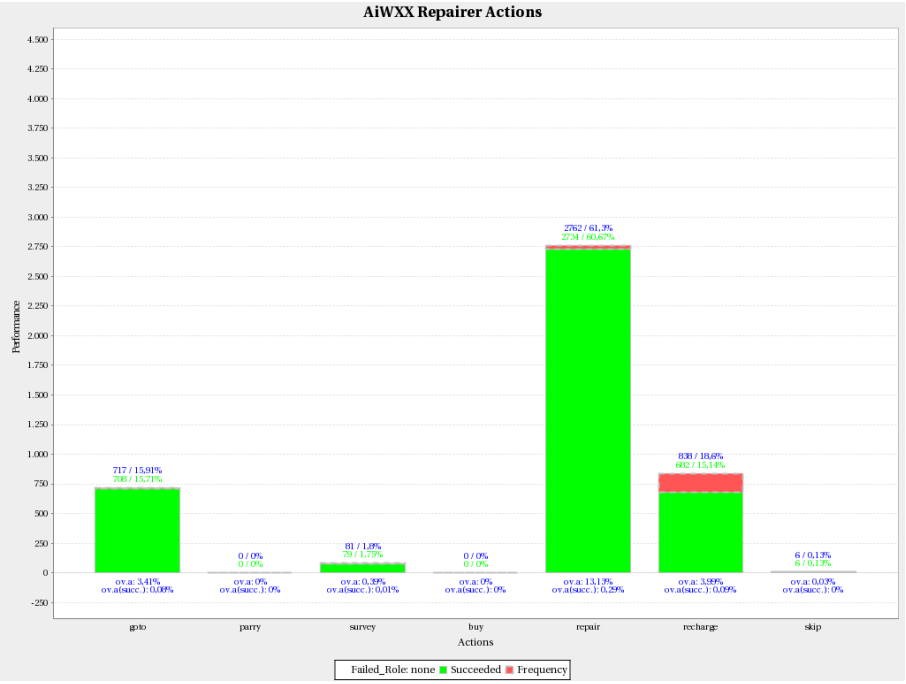

Fig. 22. AiWXX vs. GOAL-DTU Simulation 3: AiWXX Explorer Actions.

AiWXX: The `attack` action was used in 5% to 40% of steps without using a bought upgrade. Similar to TUB, the failure percentage differed per simulation and not per opponent.

Sentinel The Sentinel role is best suited to defend a zone, since it can use the `parry` action and has no other distinctive characteristic.

SMADAS-UFSC: This team parried in 12% of steps while succeeding at around 75% of these actions. The dominant actions here were `goto` and `recharge` with the latter occurring more often.

GOAL-DTU: This team parried more often, ranging from 3% to 30% of possible executions. However, the Sentinels were mostly succeeding in less than 50% of these actions. This might be a sign of increased pre-emptive parrying. The most used action again was `recharge` at 60-80%. This again underlines the tendency of GOAL-DTU to use the fewest `goto` actions.

LTI-USP: These Sentinels used the `parry` action in 1 to 17% of steps. A relation to the map size is not in evidence, however, exceptionally many `parry` actions were used in the match against SMADAS-UFSC. An example of such a match is given in Figure 24. This might be due to their Saboteurs being the most aggressive ones in using the `attack` action against LTI-USP and shows a certain degree of flexibility in adapting to the amount of incoming attacks.

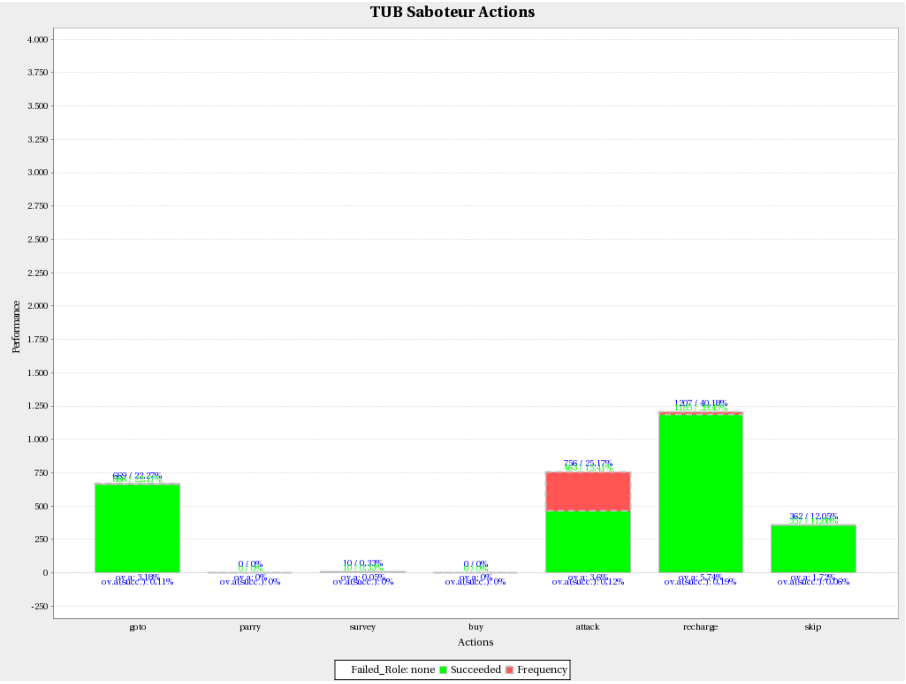

Fig. 23. TUB vs. GOAL-DTU Simulation 2: TUB Explorer Actions.

TUB: The TUB agents were again the only ones to use the `skip` action. The `parry` action was only used in 1-6% of steps and mostly failed. Also, the agents used the `survey` action in 1% of steps. In two occasions, the percentage was 5% and 15% respectively, however, the successful `survey` actions still made up only 1% of the total actions.

AiWXX: The `parry` action was not used at all. The agents performed a small amount of `survey` actions and otherwise used the `recharge` and `goto` actions in varying proportions.

Conclusion We have seen that the teams did not use the actions as diverse as one could have expected. For some teams and roles, the proportions of actions were very similar. However, some teams (mostly the ones coming 4th and 5th) showed completely different behavior. Also, some teams showed to behave similar over all simulations while others varied more with respect to using the available actions.

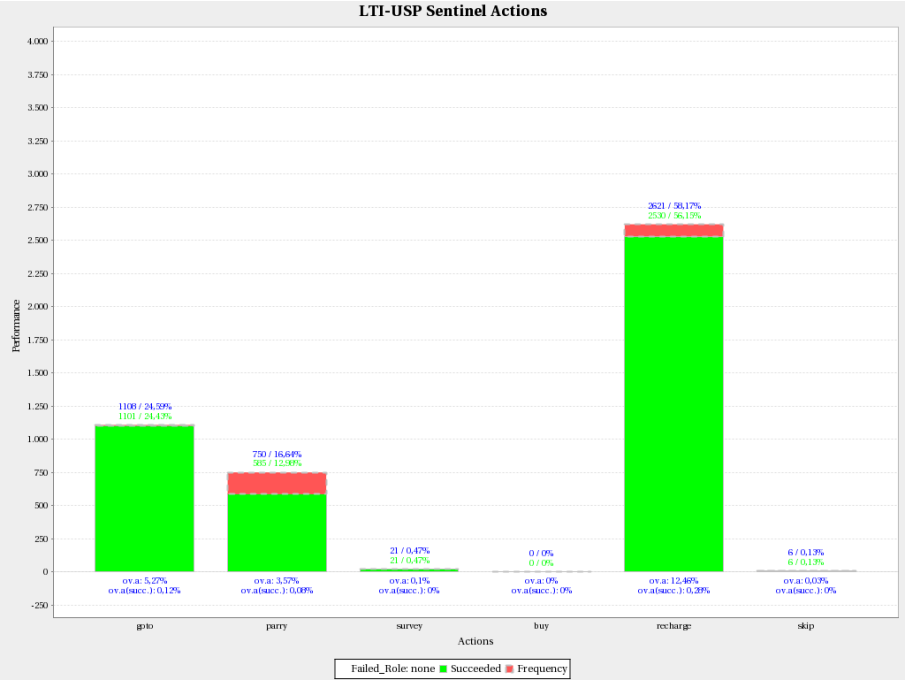

Fig. 24. LTI-USP vs. SMADAS-UFSC Simulation 1: LTI-USP Explorer Actions.

5 Summary, Conclusion and Future of the Contest

This paper provides an overview of the most recent edition (2013) of the Multi-Agent Programming Contest. We introduced the Contest in general, and we elaborated on the current scenario, with an emphasis on the changes to the last edition in 2012.

In this year, we had a plethora of statistical data available that we carefully analysed in the sections above. In a companion paper, [1], we collected the answers to 50 questions posed to the teams. They are arranged in a way to facilitate the comparison of the teams.

Here are a few observations, not just for this edition, but for the last three (where we introduced the Mars scenario).

- In all three editions a dedicated Multi-Agent Programming language or platform won.
- The runner-up in all three editions was the team headed by Jørgen Villadsen (DTU). For the first two editions they used Python, for the third one GOAL (a dedicated agent programming language which also won the first edition).
- We believe it is fair to say (taking all the results into account) 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. We shall keep this feature.
- Teams performing for the second time usually perform better. But all teams performed in previous editions (sometimes only the team leader remained and started with a new crew).
- The overall performance of the teams is improving with each new contest, although we increased the complexity considerably (size of the map, number of agents, difficulty of the task).
- Some teams were playing well when the opponent was not too agressive, but they played very bad when the opponent attacked them.
- Only two teams (placed second and third) used the buy-actions and invested money to improve agents. All others used achievements solely to improve the overall score. The part of the score related to achievements did not play a major role.
- Only one team, placed last, showed slight problems with the stability of the agents. Otherwise this did not play any role.
- Only the team placed last did not use any *parry* action (to defend a zone).
- 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 for the whole system.

  Also, in the current scenario, the computational costs of shortest path finding is high so that it is not feasible for all agents to execute it at the same time.

How can we make the contest even more exciting?

Agents: Why not using a massive number of agents: many agents with different roles and thus different capabilities. Not just 10-30, but hundreds of them. This would allow us to take into account the *scalability* of agent-oriented programming platforms.

Uncertainty: Up to now our environments were pretty observable, the amount of failing actions or wrong sensors was small. This could be changed to more indeterministic environments, where agents have to find out the effects of their actions.

Communication: It might also 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: This year, three teams started as student projects.

We hope to attract more teams and students in the future: the contest is an excellent opportunity to learn about multi-agent systems.

Acknowledgements

We would like to thank Alfred Hofmann from Springer for his support right from the beginning and for endowing the price of 500 Euro in Springer books.

References

1. Tobias Ahlbrecht, Christian Bender-Saebelkampf, Maiquel de Brito, Nicolai Christian Christensen, Jürgen Dix, Mariana Ramos Franco, Hendrik Heller, Andreas Viktor Hess, Axel Hessler, Jomi F. Hübner, Andreas Schmidt Jensen, Jannick Boese Johnsen, Michael Köster, Chengqian Li, Lu Liu, Marcelo M. Morato, Philip Bratt Ørum, Federico Schlesinger, Tiago L. Schmitz, Jaime Simão Sichman, Kaio S. de Souza, Daniela M. Uez, Jørgen Villadsen, Sebastian Werner, Øyvind Grønland Woller, and Maicon R. Zatelli. Multi-Agent Programming Contest 2013: The Teams and the Design of their System. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 372–396. Springer, 2013.
2. 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.
3. Tristan Behrens, Mehdi Dastani, Jürgen Dix, Jomi Hübner, Michael Köster, Peter Novák, and Federico Schlesinger. The multi-agent programming contest. *AI Magazine*, 33(4):111–113, 2012.
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, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner. The Multi-agent Programming Contest 2011: A Résumé. In Louise Dennis, Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *Lecture Notes in Computer Science*, pages 155–172. Springer Berlin / Heidelberg, 2012.
8. 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.
9. Mariana Ramos Franco and Jaime Simão Sichman. Improving the LTI-USP Team: A New JaCaMo based MAS for the MAPC 2013. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 344–353. Springer, 2013.
10. Michael Köster, Federico Schlesinger, and Jürgen Dix. The multi-agent programming contest 2012. In Mehdi Dastani, Jomi F. Hübner, and Brian Logan, editors, *Programming Multi-Agent Systems*, volume 7837 of *Lecture Notes in Computer Science*, pages 174–195. Springer Berlin Heidelberg, 2013.
11. Chengqian Li and Lu Liu. Prior State Reasoning in Multi-agent systems and Graph-Theoretical Algorithms. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 362–371. Springer, 2013.
12. Jørgen Villadsen, Andreas Schmidt Jensen, Nicolai Christian Christensen, Andreas Viktor Hess, Jannick Boese Johnsen, Øyvind Grønland Woller, and Philip Bratt Ørum. Engineering a Multi-Agent System in GOAL. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 334–343. Springer, 2013.
13. Sebastian Werner, Christian Bender-Saebelkampf, Hendrik Heller, and Axel Heßler. Multi-Agent Programming Contest 2013: TUB Team Description. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 354–361. Springer, 2013.
14. Maicon R. Zatelli, Maiquel de Brito, Tiago L. Schmitz, Marcelo M. Morato, Kaio S. de Souza, Daniela M. Uez, and Jomi F. Hübner. SMADAS: A Team for MAPC Considering the Organization and the Environment as First-class Abstractions. In Michael Winikoff, Amal El-Fallah Segrouchni, and Massimo Cossentini, editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 324–333. Springer, 2013.

A Scenario Description

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

> *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.*

A.1 The Map

The environment's topology is constituted by a weighted graph. Each edge has a weight, which is a number that represents the costs of moving from one of its vertices to the other. Each vertex has a unique identifier and a value indicated by a number from 1 to 10. The vertices' 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, such that there are no rival agents inside of it, all the nodes that are inside the 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 25 shows a screenshot of a relatively small map, depicting, amongst other things, the graph coloring.

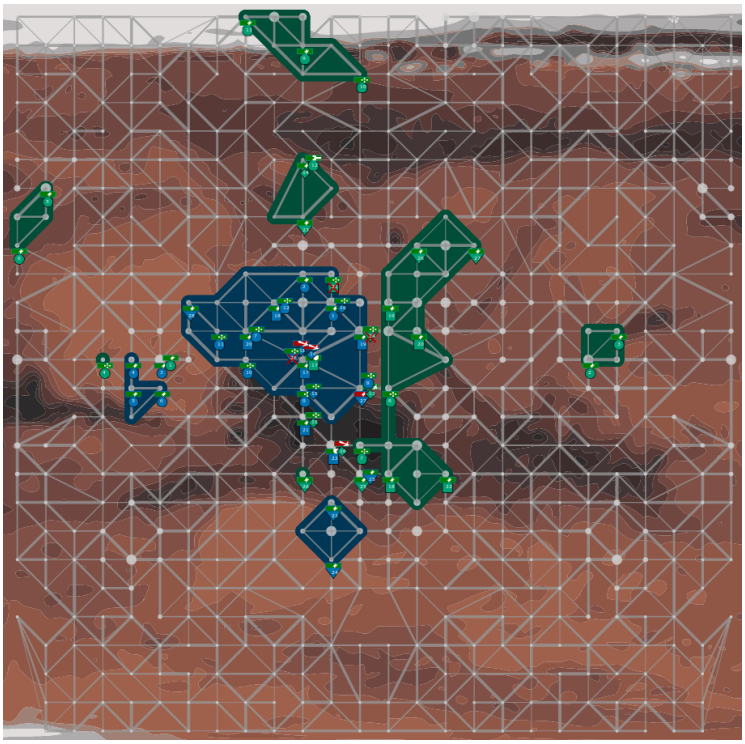

Fig. 25. A screenshot of the agents on Mars scenario.

A.2 The Agents

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, plus a visibility range and a strength level. On top of that, each team has a budget for equipping the vehicles during the simulation.

Of course, all the actions that cost energy will fail if the vehicle under consideration does not have enough energy. When the health level drops to 0 (due to opponent attacks), the vehicle becomes *disabled* until repaired: it can then only perform a subset of the actions, and it does not count for node domination nor for zones calculation.

Actions These actions are defined by the scenario:

- `skip` is the noop-action, which does not change the state of the environment,
- `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 that stands within the visibility range from the attacker, 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 a given vertex within the visibility range,[14] and decreases the vehicle's energy,
- `survey` yields the exact weights of visible edges while decreasing the energy,
- `inspect` costs energy and yields the internals of all opponents standing on the same node, or a given opponent within the visibility range,
- `buy` equips the vehicle with new components, which increase its performance, and cost money, and
- `repair` repairs a given teammate in the visibility range, which, again, costs energy.

The actions that can act at a distance (`probe`, `inspect`, `attack` and `repair`) are regarded as *ranged actions*. When the target of such actions is not the same node where the agent stands nor is an agent standing on the same node, the action can fail randomly following a probability factor, that is calculated based on the visibility range and the distance to the target.

Roles We have defined five different roles. 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.

Each team consists of 28 agents: 6 Explorers, 6 Repairers, 6 Sentinels, 6 Inspectors and only 4 Saboteurs.

A.3 The Scoring

A *step-score* is calculated for each team in every step, and the final score of a simulation is the sum of all step-scores. The step-score is the sum of all area values plus the achievement points that the team retains at the given step.

[14] It is required to probe a node in order to get its full value summed to score when the node belongs to a zone. Otherwise, it only sums as 1 point.

Achievements Achievements are tasks that, when 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. The numbers needed to reach an achievement of a certain type increase exponentially, making them harder to get as the game advances.

For every achievement, a team gets 2 achievement points. These can act as money, that the team may opt to spend in improvements for the agents at any time of the simulation. If not spent, these points contribute to the step-score.

A.4 The Execution Cycle

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. calculate zones and step-score
7. prepare percepts,
8. deliver the percepts.