Multi-agent programming contest 2016

Tobias Ahlbrecht*, Jürgen Dix and Niklas Fiekas

Department of Informatics,
Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
Email: tobias.ahlbrecht@tu-clausthal.de
Email: dix@tu-clausthal.de
Email: niklas.fiekas@tu-clausthal.de
*Corresponding author

Abstract: We present the 11th edition of the multi-agent programming contest, 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 their design and implementation. The long-term aim is to evaluate the specific features of agent programming languages and to compare them to more traditional languages.

Keywords: multi-agent systems; programming contest; multi-agent simulation; MAS; programming competition; MAS benchmark; MAS comparison; agent-oriented software engineering.

Reference to this paper should be made as follows: Ahlbrecht, T., Dix, J. and Fiekas, N. (2018) 'Multi-agent programming contest 2016', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.58–85.

Biographical notes: Tobias Ahlbrecht is a PhD student in the Computational Intelligence Group at the Department of Informatics of Clausthal University of Technology. He earned his Master's degree in Computer Science from the same department in 2016. He is a co-organiser of the multi-agent programming contest since 2013 and his current research interests include multi-agent systems modelling and simulation.

Jürgen Dix is a Professor for Artificial Intelligence and Dean of the Faculty at Clausthal University of Technology. In the past 30 years he worked on basic research in knowledge representation and reasoning, deductive databases and multi-agent systems. He co-authored or co-edited more than 20 books, over 70 journal publications and organised/chaired more than 40 conferences and workshops. He is on the editorial boards of seven journals and several steering committees.

Niklas Fiekas is a graduate student in Clausthal, Germany. He earned his Bachelor's degree in Computer Science from Clausthal University of Technology in 2016. Previously, he was working on scalable multi-agent simulation in the Decentralised Simulation project within the Simulation Science Center Clausthal/Göttingen.

1 Introduction

In this preface to the special issue, we

1 briefly introduce our *contest*[1] and its development in the last ten years

2 elaborate on the brand new 2016 scenario (which will also be used, slightly modified, in the 2017 competition)

3 introduce the five teams that took part in the tournament

4 analyse and interpret interesting matches

5 evaluate the performance and strategies of the teams.

The *multi-agent programming contest* (MAPC) is an annual international event that started in 2005, initiated by Jürgen Dix and Mehdi Dastani (with a lot of help from Peter Novak). In 2016, the competition was organised and held for the 11th time. It is an attempt to stimulate research in the field of programming multi-agent systems by

1 identifying key problems

2 developing suitable benchmarks

3 comparing agent programming languages and platforms

4 compiling test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools.

Furthermore, it supports educational efforts in the design and implementation of MAS's by providing concrete problems to solve.

The partaking teams develop one team of agents each, which connects remotely to the contest server holding the environment. The server sends percepts to the agents and awaits their actions, which are executed resulting in changes to the environment.

Detailed information about the strategies of the teams can be found in the subsequent papers in this volume.

1.1 Rationale behind the contest

Originally, our *contest* had been designed for problem solving techniques based on approaches using computational logics. This, however, has never been a requirement to enter the competition. In the last eight years, we were focussing more on general agent-based approaches.

But, again, we do not require participants to use any multi-agent platforms at all: even centralised solutions are allowed. This allows us to compare solutions with and without multi-agent characteristics. This years runner-up, team Flisvos-2016, developed a strategy implemented in Python but without using any specific agent-related concepts, only some HTN-based planning and concepts from logic programming.

Although each single instance of the *contest* has been won by a team using a MAS, we have had almost always some approaches not based on agent technology at all.

When we started the contest, there were not too many professionally developed multi-agent programming languages around. In the first few years, the *contest* was mainly

used for debugging purposes of the participating teams (to improve their own programming language or platform).

We were, however, always interested in developing scenarios to show that the *agent paradigm really pays off*: the particular features in agent programming might be used to *solve* a non-trivial problem *more elegant*, *with less effort* or *more efficient* than with traditional software engineering languages (the questions we require the teams to answer are intended to give us insights into the overall effort spent).

1.2 Related work

For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to Ahlbrecht et al. (2014), Behrens et al. (2010a, 2010b, 2009, 2012b), Dastani et al. (2007, 2008), Köster et al. (2013) and Ahlbrecht et al. (2013a, 2013b). A quick non-technical overview appeared in Behrens et al. (2012a). Similar contests, competitions and challenges have taken place in the past few years.

Google's AI challenge[2], lying dormant since 2011 now, put its focus on intelligent solutions to rather simple games. They also did not impose any restrictions on the participants' implementations, as long as they could be run on their servers. Compared to the challenge, we employ more complicated scenarios in our *contest* so that more potential of agent platforms may be leveraged.

The *AI-MAS (Winter) Olympics*[3] last took place in 2013, featuring a game called 'crafting quest 3'. This competition aims mostly to bring together students and researchers in the AI field. At most three students each may collaborate to create a solution based on a custom SDK. According to their tutorial, participants have to submit JAR files implying that solutions must be in Java.

Other competitions, like the *student StarCraft AI tournament*[4] or the *Mario AI championship* (Karakovskiy and Togelius, 2012) focus on creating solutions for existing games, which are intended for humans to play with the objective of benchmarking reinforcement learning and general (video)game-related AI techniques. Also, these programs have to be able to cope with 'real time', while we allow plenty of time (ca. 4 seconds per step) for reasoning (and network communication).

Various *planning competitions*[5], e.g., the *RoboCup logistics league*[6], as their name suggests, centre around a single aspect of agent systems.

The *general game playing*[7] competitions differ from the others as the games are not known to the participants beforehand. The programs receive game and objective descriptions just before playing and need to come up with a solution on the fly. Most of the games are for 1 or 2 players only, which makes multi-agent approaches rather unsuitable.

Finally, the *trading agent competition*[8] targets the trading agent problem, e.g., managing supply chains. Again, it is more appropriate to use single agent solutions.

In summary, other competitions

- impose restrictions on the software used by the participants
- search solutions for a particular problem domain
- feature simple or single-player games which are more likely to be solved by a 'centralised' solution.

1.3 History: the contest from 2005 to 2014

Through the history of the *contest*, changes to the scenarios were introduced with every new edition including *four* major redesigns.

From 2005 to 2007, a *gold miners* scenario was used, where agents moved on a grid to collect randomly placed gold pieces which had to be taken to the centre. Here, we introduced the MASSim software: a platform for executing the *contest* tournaments.

From 2008 to 2010 we developed the *cows and cowboys* scenario, which was designed to enforce cooperative behaviour among agents (Behrens et al., 2009). The previous gold miners scenario proved to be too simple and could be solved without much agent coordination, i.e., each agent of a team could work completely on its own. The topology of the new scenario's environment was again 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. For example, agents had to coordinate their positions in order to force cows in the desired direction. The scenario itself evolved, too. The first instance was won (out of competition) by a really simple team of agents where each agent worked on its own, only pushing a single cow at a time. This showed again that participants tend to ignore cooperation aspects if the scenario does not require them. Thus, cows were improved with a flocking algorithm to make it harder to chase single cows and allow groups of agents to move an entire herd. Also, fences were added, which required one agent to stand on a switch on one side and another agent to move through and keep the fence open with a switch on the other side, again discouraging lone wolf behaviour.

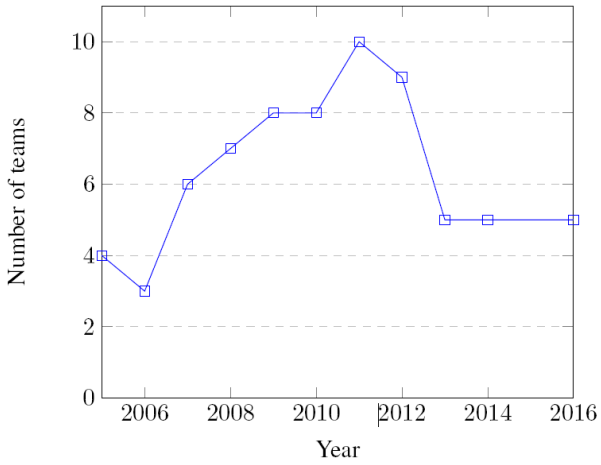
In 2011, the *agents on Mars* scenario (Behrens et al., 2012b) was newly introduced. In short, the environment topology was generalised to a weighted graph. Agents were expected to cooperatively establish a graph covering while standing their ground in an adversarial setting and reaching certain achievements. In the previous scenario, agents had to work together in order to achieve their goals. However, interaction with the opponent team was still rather indirect. Now, agents could only earn points in an area if the other team was hindered from doing so. For example, agents now possessed the ability to disable other agents. Also, complexity was increased once again by adding multiple agent roles with different characteristics and available actions. For now, higher complexity is our only measure to discourage centralised solutions. The basics of the *agents on Mars* scenario remained until the 2014 edition[9], although several modifications were introduced to keep the *contest* challenging.

In 2015, work on the current scenario, which is presented later in this paper, began, as the *Mars scenario* had been scaled up a few times and some good strategies began to emerge. The new scenario was first used in the contest's 2016 edition. Again, one of the main goals was to discourage centralised solutions and instead favour those where intelligent behaviour was emergent, like in a real MAS.

1.4 History 2: participants and their origins

In its 11th editions, the *contest* has seen 70 team participations (some teams partaking in multiple editions, but mostly with changing members) from all over the world. Figure 1 shows the development of team numbers over the years.

Figure 1 Participants of the MAPC through the years (see online version for colours)



In its humble beginnings, the *contest* started off with four teams in 2005 and three the year after. From then on, the number of teams monotonically increased until 2011, the best year yet with ten participating teams. The number has gone down to five teams since and has remained there for three editions. The majority of teams originated from academia: only two contestants competed without affiliation.

As of 2016, we are counting 19 different countries. The most frequently participating country, Germany, denotes a total of 22 attempts to win. This is mostly due to the tireless efforts from the Technical University of Berlin, who started in 2007 and have not missed a contest since, sometimes even contributing two teams in the same year. This has already fetched them the first place in 4 consecutive editions, starting right in 2007.

Closely following is Brazil, with 12 attempts and leading the list of winning countries with five first placements from three universities. The team from Federal University of Santa Catarina participated in every instance of the Mars scenario and won all three contests. Brazil is immediately followed by eight participations of Danish teams, all from Technical University of Denmark. They started participating in 2009, also not having missed a single contest since, while having contributed two teams in 2014. Starting in 2008, the *contest* also saw a team from University College Dublin for five consecutive years. In the same order of magnitude, seven teams from Iran participated already, from which four teams just in 2011, three of which originated from Arak University, Iran.

Other sporadically participating countries, sorted by their first appearance, include the UK, Japan, Chile, Spain, Switzerland, the Netherlands, Australia, Poland, France, Turkey, Argentina, China, the US and Greece.

2 MAPC 2016: the new scenario

The new scenario consists of two teams of 16 agents each moving through the streets of a realistic city. The goal for each team is to earn as much money as possible, which is

rewarded for completing certain jobs. An agent is characterised by its battery (i.e., how long it can move without recharging), its capacity (i.e., how much volume it can carry) and its speed. The scenario features four distinct roles: drones, motorcycles, cars and trucks, sorted by increasing capacity and energy, and decreasing speed.

Table 1 Agent roles and their properties

Role	*Battery*	*Capacity*	*Speed*
Drone	250	100	5 (not limited to roads)
Motorcycle	350	300	4
Car	500	550	3
Truck	1,000	3,000	2

The simulation is divided into discrete steps and each agent can execute at most one action per step. In the following we give a detailed description of the scenario and actions. See Table 4 for an overview of all available actions.

The city map is taken from *OpenStreetMap*[10] data and routing is provided by the *contest* server. Each agent can move a fixed distance each turn (unless its destination is closer than this distance). For this, agents can use the `goto` action specifying a location on the map. The agents are positioned randomly on the map, as are a number of *facilities*, which allow the agents to perform certain actions. In *shops*, items can be bought at prices specific to the particular shop. Each shop only offers limited quantities of a subset of all items. However, items are restocked after a number of steps. Agents can use the `buy` action to obtain number items of a single type per step whenever they are in the same location as a shop where this item is available. To promote exploration behaviour, an agent can only perceive the number and cost of items in a shop if it is currently standing in said shop.

Charging stations have to be frequently visited by agents in order to recharge their battery. They always have to be taken into account when planning routes.

In workshops, agents can assemble items out of other required items. Most items also need specific tools, which can only be used by a particular role. Agents may also cooperate here. In that case, the resources of those agents are combined and the initiating agent receives the finished item (if all prerequisites are satisfied). Thus, the agents need to coordinate who obtains which items and how they work together for assembling complex items. Agents have the actions `assemble` and `assist_assemble` at their disposal, whereby the first has to be called by the initiator, who also wants to receive the assembled item. The latter action has to be used specifying the agent that is to be assisted.

Items can be destroyed at *dumps* (to free capacity), while *storages* allow to store items up to a specific volume for a certain price and also are the target for completing jobs. Intuitively, storages require a `store` action and two types of `retrieve` actions. One for getting items stored intentionally and one for retrieving items which belong to the team but have been added to the storage for other reasons (and which do not count towards the storage's capacity).

Jobs comprise the acquisition, assembling, and transportation of goods. These jobs can be created by either the system (environment) or one of the agent teams. There are two types of jobs: normal ones and auctions. A team can accept an auction job by bidding

on it. The bid will be the reward, i.e., the team is willing to procure the items for that amount of money. Thus, if both teams bid, the lowest bid wins. If a job is not completed in time, the corresponding team is fined to discourage auction 'hoarding'.

Regular jobs have their rewards defined upfront, which is given to the first team to complete that job, while the other team goes away empty-handed. The teams have to decide which jobs to complete and how to do them, i.e., where to get the resources and how to navigate the map considering targets like shops, charging stations, workshops, or storage facilities. While auction jobs have to be more thoroughly assessed in order to determine a minimum threshold at which the team would still earn money, they provide some safety since the team that did not win the auction is effectively barred from completing it.

Actions associated with jobs are `bid_for_job`, `deliver_job` and also `post_job`, which allows agents to add their own jobs to the simulation. The source of a job is not disclosed to the agents, so that posted jobs are potentially indistinguishable from those created by the server.

Item types are also randomly generated and thus differ in each simulation. They can be either bought or crafted. Agents can give them to a teammate, store them temporarily in a storage, deliver them as part of a job or ditch them in a dump.

In case an agent runs low on energy with no charging station in sight, it can call the breakdown service, which takes a long time to arrive and a considerable fee to charge the agent's battery.

Tournament points are distributed according to the amount of money a team owns at the end of the simulation. To get the most points, a team has to beat the other, as well as surpass the seed capital given to the team at the beginning of the simulation, i.e., make an overall profit.

The scenario calls for increased coordination and planning among agents of the same team. They have to decide which jobs to go for and how to handle them. Items for the jobs can be either bought or partly assembled to save money (as assembly is intended to be much cheaper than buying). Also, mixed strategies are possible, where only a convenient subset of items for a job is assembled. Routing from point to point is provided by the server, while planning the sequence of points is of course still up to the agents.

Interaction with the opponent team is for now limited to job and resource (items in shops) contention, the bidding for auction jobs, and the posting of jobs for the other team, which allows to outsource parts of a 'real' job.

3 The tournament

Following the tradition, 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. The qualification rounds showed extremely positive results: each and every team encountered not even a single timeout.

The tournament took place on the 12 and 13 September 2016. Each day the teams played against two other teams so that in the end all teams played against each other. We started the tournament each day at 1 pm and finished around 6 pm.

3.1 Simulation definition

A match between two teams consisted of three simulations comprising 1,000 steps each. These three simulations differed in the map that was used and the form of jobs that were generated. However, each match featured the same three sets of simulation parameters.

The first of each set of simulations was played on the street graph of (a part of) London. Jobs had to be completed in 250 to 350 steps. The jobs' rewards were comparatively the lowest of all three simulations. For the second simulation, played on a map of the German city Hannover, the job completion time bounds were decreased by 25 steps each and the potential job rewards increased. The same adjustments were made for the third simulation, played on the San Francisco street graph, again decreasing completion times and increasing job rewards.

To summarise, in one match it became (on average) more difficult but at the same time more rewarding to complete a job.

Note that, while the simulation sets featured the same parameters, of course the concrete simulation instances that were played by the teams were (with a very high probability) never the same due to the random generation mechanism. The goal was, as in previous editions of the *contest*, to generate a set of similar simulation instances, so that one could compare the simulations afterwards without risking to give the agents the possibility to learn the exact simulation from match to match (e.g., the prices of items or what job will be generated in which step).

3.2 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 2).

Table 2 Participants of the 2016 edition

Team	*Affiliation*	*Platform/language*
BathTUB	Technical University of Berlin	JIAC V
Flisvos 2016	None	Python
lampe	Clausthal University of Technology	C++
PUCRS	Pontifical Catholic University of Rio Grande do Sul	Jason, CArtAgO, Moise
Python-DTU	Technical University of Denmark	Python

The teams were awarded three points for winning a simulation (minus 1 point if they did not make a profit) and 1 point in case of a draw. The results of this year's Contest are summarised in Table 3.

Table 3 Results

Pos.	*Team*	*Score*	*Difference*	*Points*
1	PUCRS	6,503,165 : 994,109	5,509,056	33
2	Flisvos 2016	7,197,893 : 941,069	6,256,824	30
3	lampe	1,320,153 : 1,601,549	–281,396	12
4	Python-DTU	532,714 : 7,764,645	–7,231,931	6
5	BathTUB	–625,240 : 3,627,313	–4,252,553	5

PUCRS secured an almost flawless victory, only losing a single simulation against runner-up *Flisvos 2016*, thus scoring 33 out of 36 possible points. *Flisvos 2016* in turn only lost two of the simulations against *PUCRS*, resulting in 30 points total. Nevertheless, *Flisvos 2016* had a better score difference, which may be attributed to variations in the simulation instances (see Sub-section 3.1 for details).

Following with a rather significant margin, team *lampe* won the third place with 12 points, still one out of each three possible points. The points were gained from two victories each against *BathTUB* and *Python-DTU*.

Making for a close battle for the fourth place, *Python-DTU* was able to take it with a one point lead over *BathTUB*. Having the edge in their match against each other, *Python-DTU* won 2 out of the 3 simulations. However, no team was able to make a profit, resulting in 4 and 2 points respectively. Both teams were able to win one simulation against team *lampe*, resulting in 2 additional points for *Python-DTU*, and even 3 points for *BathTUB*, since they made a profit in that simulation.

3.3 The teams and their agents

In this section we collect information about the participants and their agent team strategies. For more details, we refer to the team description articles that follow this preface.

- *BathTUB:* the team *BathTUB* (Hessler, 2017) from Technical University Berlin, Germany, is a regular contender of the *MAPC*. Their agents are once again developed with the JIAC V platform (which won the contest several times in previous years). This time, six students and their supervisor have spent around 1,000 person-hours developing their agent team. The approach is partly centralised, each agent being coordinated by a central instance. It was planned to complete jobs as fast as possible. To achieve that, a similar approach to *PUCRS* was chosen: any agent may initiate the planning of a job and receives proposals from all the agents, i.e., which items they can procure at which cost. The initiating agent then decides on the optimal course of action and informs the other agents.

  To prevent idling, a measure for proactiveness was implemented: the agents explore the map, plan a new job or keep watch on the opponent agents.

- *Flisvos 2016:* the team *Flisvos 2016* (Sarmas, 2017), consisting only of a single person, participated for the first time in the Contest and promptly came second, losing only two simulations against this year's winners. The agents were implemented in Python, having invested roughly 250 hours. The centralised approach relies on no special agent-related concepts, instead employing a global planning technique together with known optimisation heuristics. The agents only communicate by updating a shared data structure in order to exchange percept information.

- *lampe:* the two students of team *lampe* (Czerner and Pieper, 2017) from Clausthal University of Technology developed their agents in C++, spending about 150 person-hours. They also rely on a centralised approach together with a heuristic for choosing profitable jobs.

- *PUCRS:* the team *PUCRS* (Cardoso et al., 2017) from Pontifical Catholic University of Rio Grande do Sul won this year's contest, only losing a single simulation against the runner-up. Approximately 230 person-hours were invested by the 11 members into developing the agent team in Jason, CArtAgO and Moise (JaCaMo). The *PUCRS* agents start each simulation by exploring nearby shops to get information about the prices of items. The exploration is coordinated with a token ring communication, where each agent places its route to each facility if it is the shortest one yet.

  After that, they start evaluating the incoming jobs and estimate their costs (for recharging the agents and buying or assembling the items). If a job's reward surpasses the cost, the agents try to complete it.

  This is achieved by splitting the job into tasks which are distributed among the team members using the contract net protocol.

  In addition, the agents try to deceive the opponent agents by posting their own jobs, rather to distract them than to outsource a task.

  Nearing the end of the simulation, the agents adjust their strategies so they do not take any job they cannot complete anymore.

- *Python-DTU:* the team *Python-DTU* (Villadsen et al., 2017) from the Technical University of Denmark is another regular contender of the *MAPC*. After having tried GOAL in the 2013 (and 2014) edition, as the name suggests, the team changed back to using Python for this *contest*. The four members spent around 400 person-hours developing their agents, not using any existing multi-agent technique or framework but plain Python instead. As most of the other teams, they chose a centralised approach. Simulations have shown the agents to always spend a certain amount of money (a few hundred up to 15,000) and then stopping any noticeable behaviour. In one particular match, the team tried a new strategy and posted up to 16 new (non-sensical) jobs in each step[11].

4 Performance of the teams

In this section, we will look at how the teams performed in the *contest* regarding scores, completed jobs, stability and how they used certain aspects of the scenario.

4.1 Score and completed jobs

The score or the team's money is mostly depending on how many jobs the team completed and how economical the agents did it, as completing jobs is the only source of income in the scenario. Money is decreased through buying items for the jobs or recharging the agents when their energy gets low.

The scores and the number of completed jobs allow to assess a team's overall performance and compare it to other teams on the simulation level[12].

- *PUCRS:* the *contest* winner completed 343 jobs in total, earning a money value of 18,221,172, or 53,122 on average per job.

Comparing by simulation, the biggest difference in completed jobs was to *Python-DTU*, the smallest to *Flisvos 2016*. Somewhere in between, we see similar numbers for the simulations against *BathTUB* and *lampe*. *PUCRS* completed more jobs than the opponent team in all but one simulation against *Flisvos 2016* (also being the one simulation they lost). However, in two simulations, *Flisvos 2016* earned more money, which means *PUCRS* was a little more efficient in choosing and completing jobs, as they won one of these simulations anyways. Namely, in their second simulation, *PUCRS* completed 13 jobs more, earning on average 11,404 (vs. 21,910 for *Flisvos 2016*), showing that *PUCRS* focused on smaller jobs.

- *Flisvos 2016:* the runner-up completed 402 jobs, earning a total sum of 23,247,534. Of course, the smallest difference in completed jobs shows in the simulations against *PUCRS*. However, the numbers against all other teams look similar. *Flisvos 2016* was able to complete more jobs than any opponent team in all but two simulations. Correspondingly, there was only one simulation where *Flisvos 2016* earned less money than the opponent.

  Against *lampe* and *BathTUB*, the team completed smaller jobs in comparison in two out of three simulations each. In contrast, against *PUCRS* the average reward of jobs completed by *Flisvos 2016* was higher in all three simulations.

  This leaves us with two possible conclusions. Either, *Flisvos 2016* was second best in decomposing the agents into smaller teams, thus completing more jobs in parallel, or the smaller size of jobs can be attributed to efficiency, being able to finish jobs with a smaller reward and still making appropriate profit of it (but most likely a combination of both).

- *lampe:* the team completed rather few jobs compared to the previous two. Also, it finished less jobs than *BathTUB* in all of their three simulations, winning two of them anyway. In their first simulation, they earned less money on average per job (and in total) than *BathTUB*, resulting in the lost simulation. Interestingly, in their second simulation, they still earned less money in total, but more on average per job. Having spent less money than *BathTUB*, this win went to *lampe*.

  In two out of 12 simulations, one against *PUCRS* and one against *Python-DTU*, *lampe* did not complete any job. In the first case, they finished with their starting capital, indicating some one-time bug.

  Looking at the averages again, *lampe* aimed for comparatively high job rewards, pointing to a rather conservative heuristic. Only in the first simulation against each team the average reward for *lampe* was smaller compared to the opponent's.

- *Python-DTU:* weirdly enough, the fourth-placed team did not complete a single job. They won two simulations against *BathTUB* in which the opponent team finished with a big negative score and one against *lampe* with a monetary lead of roughly 2,000.

- *BathTUB:* the team completed more jobs than *lampe* or *Python-DTU*, in total 54. Unfortunately, the agents could not make a profit in any simulation on the first day of the *contest*. On the second day, they showed a varying performance. For example, against *Flisvos 2016*, *BathTUB* was able to make a good profit (134,763) followed by a rather big loss (–558,096) and only a small loss (–12,742). In all of these

simulations, the team completed a similar amount of four to six jobs, pointing to rather unpredictable investments. Their average reward against *Flisvos 2016* was higher in two out of three cases, however, they completed far less jobs.

Against *lampe*, they won the first simulation, in which they completed jobs with a higher average reward. In the following two simulations, they still completed more jobs but with a lower average reward than their opponent and lost both of those, again with varying final scores: some profit in the first simulation, some loss in the second and a good (but not big enough) profit in the third.

Having a final look at the completed jobs overall, none of them has been posted by any competing team, i.e., unfortunately the teams did not try or succeed in fooling their opponents into doing some work for them.

4.2 Agents' behaviour

In this section, we will have a look at what actions the agents used to which extent and how that possibly affected the outcome of single simulations and the *contest* as a whole. As each team consisted of 16 agents, there were 16,000 actions in each simulation per team. A big part of the actions of all teams is naturally made up of the actions `skip`, `continue` and `abort`, as those are necessary to maintain the also frequently used actions `goto` and `charge`.

Table 4 Total action counts

	Flisvos 2016	*lampe*	*BathTUB*	*PUCRS*	*Python-DTU*
goto	5,224	3,721	26,860	7,134	4,892
noAction	0	35,824	7,823	1,855	4,010
skip	169,034	192	153,588	137,936	142,011
abort	0	108,491	32	0	0
continue	0	19,245	0	35,017	3,932
charge	2,042	2,951	1,885	2,326	3,910
retrieve	0	0	0	0	0
retrieve_delivered	0	406	0	0	0
store	0	0	0	0	0
dump	0	0	0	36	0
deliver_job	1,219	274	406	1,764	0
buy	1,695	342	1,166	2,044	164
assemble	0	10,010	0	0	17,734
assist_assemble	0	10,269	0	0	12,483
give	0	0	0	0	0
receive	0	0	0	0	0
call_b. (inst.)	12,824 (438)	467 (16)	0	3,338 (114)	0
post_job	154	0	0	742	3,056
bid_for_job	0	0	432	0	0

Also, the actions `store`, `retrieve`, `give` and `retrieve` were not used at all, indicating the direction in which the scenario needs to be improved. Total action counts for all teams can be found in Table 4.

4.2.1 PUCRS

The team used the actions corresponding to job activities according to previous observations. Comparing these job related actions, as depicted in Figures 2 and 3 for the first and second simulation against *Flisvos 2016*, the numbers go in line with *PUCRS* having completed the most jobs in the *contest*. Looking at the first simulation (where *PUCRS* lost), we see that *PUCRS* used more `deliver` but less buy actions than the opponent. However, around 50% of *PUCRS*' `deliver` actions against *Flisvos 2016* failed, many of them due to the job not being active (possibly already completed by *Flisvos 2016*) or because the delivering agent did not carry any items that were still missing for the particular job. Thus we can conclude that *PUCRS* employed less partial deliveries than *Flisvos 2016*, as they needed less (successful) actions to complete more (or comparable amounts of) jobs.

Figure 2 Job actions – *Flisvos 2016* vs. *PUCRS* – simulation 1 of 3 (see online version for colours)

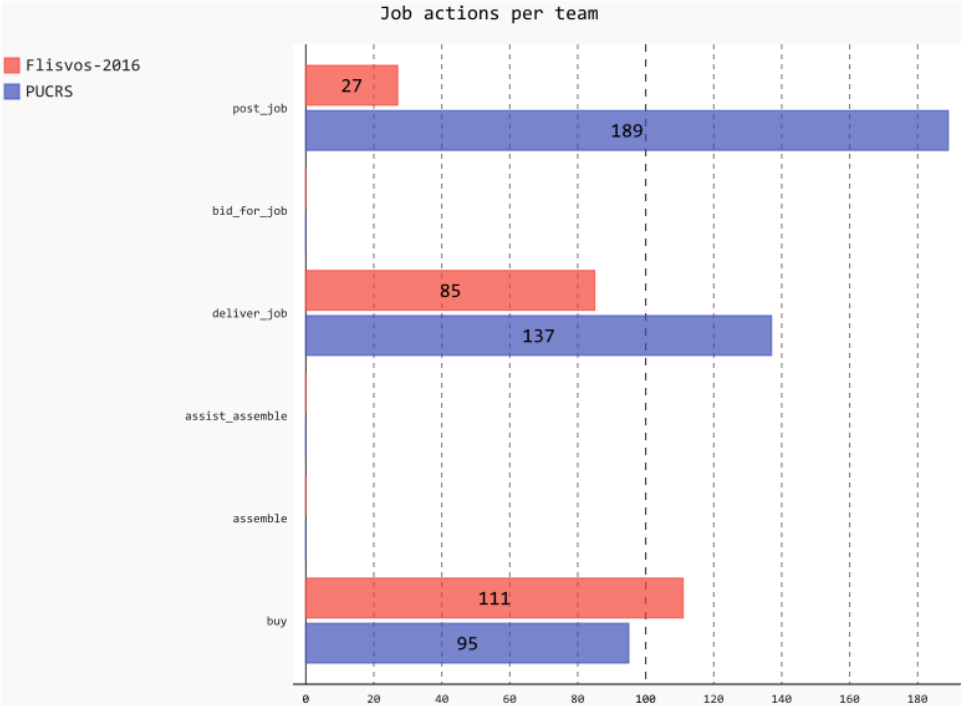
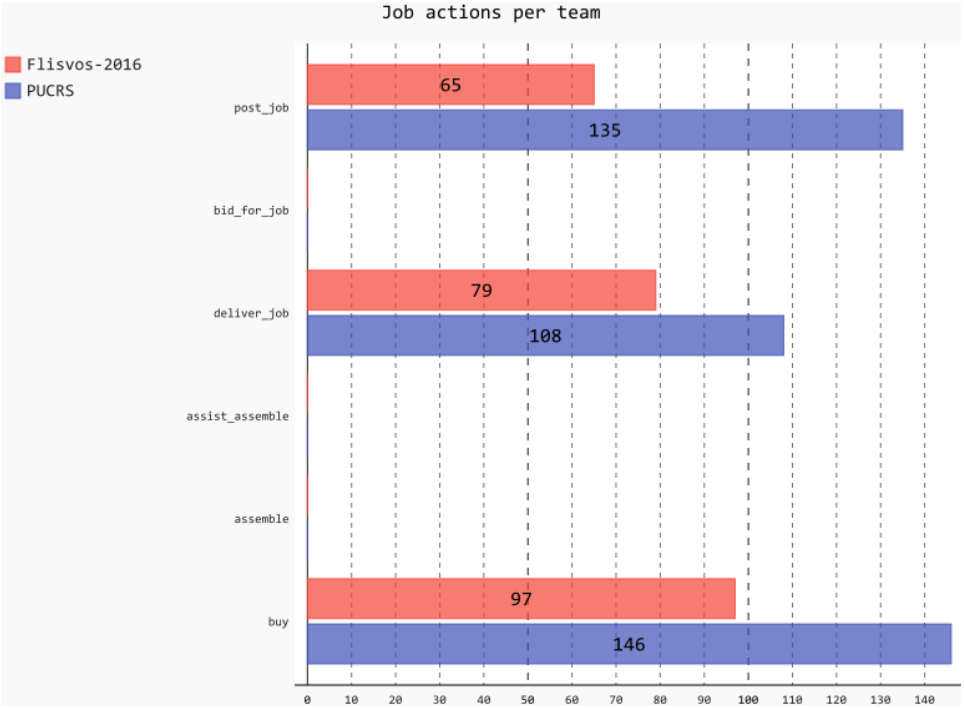


Figure 3 Job actions – *Flisvos 2016* vs. *PUCRS* – simulation 2 of 3 (see online version for colours)



From the same figures, we can see that *PUCRS* did not use any actions for assembling products (making it another action to promote in future editions of the *contest*). Indeed, *PUCRS* did not use the actions `assemble`, `assist_assemble`, `retrieve_delivered` and `bid_for_job` for the entirety of the *contest* (in addition to the actions that were never used by any team).

The `buy` action was used the most by *PUCRS* which also agrees with them having completed the most jobs. They were also the only team to use the `dump` action, however, there were only 36 cases where they saw the need to dispose of some items, 20 of them in the match against *Flisvos 2016*. The `post_job` action was used sparingly in order to distract the opponent teams by adding non-sensical jobs and therefore consuming some of their processing time.

Finally, the team was one of the few to use the `call_breakdown_service` action regularly, counts reaching from 100 to 500 times per simulation. However, as the action has to be called repeatedly (25 times if it does not fail randomly) to have an effect, this amounts to the agents being left without charge 114 times in total, or 9.5 times per simulation, which is less than the number of agents.

4.2.2 Flisvos 2016

The action counts of team *Flisvos 2016* are overall similar to *PUCRS*. The actions `abort` and `continue` were not used, the latter being replaced by more `skip` actions. The team did not make use of the storage actions `retrieve`, `store` and `dump`. Also, `bid_for_job` and `retrieve_delivered` were not used, thus ignoring auction jobs and the possibility to retrieve (unsuccessful) partial deliveries.

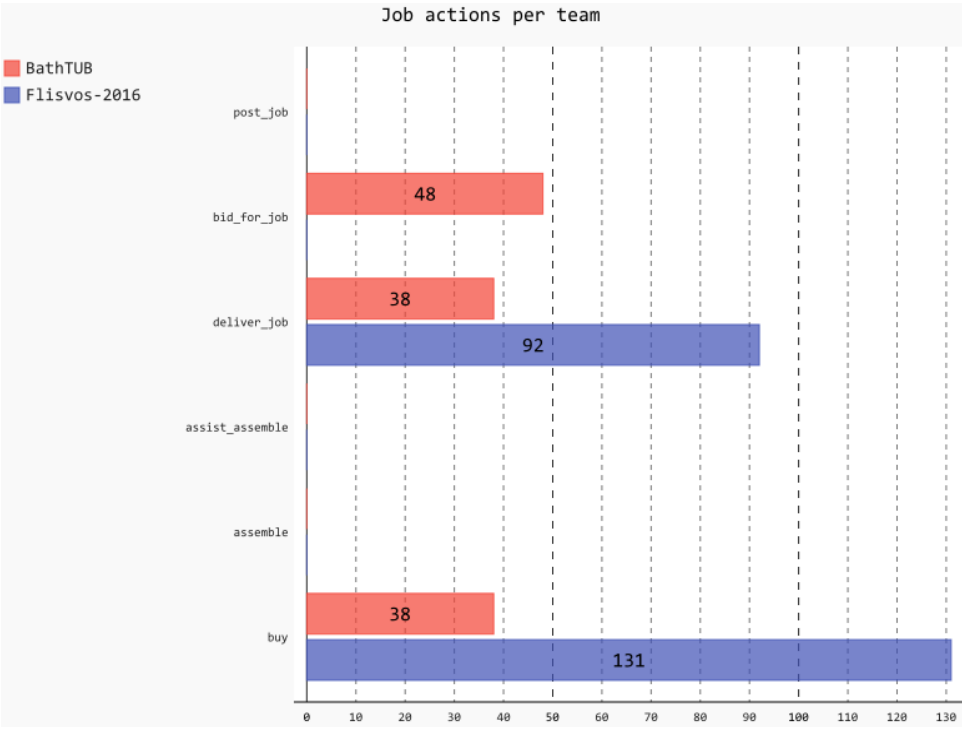
Unfortunately, the explicitly cooperative actions `give`, `receive`, `assemble`, `assist_assemble` were also not used at all by *Flisvos 2016* (again, possibly not being pushed enough by the concrete simulation instances).

Flisvos 2016 was one of three teams to use the `call_breakdown_service` action, roughly three times more often than *PUCRS*, indicating either a tiny routing/planning problem or that the team accepted to use the action in order to complete some otherwise unachievable jobs. In total, uncharged agents were counted 438 times, which is 36.5 times per simulation or 2 to 3 times per agent and simulation.

The `post_job` action was also used, however, more moderately than by *PUCRS*. As no team completed a job posted by an opponent team, the team did not need to use the `retrieve_delivered` action, as noted above.

Comparing the job related action counts to those of *BathTUB*, e.g., for their first simulation as given in Figure 4, again the difference in `buy` and `deliver` actions becomes obvious.

Figure 4 Job actions – *BathTUB* vs. *Flisvos* 2016 – simulation 1 of 3 (see online version for colours)



4.2.3 lampe

The *lampe* team decided to use the `continue` action in combination with `abort` instead of `skip`, thus being the only team to make sure that ongoing actions are explicitly stopped.

Interestingly, the team was the only one attempting to use the `retrieve_delivered` action (however, only making up ca. 0.2% of their actions). Unfortunately, the action only succeeded 20 times out of 406 for them.

The *lampe* agents did not use the `give` and `receive` actions, but they performed the most successful `assemble` (and accordingly `assist_assemble`) actions, which allowed them access to more valuable jobs. The ratio of these actions was almost 1:1 suggesting that most often exactly two agents cooperated.

Of the teams to use the `deliver_job` action, this team used it the least, again highlighting their conservative heuristic which made the agents only work on the most rewarding jobs while forgoing a lot of smaller ones. For example, much of this can be seen in their third match against *BathTUB*, with both teams' actions compared in Figure 5 and the failed actions of team *lampe* in Figure 6.

We see that *lampe* used fewer delivery actions but had some successful assemblies. Comparing this to the money development in that simulation, as charted in Figure 7, one can see again how it paid off (here) to complete jobs with higher rewards on average, especially halfway through the simulation, where *BathTUB* only little more than broke even. Counting the increments, *BathTUB* indeed completed one job more than *lampe* in this simulation.

Figure 5 Job actions – *BathTUB* vs. *lampe* – simulation 3 of 3 (see online version for colours)

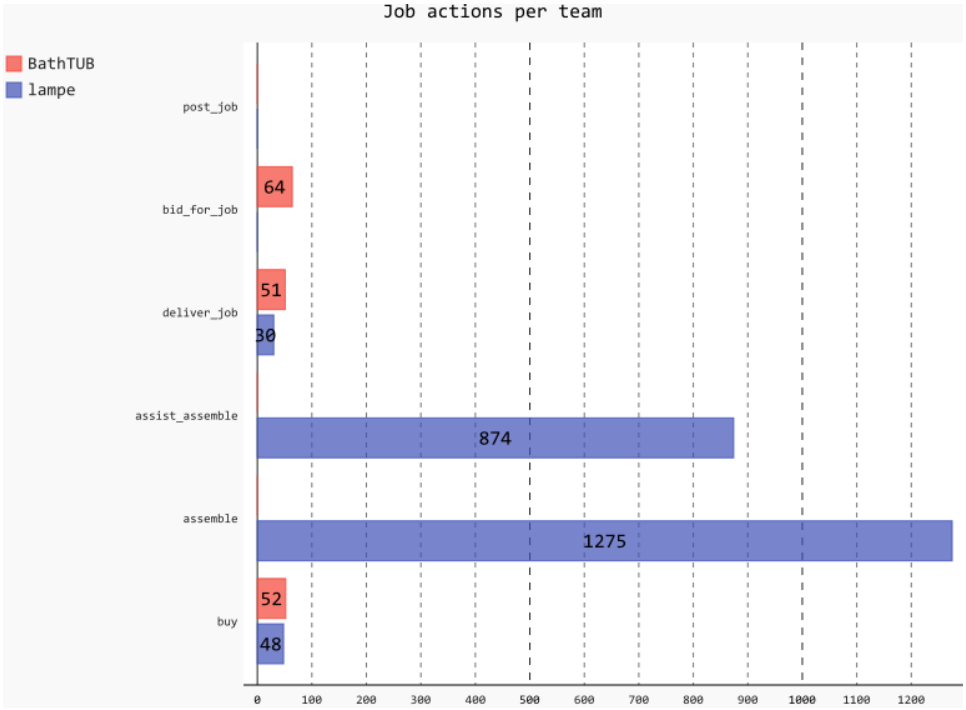


Figure 6 Failed job actions *lampe* – *BathTUB* vs. *lampe* – simulation 3 of 3 (see online version for colours)

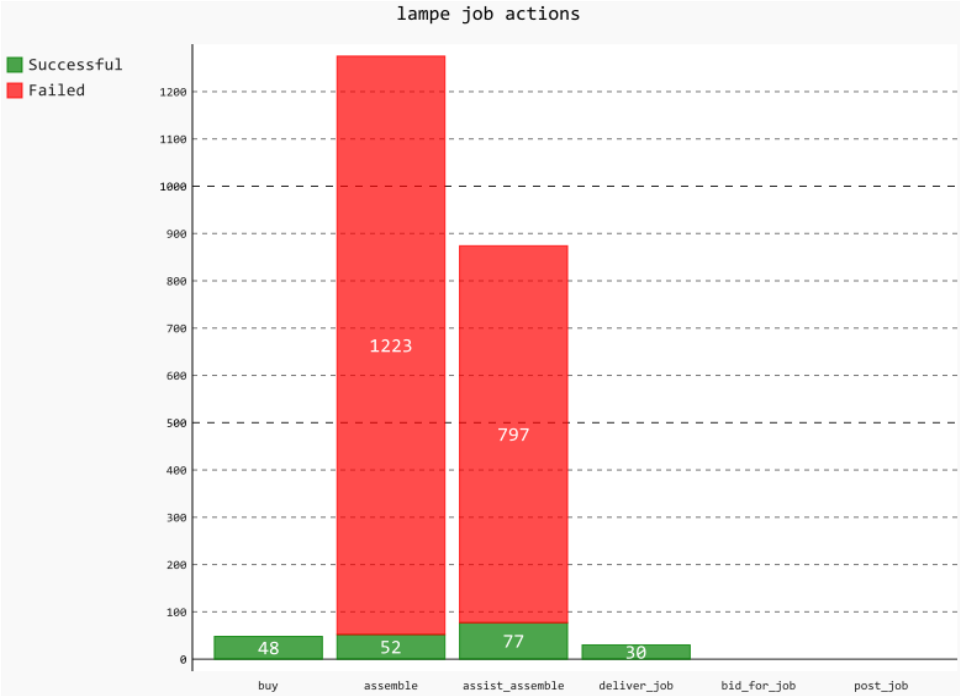
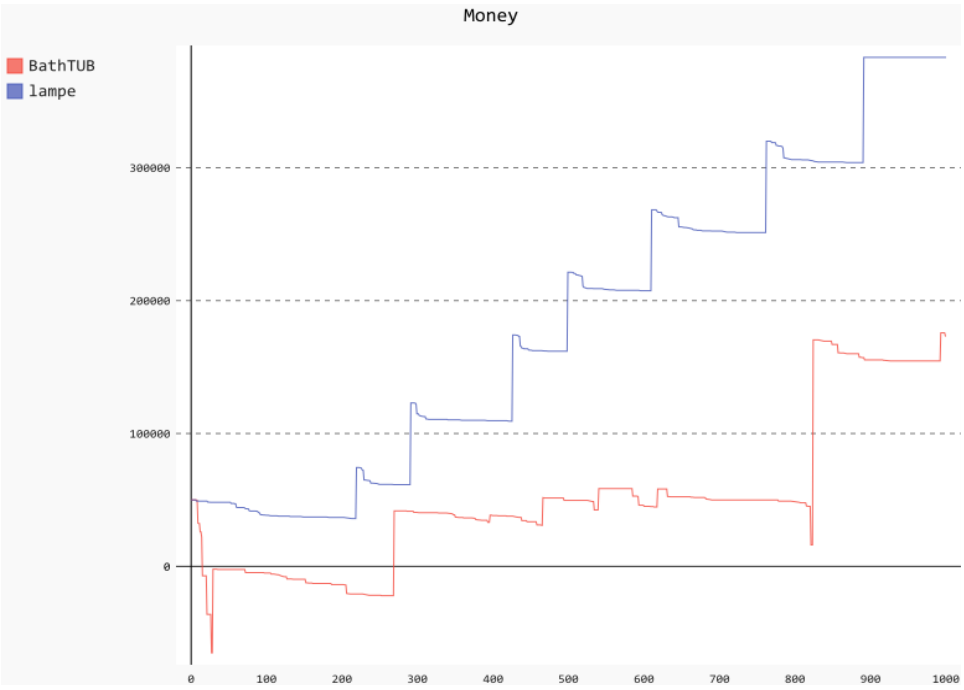


Figure 7 Money – *BathTUB* vs. *lampe* – simulation 3 of 3 (see online version for colours)



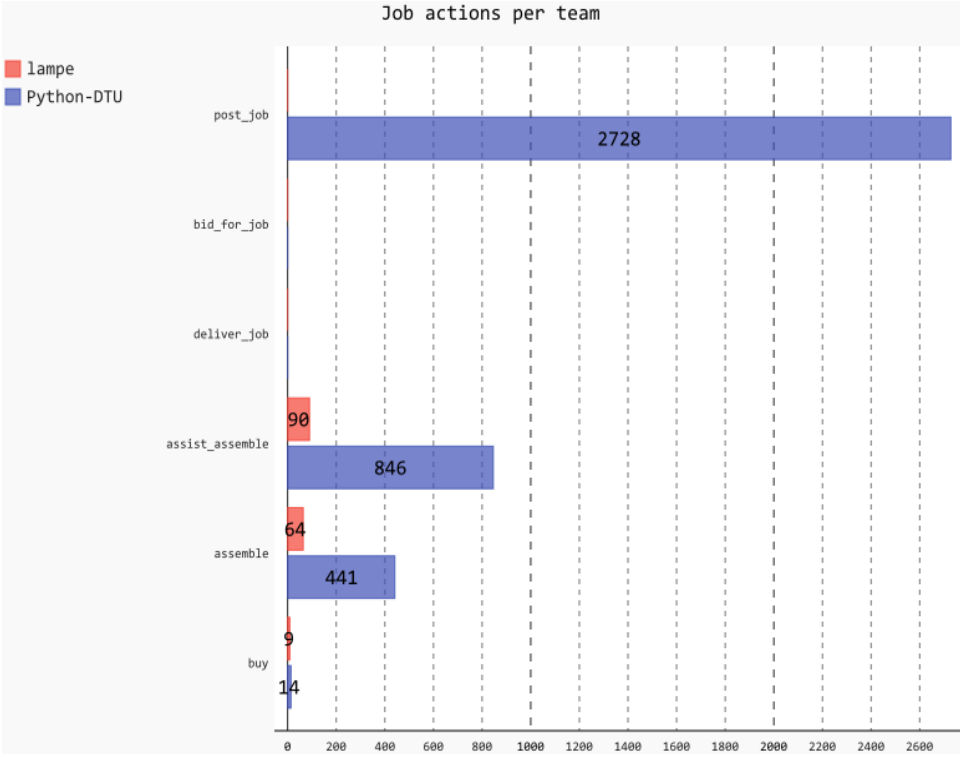
4.2.4 Python-DTU

The team *Python-DTU* used all the common actions to a 'normal' degree, featuring a relatively high amount of `skip` actions.

Regarding storage actions, the team did not use any of them (e.g., `retrieve` or `deliver`). Also, as with any other team, `give` and `receive` were not used. As for the remaining actions, the team did not make use of `bid_for_job` and – surprisingly – `deliver_job`.

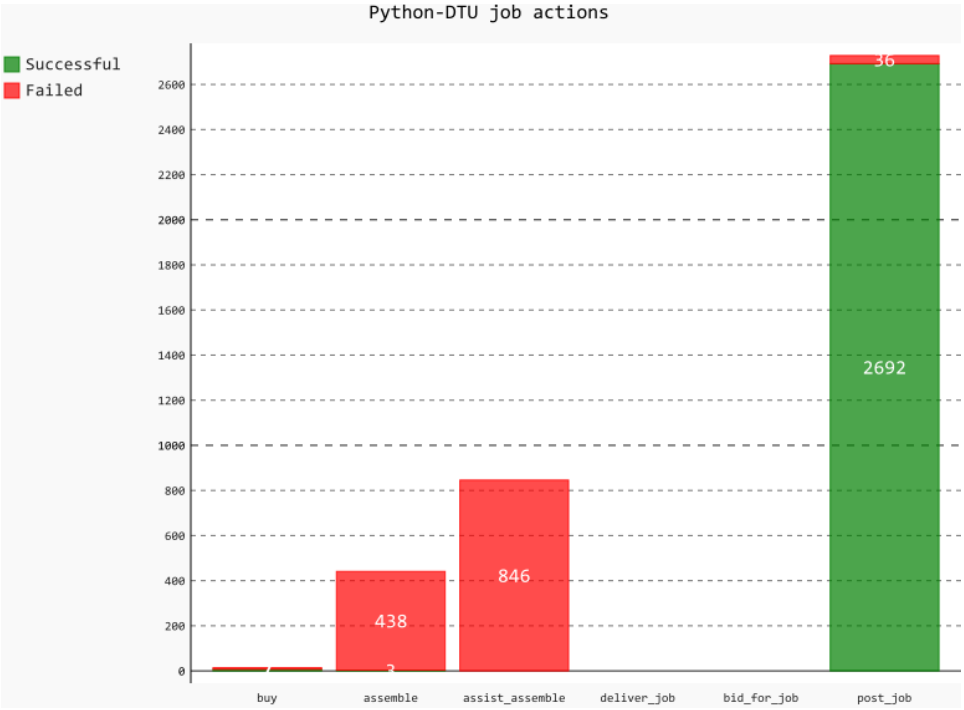
The team used even less `buy` actions than *lampe*: only 164 during the whole *contest*. However, surprising us again, the team used the most (`assist-`)`assemble` actions out of all teams, even surpassing team *lampe* by roughly 77%. Recall that those were the only two teams to use these actions at all.

Figure 8 Job actions – *lampe* vs. *Python-DTU* – simulation 1 of 3 (see online version for colours)



Probably as an attempt to completely occupy the opponent agents, the team used a comparatively high amount of `post_job` actions. Fortunately, only the simulation web monitor seemed to struggle with that much new information and could be quickly repaired. This behaviour was only observed in certain simulations, e.g., the first one against *lampe*, as depicted in Figure 8. In simulations where the *Python-DTU* team used less `post_job` actions, the amount of `assemble` actions was noticeably higher. In the accompanying Figure 9 we can also observe that the majority of `assemble` actions failed.

Figure 9 Failed job actions *Python-DTU-lampe* vs. *Python-DTU* – simulation 1 of 3 (see online version for colours)



4.2.5 *BathTUB*

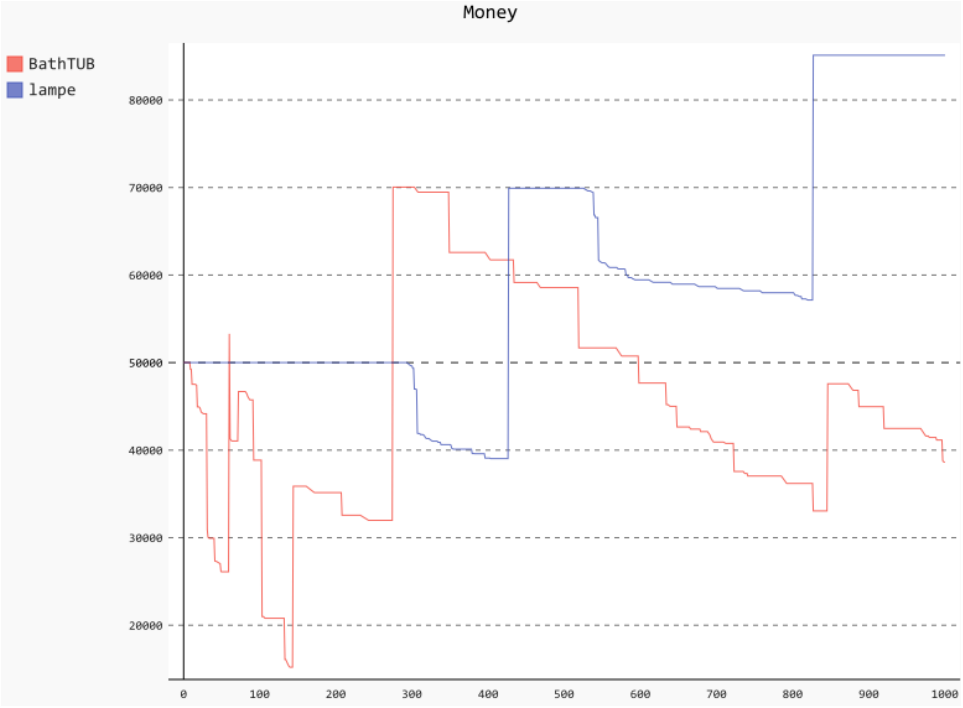
The *BathTUB* team also used no common action to an unusual degree. Like all others, the team did not use the storage actions, nor any of the cooperative assembly and item exchange actions.

Like *Python-DTU*, the team did not need the `call_breakdown_service` action. Also, the *BathTUB* agents did not make use of the `post_job` feature. On the other hand, they were the only ones to use the `bid_for_job` action, which did not fail in most of the cases. The team was also the only one to complete several auction jobs per simulation.

The amount of buy actions for team *BathTUB* lies somewhere below that of *PUCRS* and *Flisvos 2016*, but considerably above that of *lampe* and *Python-DTU*. The effect of these spendings can e.g., be observed in the second simulation of *BathTUB* against *lampe*, as shown in Figure 10.

Shortly after step 400, the teams are almost on par, however, big investments of *BathTUB* are not translated into greater rewards from then on.

Figure 10 Money – *BathTUB* vs. *lampe* – simulation 2 of 3 (see online version for colours)



4.3 Agents' reliability and stability

In the last section, we mostly analysed what the agents *tried* to do. Now, we will have a look at the extent to which the agents submitted correct and *sensible* actions to the system. For example, it does not make sense for an agent to perform a `charge` action if that agent is not currently located within a charging station.

First off, the amount of `noActions`, which are registered if an agent does not submit an action before the timeout, was considerably higher than in the qualification phase. In fact, *Flisvos 2016* was the only team to keep a flawless record of 0 `noActions`. Most timeouts were encountered by team *lampe*, a solid 18.66% of their total actions. On the first day of the *contest*, their agents had a bug preventing them from reconnecting to a match once they lost connection for the first time. Thus, almost half of their 35,824 `noActions` originated from a single simulation, their third one against *PUCRS*.

The second most `noActions` were due to team *BathTUB*, roughly a fifth of those of team *lampe*. *Python-DTU* follows with half of that amount (i.e., ca. 4,000) and *PUCRS* with a little less than half of that again.

The actions that were received by the system could potentially fail due to a number of reasons. Here, we will have a look at the causes of failed actions and how often they were experienced by the teams.

- *failed_location:* this happens whenever an agent tries to perform a location-specific action outside of that location, i.e., charging, assembling or buying. It happened only a single time to *PUCRS* and about 150 times to *Python-DTU*.
- *failed_unknown_item:* this error occurs if an agent specifies a non-existing item as parameter to an action. It happened only a single time to *PUCRS*.
- *failed_unknown_facility:* this is the same as above, but for facilities. It also occurred only once for team *PUCRS*.
- *failed_item_amount:* this occurs whenever an agent tries to use more items for an action than it currently carries. It happened 13 times to *PUCRS* and about 12,000 and 18,000 times for *lampe* and *Python-DTU* respectively, making it the most frequently occurring failure code. We can assume that both of the latter teams had a rather serious planning problem, as these numbers explain the previously seen counts of failed `assemble` actions.
- *failed_tools:* whenever a group of agents tries an `assemble` action without carrying the necessary tools, this failure occurs. Again, this occurred to both *lampe* and *Python-DTU*, about 7,000 and 11,500 times respectively, making it the other big cause of failed `assemble` actions. As 'failed_tools' takes precedence over 'failed_item_amount' and both cases could be satisfied at the same time, these numbers should be taken into account at the same time.
- *failed_capacity:* this occurred whenever an agent did not have enough inventory space to obtain an item. It happened close to 600 times for each *BathTUB* and *PUCRS*.
- *failed_job_status:* this code indicates that an agent tried to either bid for a job that's not up for auctioning, or deliver items to a job which has already been completed by the opponent team. Concerning *Flisvos 2016*, this was their only self-inflicted failure and on top only two times, probably due to *PUCRS* completing a job faster. Vice versa, the same thing happened to *PUCRS* 239 times, almost only against *Flisvos 2016*, indicating the same reason. The third and last team to experience the failure code was *BathTUB* with 85 actions.
- *failed_counterpart:* this occurs whenever an agent tries to assist an assembling agent, but any requirement for assembling is not satisfied. Both teams who tried to assemble items, *Python-DTU* and *lampe*, experienced this failure on a minor scale, i.e., 146 and 328 times respectively.
- *failed_random:* with a chance of 1%, any action might fail and get this result. By the nature of this failure, all teams had a similar rate of around 1% of their total actions.
- *useless:* an agent causes this failure if it tries to deliver items towards the completion of a job, but does not possess any item that is needed. Both *Flisvos 2016* and *Python-DTU* did not cause this failure at all. *BathTUB* tried this only 35 times, *lampe* 74, and *PUCRS* a surprising 438 times.

Finally, the failure codes for invalid agent and job parameters, for an invalid location passed to the `goto` action, for being in the wrong facility, for attempting to assemble an item that cannot be assembled, for bidding on a job that is not an auction, and for using wrong parameter types were not encountered by any agent during the *contest*.

4.4 Feature usage

As already suggested, there are a number of features of the scenario which have been used only to a lesser degree or not at all. Since it was the first time the new scenario was played, it was not entirely clear how the participating teams would handle the different aspects.

For one, the teams did not use the assemble action very much. In preparation of the *contest*, we balanced job rewards so that only by assembling items themselves the teams would be able to net a notable profit. However, in previous tests, the testing agents were not reliably earning money, which led us to the decision to increase base rewards so that buying assembled items in shops still remained a viable though unfavourable option.

In addition, auction jobs were mostly neglected as well (or in case of *BathTUB* did not bring a clear advantage), possibly due to an abundance of regular jobs, which do not have a potential penalty attached (though of course any regular job comes with the risk of the opponent team completing it faster).

The storage facilities were also not used. Probably, the teams were mostly able to deliver the items they bought for their jobs on demand, while stocking up on certain items up front would have been more risk than reward.

Regarding jobs, the teams have used the `post_job` action, yet only to divert the opponent team's attention and add to the information those agents have to process, instead of outsourcing some of their own work.

5 Interesting simulations

In this section, we will have a closer look at particular simulations and how they played out. We also try to carve out a few more details and to draw some conclusions.

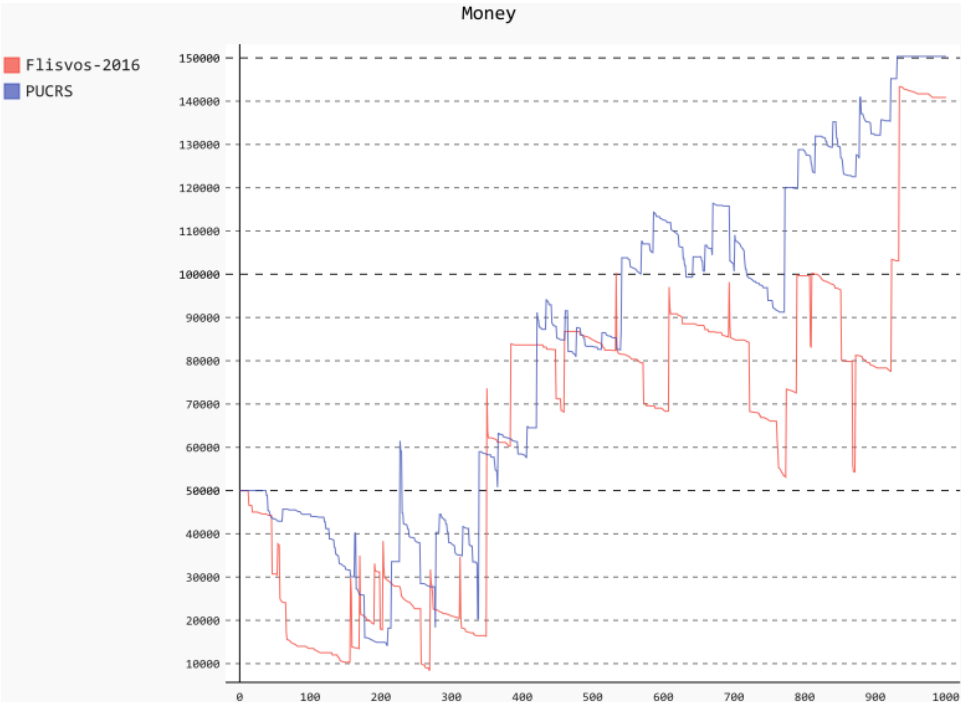
5.1 Flisvos 2016 vs. PUCRS – simulations 1 and 2

The simulations of *Flisvos 2016* against *PUCRS* were played on the second day of the *contest*. While the first simulation was won by *Flisvos 2016*, *PUCRS* was able to take the lead in the second one. Both results were relatively close, see e.g., the development of scores (money) pictured in Figures 11 and 12.

In the first simulation, both teams struggled to make a profit at first. After 100 steps, *PUCRS* was first able to surpass the initial money of 50,000 (but only for a short time), while *Flisvos 2016* was spending a lot of money overall. Around step 500, *Flisvos 2016* overtook *PUCRS* for the first time, only to lose the lead after about 50 steps. This did not change until around step 850, where both teams drew level and remained neck-and-neck for another 100 steps, gaining and spending similar amounts of money, with *Flisvos 2016* slightly in the lead. However, said team was able to complete one final job, rewarding them the certain win.

Figure 11 Money – *Flisvos 2016* vs. *PUCRS* – simulation 1 of 3 (see online version for colours)

Figure 12 Money – *Flisvos 2016* vs. *PUCRS* – simulation 2 of 3 (see online version for colours)



By comparison, *Flisvos 2016* completed four jobs more than *PUCRS*, earning over 100,000 more. As both teams finished with a difference of only around 20,000, we may deduce that *Flisvos 2016* also spent considerably more money. Also from the money charts we can see that the curves for *PUCRS* get flat at around step 900, while *Flisvos 2016* remains active until the simulation is over (an ever slightly decreasing curve indicating charging agents). However, this was one factor giving *Flisvos 2016* the edge in the first simulation, as the last and critical job was completed while *PUCRS* was already in sleep mode. Also, in the second simulation, the final spendings of *Flisvos 2016* were not crucial as *PUCRS* already had the first place when they halted their activities.

5.2 BathTUB vs. lampe – simulations 1 and 2

Each team chooses very few jobs to attempt, giving more weight to every individual decision. *lampe* struggles in the first simulation, barely balancing their account at the end. Successfully completing jobs takes them between 102 and 276 steps from the time they are posted with an average of 186 steps. It can not be ruled out that *BathTUB* beat them to some jobs with much shorter completion times between 66 and 99, averaging 72 steps. Towards the end *BathTUB* makes another big investment and secures a lead of 14,165 by delivering in time.

The pattern is reversed in simulation 2. Here *BathTUB* completes five jobs in 54, 68, 142, 206 and 218 steps. However only one of them turns out to be profitable, not allowing them to break even. Meanwhile *lampe* swiftly completes just two well-chosen jobs in 141 and 30 steps. Simulation 2 ends with *lampe* winning 38,638: 85,114.

Figure 13 Money – *lampe* vs. *Python-DTU* – simulation 1 of 3 (see online version for colours)

5.3 lampe vs. Python-DTU – simulation 1

This simulation is among the most interesting. *Python-DTU* changed their strategy and posted an astonishing number of 2,692 jobs. This not only revealed a weakness in the web monitor[13] but might have had an effect on team *lampe*, which crashed in the first few steps with the result of not much happening for about 800 steps. Fortunately, team *lampe* found the bug, that was preventing them from reconnecting the agents, during the simulation and reconnected between steps 800 and 900. Unfortunately for them, this made their agents spend more money than *Python-DTU* in the last couple of steps, as depicted in Figure 13, ultimately accounting for their loss.

6 Conclusions and outlook

In conclusion, we have seen an interesting *contest* and a solid first run of the new scenario which we can use as a foundation for future improvements. Judging from the previous section, we need to put some effort into pushing the scenario into a more cooperative direction. One way of doing this is to put more emphasis on or even enforce the use of the `assemble` action and the related systems. A rework of this is already in progression and should become ready in early 2017.

Also, the scenario still leaves room for more contention among the opposing teams. This year, the teams could mostly just work alongside each other without having much influence on the other team's possibilities.

As we are already looking forward to the next edition of the *contest* in 2017, we have to admit that our advertising and promotion efforts still leave room for improvement, to say the least. With a stagnating number of participants since 2013, which had its peak in 2011, we have to take action to attract more contestants. Hopefully, this publication helps in this respect and gives us some visibility. Also, we will focus more on pushing for the use of our platform in teaching MAS.

We conclude with a critical remark. It was our aim from the very beginning to eventually show the superiority of agent-based approaches in certain environments where autonomous behaviour of agents pays off. We were quite sure to easily develop scenarios where the use of agents would be not only natural, but also very beneficial in finding a good solution *without using a classical, distributed but hard-coded algorithm developed for that particular scenario*. We were looking forward to a solution that *evolves naturally* by the interplay among the autonomous agents. We were hoping that an underlying agent programming language can provide agent features suitable for allowing such an evolving solution.

However, we never forbade that all agents share the same information or a programmer develops a classical distributed algorithm, where one agents is the master and all other agents are slaves steered by the master (a *centralised* approach). For our scenarios, a solution could, in principle, be hard-coded by hand (although we are not aware of such a solution). This is, obviously, against the philosophy of a truly agent-based paradigm. Our way to deal with this problem is, rather than forbidding such solutions, to develop scenarios where such unwanted approaches are very difficult or, perhaps, not possible at all (or at least not as good as agent-based solutions).

At the same time, we are looking for a scenario that can be easily tested and does not have extremely difficult rules (just difficult solutions). After ten years of research, we

still have not found such a *convincing scenario*. Nor have we yet proved that agent-based approaches are clearly superior to other, sometimes even ad-hoc, approaches using traditional programming languages. This is related to the famous search for a *killer application*, as one of the referees pointed out. It may well be that such killer applications do not exist and that the advantage in using MAS technology only becomes apparent in other categories: reusability, maintenance, bug-freeness, the possibility to model-check agents, code running on different platforms, etc.

A possible way to go is to consider *many* agents, not just a few, but hundreds or thousands of *sophisticated* agents – traditional approaches do not seem to perform well in such a situation. Moreover, with many interacting agents we might see some interesting behaviour evolve.

We are also seriously considering to let participate *more than two* teams in the same simulation. The current scenario would provide for this naturally, however, the underlying technical system has evolved with only two teams in mind, making this change quite a challenge.

Our ultimate vision is an agent platform that allows to deploy agents written in very different agent languages, using the specific features of them. For example it might be beneficial for BDI agents to solve very efficiently certain tasks, whereas planning agents based on some form of HTN could do the planning for them. Being able to re-use agents already developed (and based on different paradigms) would certainly push the envelope for applications of multi-agent systems. However, the price to pay is to standardise the communication and set up common protocols and interfaces for such agents.

Acknowledgements

We would like to thank some anonymous referees for giving us interesting feedback that helped to improve this article. We also thank Alfred Hofmann from Springer for his support during the last ten years and for endowing the prices of 500 and 250 Euros in Springer books.

References

Ahlbrecht, T., Bender-Saebelkampf, C., de Brito, M., Christensen, N.C., Dix, J., Franco, M.R., Heller, H., Hess, A.V., Heßler, A., Hübner, J.F., Jensen, A.S., Johnsen, J.B., Köster, M., Li, C., Liu, L., Morato, M.M., Ørum, P.B., Schlesinger, F., Schmitz, T.L., Sichman, J.S., de Souza, K.S., Uez, D.M., Villadsen, J., Werner, S., Woller, O.G. and Zatelli, M.R. (2013a) 'Multi-agent programming contest 2013: the teams and the design of their systems', in Cossentino, M., El Fallah-Seghrouchni, A. and Winikoff, M. (Eds.): *Engineering Multi-Agent Systems – First International Workshop*, EMAS, St. Paul, MN, USA, 6–7 May, revised selected papers, *Lecture Notes in Computer Science*, Vol. 8245, pp.366–390, Springer.

Ahlbrecht, T., Dix, J., Köster, M. and Schlesinger, F. (2013b) 'Multi-agent programming contest 2013', in Cossentino, M., El Fallah-Seghrouchni, A. and Winikoff, M. (Eds.): *Engineering Multi-Agent Systems – First International Workshop*, EMAS 2013, St. Paul, MN, USA, 6–7 May, revised selected papers, *Lecture Notes in Computer Science*, Vol. 8245, pp.292–318, Springer.

Ahlbrecht, T., Dix, J. and Schlesinger, F. (2014) 'From testing agent systems to a scalable simulation platform', in Eiter, T., Strass, H., Truszczynski, M. and Woltran, S. (Eds): *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation. Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science*, Vol. 9060, pp.47–62, Springer.

Behrens, T., Dastani, M., Dix, J. and Novák, P. (2009) 'Agent contest competition: 4th edition', in Hindriks, K.V., Pokahr, A. and Sardiña, S. (Eds): *Programming Multi-Agent Systems, 6th International Workshop (ProMAS 2008)*, *Lecture Notes in Computer Science*, Vol. 5442, pp.211–222, Springer.

Behrens, T., Dastani, M., Dix, J., Hübner, J., Köster, M., Novák, P. and Schlesinger, F. (2012a) 'The multi-agent programming contest', *AI Magazine*, Vol. 33, No. 4, pp.111–113.

Behrens, T., Köster, M., Schlesinger, F., Dix, J. and Hübner, J. (2012b) 'The multi-agent programming contest 2011: a résumé', in Dennis, L., Boissier, O. and Bordini, R. (Eds.): *Programming Multi-Agent Systems*, *Lecture Notes in Computer Science*, Vol. 7217, pp.155–172, Springer Berlin/Heidelberg.

Behrens, T., Dastani, M., Dix, J., Köster, M. and Novák, P. (2010a) 'The multi-agent programming contest from 2005–2010: from collecting gold to herding cows', *Annals of Mathematics and Artificial Intelligence*, Vol. 59, No. 3, pp.277–311.

Behrens, T., Dastani, M., Dix, J., Köster, M. and Novák, P. (Eds.) (2010b) 'Special issue about multi-agent-contest I', *Annals of Mathematics and Artificial Intelligence*, Vol. 59, Nos. 3–4, pp.275–437, Springer, Netherlands.

Cardoso, R.C., Pereira, R.F., Krzisch, G., Magnaguagno, M.C., Baségio, T. and Meneguzzi, F. (2017) 'Team PUCRS: a decentralised multi-agent solution for the agents in the city scenario', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.3–34.

Czerner, P. and Pieper, J. (2017) 'Multi-agent programming contest 2016: lampe team description', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.101–117.

Dastani, M., Dix, J. and Novák, P. (2007) 'The second contest on multi-agent systems based on computational logic', in Inoue, K., Satoh, K. and Toni, F. (Eds.): *Computational Logic in Multi-Agent Systems*, *Lecture Notes in Computer Science*, Vol. 4371, pp.266–283, Springer Berlin Heidelberg.

Dastani, M., Dix, J. and Novák, P. (2008) 'Agent contest competition – 3rd edition', in Dastani, M., Ricci, A., El Fallah Seghrouchni, A. and Winikoff, M. (Eds.): *Proceedings of ProMAS '07*, revised selected and invited papers, *Lecture Notes in Artificial Intelligence*, Vol. 4908, Springer, Honululu, US.

Hessler, A. (2017) 'BathTUB team description – multi-agent programming contest 2016', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.118–127.

Karakovskiy, S. and Togelius, J. (2012) 'The Mario AI benchmark and competitions', *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 1, pp.55–67.

Köster, M., Schlesinger, F. and Dix, J. (2013) 'The multi-agent programming contest 2012', in *Programming Multi-Agent Systems*, *Lecture Notes in Computer Science*, Vol. 7837, pp.174–195, Springer Berlin Heidelberg.

Sarmas, E. (2017) 'The Flisvos-2016 multi-agent system', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.35–57.

Villadsen, J., From, A.H., Jacobi, S. and Larsen, N.N. (2017) 'Multi-agent programming contest 2016 – the Python-DTU team', *Int. J. Agent-Oriented Software Engineering*, Vol. 6, No. 1, pp.86–100.

Notes

1 https://multiagentcontest.org.

2 http://aichallenge.org/.

3 http://aiolympics.ro/.

4 http://sscaitournament.com/.

5 http://ipc.icaps-conference.org/.

6 http://www.robocup-logistics.org/sim-comp.

7 http://games.stanford.edu/.

8 http://tac.sics.se/.

9 The 2014 contest was an "unofficial" edition (i.e., no publications and prizes, only glory) with no changes to 201.

10 https://www.openstreetmap.org.

11 Fortunately, only the web monitor had to be quickly patched to accommodate for the increased data volume. The *MASSim* server was able to handle the situation well.

12 Due to the random generation of simulations, score (i.e., money) comparisons on the *contest* level can only serve as estimations.

13 Increasing the JVM's stack size for the monitor remedied the problem.