Noname manuscript No.
(will be inserted by the editor)

Multi-Agent Programming Contest 2017
The twelfth edition of the MAPC

Tobias Ahlbrecht · Jürgen Dix · Niklas Fiekas

Received: date / Accepted: date

Abstract We present the twelfth edition of the Multi-Agent Programming Contest[1], an annual, community-serving competition that attracts groups from all over the world. Our contest facilitates comparison of multi-agent systems and provides a concrete problem that is interesting in itself and well-suited to be tackled in educational environments. This time, seven teams competed using strictly agent-based as well as traditional programming approaches.

Keywords multi-agent systems · programming · competition

1 Introduction

In this introductory article to the special issue, we (1) briefly introduce our Contest and its development, (2) explain the scenario with a focus on the changes that were necessary compared to the 2016 edition, (3) introduce the seven teams that took part in the tournament, (4) analyze and interpret a few interesting matches, and (5) evaluate each team's performance and strategy from a bird's-eye view.

The Multi-Agent Programming Contest (MAPC) is an annual international event. It was initiated in 2005 by Jürgen Dix and Mehdi Dastani (with a lot of help from Peter Novak who joined the following years). In 2017, the competition was organized and held for the twelfth time. The goal of the Contest is 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, and (4) compiling test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. Moreover, we aim to support educational efforts in the design

Department of Informatics
Clausthal University of Technology
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld, Germany
E-mail: {tobias.ahlbrecht,dix,niklas.fiekas}@tu-clausthal.de

[1] `https://multiagentcontest.org`

and implementation of MAS's by preparing our ready-made environment which provides a concrete problem for agent systems to solve.

For the Contest each participating team develops a team of agents which remotely connects to our Contest server where the current scenario (i.e. the game) is being run. The Contest server sends the current game state in the form of percepts to each agent and expects an executable action in return. The gathered actions are executed and the game state is advanced. This cycle is repeated until a predefined number of steps is reached.

Detailed information about the strategies of the teams from each team's own perspective can be found in the subsequent papers in this volume.

1.1 Related work

For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to [3,6,10,11,7,5,8,14,2,1]. A quick non-technical overview appeared in [4].

There is a good number of agent and more general AI contests, competitions and challenges which have been organized in the past decade.

Firstly, the *Trading Agent Competition*[2] is one of the few strictly agent focused contests. Here, agents have to solve trading related problems, e.g. managing a supply chain. Since 2012, the related *Power Trading Agent Competition*[3] specializes on agents trading in the energy market. Each team only consists of a single "broker" agent.

Secondly, the *General Game Playing*[4] competitions do not focus on team-based games which require interaction and cooperation. Interestingly, the game is never known to the participants beforehand. Instead, a game description language is employed and agents have to understand each game's rules themselves.

Google's AI challenge[5] provides rather simple games requiring good strategies. Unfortunately, the last challenge was in 2011.

The *AI-MAS (Winter) Olympics*[6] features a more sophisticated scenario called "Crafting Quest 3". As its name implies, agents are among the desired solution approaches. The solutions have to be implemented in Java however.

A number of *Planning Competitions*[7], e.g. the *RoboCup Logistics League*[8], focus mostly on the single aspect of planning in agent systems.

Finally, there are a lot game-based challenges, some with existing games, like the *Student StarCraft AI tournament*[9] or the *Mario AI Championship* [13], and some with games created specifically for the challenge, like *BattleCode*[10]. All of these games impose some more or less hard restrictions on the program playing the game. Some have to play the game in "real time" while others require a specific

[2] https://www.sics.se/projects/trading-agent-competition
[3] http://powertac.org/
[4] http://games.stanford.edu/
[5] http://aichallenge.org/
[6] http://aiolympics.ro/
[7] http://ipc.icaps-conference.org/
[8] http://www.robocup-logistics.org/sim-comp
[9] http://sscaitournament.com/
[10] https://www.battlecode.org

programming language or limit the amount of bytecode (i.e. instructions on the Java Virtual Machine) that can be used.

To summarize, our Contest differs from other competitions by

- allowing any programming language or approach to be used,
- not focusing on solutions for a particular problem domain, and
- featuring a rather complex game which requires multiple agents to coordinate their actions.

2 MAPC 2016 and 2017: The City scenario

In this year's scenario[11], two teams of 28 agents try to earn as much virtual money as possible. To do this, they have to navigate through realistic city graphs, imported from *OpenStreetMap*[12], and complete as many high-valued *jobs* as possible. These jobs are randomly generated and require the agents to obtain a certain number of different *items* which have to be delivered to one of multiple *storage* locations. Items can be either bought in *shop* facilities or gathered from *resource nodes*. These items however are only the source material for constructing *assembled items* in a *workshop* facility. So, items have a name, a volume and optionally a number of requirements, i.e. which other items and *tools* are required to assemble one instance of this item. Jobs only request assembled items.

Each agent belongs to a role, which defines its maximum *battery charge*, *loading capacity* and which *tools* the agent is allowed to use. In order to increase speed and decrease capacity the roles are: trucks, cars, motorcycles, and drones. Drones are the only vehicles that are not bound to streets. To recharge their batteries, agents can visit *charging stations* which restore a certain amount of charge per step.

Each job has a start time, an end time, a reward specifying how much money a team receives upon successful completion, and a number of requested items. There are now three different kinds of jobs. Firstly, regular jobs do end once the first team has delivered all requested items or its time limit is reached. Either way, the second team cannot continue completing the job anymore. If a job ends, each team that did not complete it may retrieve the items it already delivered towards the partial completion of said job.

Secondly, *auctions* are preceded by an auctioning phase which lasts a couple of steps. During this phase, agents may place their bids. After the phase, the job gets assigned to the team which bid the lowest amount of money. This is because this value is also the reward the team will receive upon successful completion. Auctions are advantageous since the assigned team does not have to fear the other team completing it first. On the other hand, a team has to pay a fine if it does not complete an assigned auction, since it could hoard auctions otherwise.

Thirdly, the new *missions* are like regular jobs, only that one "private" instance is created for each team. Similar to auctions, teams also have to pay a fine if they do not complete missions. This is to ensure that ignoring missions is discouraged.

Points for the overall competition are distributed at the end of each simulation. The winning team gets 3 points, unless it did not make a profit, in which case it only receives 2 points. In case of an unlikely draw, each team is awarded 1 point.

[11] the complete documentation is linked from `https://multiagentcontest.org/2017/`

[12] `https://www.openstreetmap.org`

This year, with 7 teams and 3 simulations for each combination of teams, there was a maximum obtainable score of 54 points.

The strengths of the scenario lie in the amount of coordination and planning that is required among agents from the same team. Then again, as noted last year already, interaction with the opponent team is quite limited, apart from possible resource contention in terms of available items and jobs (externalities).

In the next section, we take a closer look at the improvements compared to the first implementation of the scenario.

3 Changes for 2017

Drawing from our experiences with the 2016 edition, we made a number of changes to reinforce certain aspects of the scenario and alleviate some of its previous shortcomings. As always, we scaled the scenario up: it now features 28 agents per team. Previously, we had four agents per role. The number for each role was doubled, except for drones, to put more emphasis on slower vehicles.

3.1 Cooperation

Last year's instance of the scenario has shown that agents do not cooperate at all (not even with agents of the same team) if they are not explicitly required to do so. While the `assemble` action was designed to be such an explicitly cooperative element, using it was only encouraged but not required. Thus, many teams (if not all) did not even implement some assembling functionality for their agents and made do with jobs where no assembly was required.

To get the agents to more cooperation, we assured that each job created by the system requires only items which need assembly. So, to make some profits, the teams now have to make plenty of use of the `assemble` action.

3.2 Proactiveness

In the previous Contest, we observed rather long phases of the simulation where agents were only waiting for the next profitable job, effectively doing nothing else. Therefore, we wanted to reward those agents which actively prepare for possible future events. To this end, we made two changes.

On the one hand, we removed the cost that was associated with charging agents. This now allows the agents to travel the map and acquire or move items without getting a large penalty if these preparations turn out to be useless in the end.

On the other hand, `resource nodes` were introduced to give agents an additional occupation when they are not actively pursuing a job. `Resource nodes` are associated with a special `resource` item that agents can obtain by using the `gather` action at a node's location. Gathering one instance of the resource takes a fixed number of actions, so employing more agents will yield more resources. If agents from different teams work on the same node, however, the result will depend on

the order in which the actions are executed (which is reshuffled for each step), so that agents cannot predict the outcome.

Compared to shops, items from `resource nodes` will take more time to be acquired but have no monetary cost associated: they are freely available.

In summary, agents now have valid things to do in their off time, which are very likely to give their team a serious advantage over a team which only waits during those times.

3.3 Comparability

The 2016 parameter configuration allowed for vastly different simulation instances. While this was good for providing a wide variety of different simulations, it made it also more difficult to compare different simulation runs. This year, we modified the complete random generation mechanism and set new parameters, so that simulation instances with the same parameters still differ in their details while retaining an overall more uniform shape (e.g. in the number, structure and value of items). We could have opted to generate identical simulation instances each time, however, this would give teams who play later an unfair advantage, as they could learn and therefore predict exactly what happens in later simulations.

Also, the new *missions* were devised to directly compare agent teams. Since both teams get the exactly same mission at the same time, we can compare the time it took each team to complete the mission to get a first performance indicator (aside from which team won the game).

3.4 Uncertainty

Finally, to increase the amount of uncertainty in the environment and therefore reward agent teams which are more adaptable, we introduced blackouts at `charging stations`. With a small probability, each `charging station` may stop working for a couple of steps now. Agents can find out about the state of a `charging station` only by attempting to charge at that facility and subsequently failing.

4 The tournament

The tournament took place during four days, from 18th to 21st September. As always, the main phase was preceded by a qualification phase to ensure that all teams were able to receive messages from and send actions to the contest server. Each team had to achieve a failure rate below 5% to be admitted to the Contest itself. This did not prove to be a big hindrance to any one of the seven teams.

4.1 Simulation definition

Each simulation once again consisted of 1000 discrete steps. The three simulations of each match had different parameters to test a variety of different settings.

However, each match had the same set of parametrized simulations to facilitate comparability among different matches (see also Section 3.3).

We chose the concrete parameters to ensure certain evolutions in the simulations' difficulty. The first simulation of each set was played on a rectangular excerpt from the street graph of Paris and intended to be the "easiest" one to play. From there, we increased the size of the map for the second simulation (Tokyo) and again for the third one (Mexico City). Bigger maps imply more facilities and more available items (because there are more shops and resource nodes) but at the same time longer routes to arrive at specific locations. Additionally, we increased the difficulty level of jobs for each simulation to require more distinct and more complex items per job. Also, we shortened the average time limit on each job.

4.2 Participants and results

This year, the team count went up slightly: seven teams from around the world registered and made it through the qualification phase (see Table 1).

Team	**Affiliation**	**Platform/Language**
BusyBeaver	Clausthal University of Technology	Pyson (Jason)
Chameleon	Shahid Beheshti University	Java
Flisvos 2017	none	Python
Jason-DTU	Technical University of Denmark	Jason
lampe	Clausthal University of Technology	C++
SMART-JaCaMo	Pontifical Catholic University of Rio Grande do Sul	Jason, CArtAgO, Moise
TUBDAI	Technical University of Berlin	Python

Table 1: Participants of the 2017 Edition.

Table 2 summarizes the results of this year's Contest. *BusyBeaver* secured a flawless victory, not losing a single simulation, thus netting the maximum number of 54 points. Tied for second place are *Flisvos 2017* and, once again, *Jason-DTU* with 36 points. A close fourth place went to *SMART-JaCaMo* with only 3 points

Pos.	**Team**	**Points**	**Simulations won**
1	*BusyBeaver*	54	18
2	*Flisvos 2017*	36	12
2	*Jason-DTU*	36	12
4	*SMART-JaCaMo*	33	11
5	*lampe*	16	6
6	*TUBDAI*	6	2
7	*Chameleon*	4	2

Table 2: Results.

less. Still almost half those points brought team *lampe* the fifth place and *TUBDAI* made sixth place with 6 points still. Finally, the seventh place went to team *Chameleon* due to lots of problems on their side.

Most of the simulations awarded 3 points. Only *lampe* did not turn a profit in two simulations against *TUBDAI* and *Chameleon* but still won by spending less, thus only getting 2 points for those simulations.

We gratefully acknowledge Springer's support over the years (many thanks to Alfred Hofmann who is responsible): the winner gets a voucher worth 500 Euros (in Springer books) and the runner-up one worth 250 Euros.

4.3 The teams and their agents

In this section we collect information about the participants[13], their development efforts and their agent team strategies from a high-level viewpoint. For more details, we refer to each individual team description paper in this special issue.

BusyBeaver: The team *BusyBeaver*, one of the two single person groups, used the Pyson[14] agent platform which is an implementation of AgentSpeak based on Jason [9]. Around 300 hours were spent over the course of 3 to 4 months, of which 200 were dedicated to implementing the system. The agents' strategy is a heavily proactive one, always trying to assemble items in advance. A leading truck agent does most of the planning. Coordination is mostly achieved through the contract net protocol. The agents are further statically assigned to groups for acquiring, assembling and delivering items.

Flisvos 2017: *Flisvos 2017*, the other single person team, invested an approximate 120 hours into producing a final result of almost 5000 lines of code. As last year, the agents are written in Python and completely controlled by a centralized "mind". Their strategy is to complete the jobs which have all their requirements already available (in shops or carried by the agents themselves). Proactively, idle agents try to occupy the shops or even buy some items which might be useful in the future.

Jason-DTU: The team *Jason-DTU* consists of 4 people, 2 master students and 2 supervisors, who invested a combined 300 hours into the project. Their agents are implemented using the Jason agent framework with a rather big part written in Java. Jobs are assigned to a group of agents by a centralized entity, where each group is then responsible for deciding how to complete that job.

SMART-JaCaMo: The 7 members of team *SMART-JaCaMo* spent approximately 120 hours, mainly improving their team from 2016, resulting in 4354 lines of code. With that, their code base has doubled after all. The agents also use the contract net protocol for coordination (similar to *BusyBeaver*) and additionally features of MOISE [12], an "organisation oriented programming framework" for multi-agent systems.

lampe: The *lampe* team again consists of two people, one of which also constitutes the team *BusyBeaver*. *lampe* dedicated around 200 hours to achieving a final code line count of 8512. Similar to *Flisvos 2017*, these C++ agents are

[13] Unfortunately, we will not report on the *Chameleon* team as they decided to leave the Contest after having played their matches.

[14] `https://github.com/niklasf/pyson`

Team	hours	lines	coordination
BusyBeaver	300	2000	mostly centralized
Flisvos 2017	120	5000	centralized
Jason-DTU	300	5361	mostly centralized
SMART-JaCaMo	120	4354	decentralized
lampe	200	8512	centralized
TUBDAI	300	5056	decentralized

Table 3: Key details.

also centrally controlled. The team's strategy is to internally simulate multiple strategies and chose the one with the best outcome based on custom heuristics.

TUBDAI: The 4 people of team *TUBDAI* spent around 300 hours implementing 5056 lines of Python code with the goal to evaluate their decision-making and planning framework from a multi-robot context. The agents form 4 static groups who work on jobs independently, only communicating to avoid working on the same job.

Chameleon: Unfortunately, the team *Chameleon* failed to submit anything after the Contest. Due to this and coupled with their performance, we will exclude them from most of the following analysis.

Some of the key details have been summarized in Table 3. From these numbers, we can see that all participants who have started a new team (i.e. a new codebase or approach) have estimated their efforts at roughly 300 hours. At the same time, the teams who evolved their team from last year invested "only" 120 to 200 hours.

5 Performance of the teams

In this section, we will take a closer look at the outward behavior of each team and how it affected the outcome. Key indicators we will analyze include the actions an agent team used, as well as the number and types of jobs a team completed and how long it took the agents to complete jobs in general. In the end, we will look at the agents' stability in terms of action failure codes.

BusyBeaver: The Contest's winning team distinguishes itself by overall very fast job completion times. In most simulations, the average completion time was below that of the opposing team. In the few cases where the other team had a better average time, that team only completed a minor amount of jobs, which makes it easier to have a good average time. This fact also enabled the team to complete significantly more jobs than each other team. Looking at the comparison of mission times (i.e. the identical jobs that each team had to complete at the same time), we again see that *BusyBeaver* was way faster at dealing with most of the missions. Regarding auctions, we can see no clear strategy. The team either did a lot of or almost none of the auctions per simulation, but never a medium amount.

When we look at the team's action counts, we see a naturally high number of assembly related actions due to the high number of completed jobs. Also, the actions `give` and `receive` where used frequently indicating a good division of work. The only other team to make use of these actions was *lampe*, however, to

a lesser degree. *BusyBeaver* did not use actions related to storing items. Instead, the `truck` agents seemingly served as mobile storage units. Additionally, the team used a lot of `gather` actions to get "free" resource items. They make up around 10% of the team's total actions, which agrees with the team's overall perceived proactivity.
A typical action distribution for *BusyBeaver* is given in Figures 1 and 2.

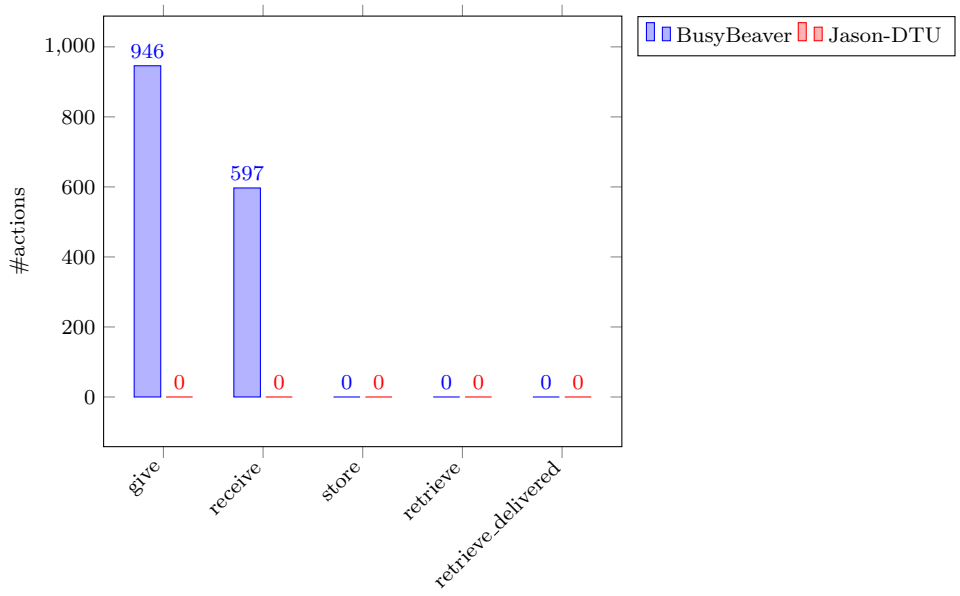


Fig. 1: *BusyBeaver* vs. *Jason-DTU* sim 2 item transfer actions

To conclude, the team's high (but still reasonable) proactivity has clearly paid off.

Jason-DTU: The agents of *Jason-DTU* completed a fair amount of jobs somewhere in the region of one half to two thirds of all completed jobs per simulation against teams of the same or lower ranking. However, the average time spent working on each job does not deviate much from those opponent teams' times. Often, it is even a few steps more in comparison. Looking at average mission times, exemplarily listed in Table 4, reinforces the picture: there is no clear trend for either team.

Mission	Reward	Jason-DTU	SMART-JaCaMo
1	2716	62	55
2	3008	31	30
3	1548	17	24
4	2257	29	32

Table 4: *Jason-DTU* vs. *SMART-JaCaMo* sim 2 mission times

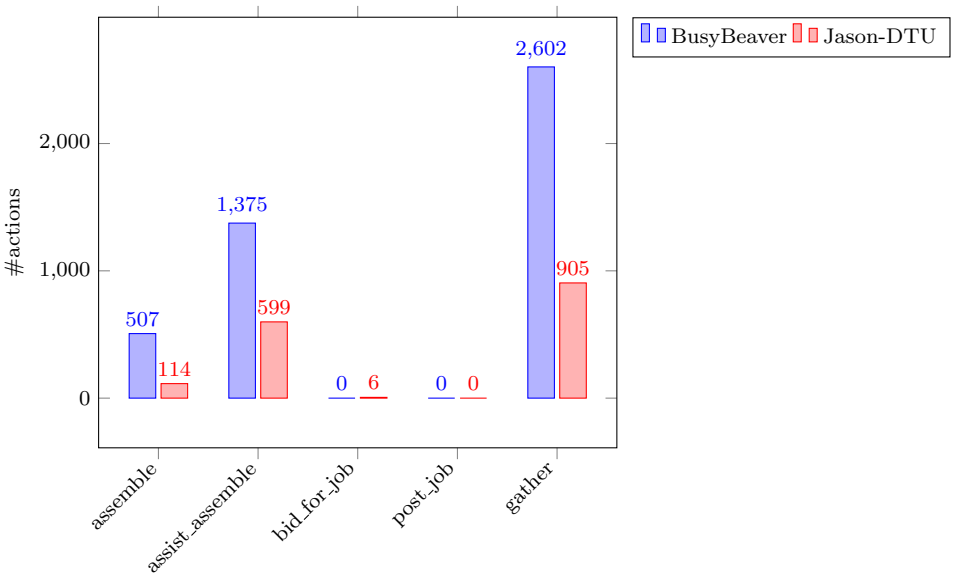


Fig. 2: *BusyBeaver* vs. *Jason-DTU* sim 2 job related actions

Auction jobs seem to have played a key role in gaining the edge over *Flisvos 2017* in one out of three simulations.

Again looking at the agents' actions, we see the second highest amount of `gather` actions after *BusyBeaver*, suggesting again that proactively gathering materials paid off after all. Aside from that, the team did not use the actions `give` and `receive` or `store` and `retrieve` at all. A typical action distribution for one simulation can also be seen in Figures 1 and 2.

Flisvos 2017: The other team to share the second place, *Flisvos 2017*, completed more jobs than *Jason-DTU* at comparably average times. The agents completed most of the jobs against *SMART-JaCaMo*, except for their third simulation, where *SMART-JaCaMo* was able to get two more regular jobs done and win in a photo finish. Unfortunately, *Flisvos 2017* did not consider auction jobs at all. The team's action profile shows a team focused on "core functionality". The agents never used actions for storing items and neither for transferring items between agents. Also, as noted, actions related to auctions were not used at all. The same holds for the `gather` action.

SMART-JaCaMo: The team *SMART-JaCaMo* completed a rather low number of regular jobs with average completion times on par with *Jason-DTU*. In the match against *Jason-DTU* we see that *SMART-JaCaMo* begins to shine when more auctions are involved. In their first simulation, they were able to complete 22 relatively high-valued auctions, while *Jason-DTU* only completed 6, which let them capture a victory despite having completed only half the amount of regular jobs as *Jason-DTU*.

Interestingly, the agents of *SMART-JaCaMo* were the only ones to employ the `store` and `retrieve` actions in their strategy. Unfortunately, this does not reflect in faster average completion times. The team also did not use the new

`gather` action. A sample action profile for this team can be seen in Figures 3 and 4.

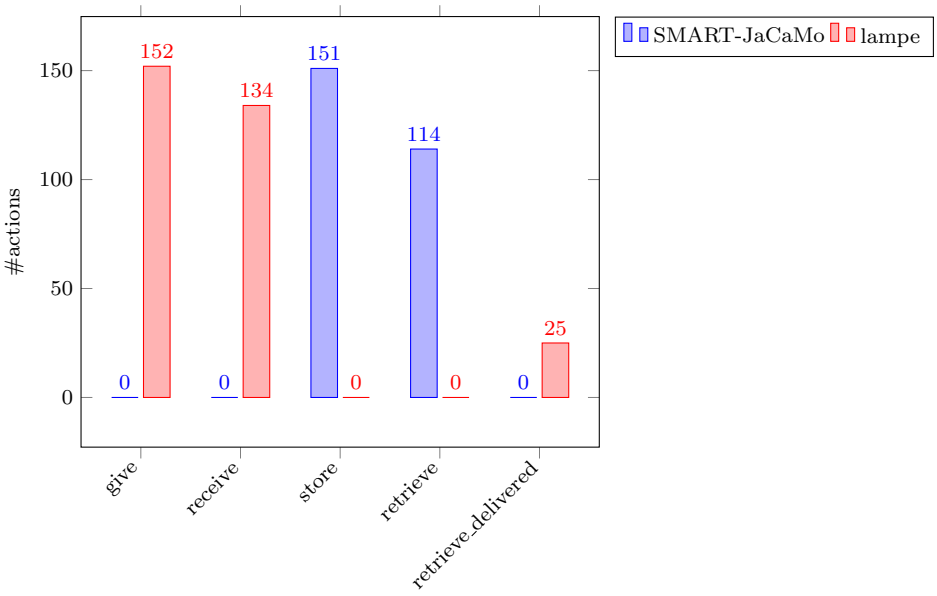


Fig. 3: *SMART-JaCaMo* vs. *lampe* sim 3 item transfer actions

lampe: The *lampe* team got a number of regular jobs done similar to *SMART-JaCaMo*, however, the agents were outclassed regarding auctions. Against *Flisvos 2017*, the team mostly completed jobs with a relatively low reward, however, the team could win one of these simulations due to many high-valued auctions. The average time spent on each job was comparable to the opponent team in some simulations and exceptionally high in others.
lampe was the only team to make use of the `give` and `receive` actions, which indicates a good degree of cooperation among agents. Also, *lampe* used the `retrieve_delivered` action quite often, which might be associated to the high job completion times in some simulations. The `gather` action was also not used by *lampe* at all. A typical action distribution for the team is also given in Figures 3 and 4.

TUBDAI: The team *TUBDAI* completed rather few jobs overall, but almost half of a simulation's jobs in some cases. However, this was mostly outweighed by high spendings. The average job completion time is also not exceptionally high and even better in direct comparison in the match against *lampe*, which lends further credibility to unusual high investments. Overall, the team did not complete many auction jobs.
TUBDAI used the `gather` action to some degree and, as many other teams, did not make use of `give` and `receive` or `store` and `retrieve`.

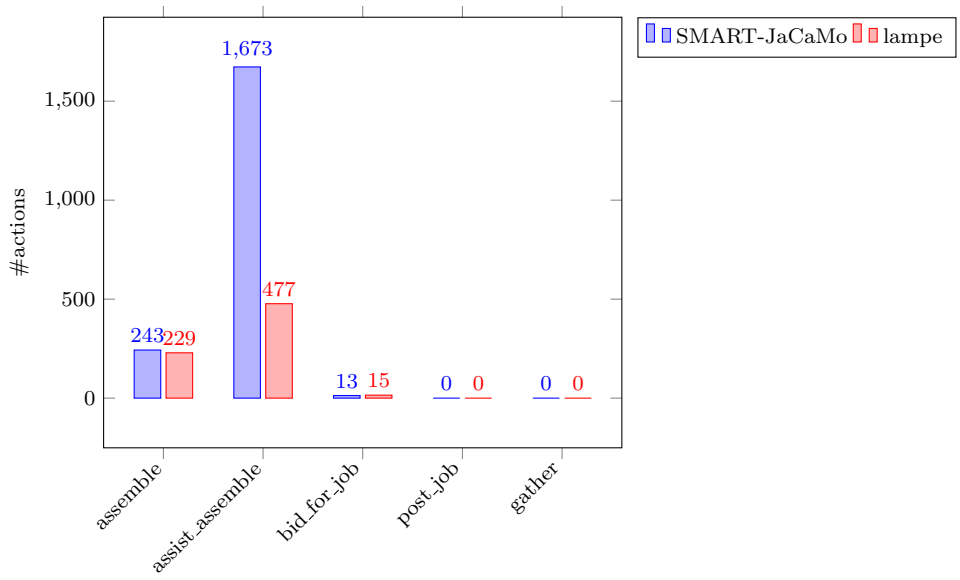


Fig. 4: *SMART-JaCaMo* vs. *lampe* sim 3 job related actions

5.1 Agent stability

Each agent action returns either *successful* or one of a number of failure codes which are specific to each action type. We recorded the total counts of each failure code per team, listed in Table 5, to get an understanding of where each team might have had some problems and where teams acted more reliably than others.

Reason	lampe	Jason DTU	Chameleon	TUBDAI	Flisvos	Busy Beaver	SMART JaCaMo
wrong_param	-	-	2391	1	-	1	-
wrong_facility	-	-	2	813	-	700	-
failed_tools	53	4889	18446	29112	560	914	1451
failed_location	-	53	4087	1882	-	-	-
failed_capacity	8	466	1054	1293	-	830	-
no_route	1058	1	56	727	-	-	-
item_amount	55	1415	2332	8716	57	148	815
job_status	133	145	63	1825	24	241	126
failed	5110	4996	5040	5067	5112	5008	5046
useless	325	-	-	320	2	134	-
unknown_agent	-	-	8741	-	-	-	-
counterpart	171	8154	4314	42835	705	30346	4072

Table 5: Total failure counts

The code `wrong_param` indicates a general problem in the implementation, where action parameters are not correctly set or sent to the server. Fortunately, only the

team *Chameleon* suffered a noteworthy amount of this error. The other teams encountered no or in the worst case one instance, which is very good considering that each team had to submit 168,000 actions over the course of 6 matches.

The `wrong_facility` failure occurs when an agent tries to perform an action that is related to a specific facility, e.g. `assemble` in a `workshop`, while it is not located in such a facility. This was mainly encountered by *TUBDAI* and *BusyBeaver* and may indicate a minor lack in adaptivity, that is, agents having beliefs that contradict the actual environment state.

`failed_tools` shows a problem with item assembly. This was most prominently encountered by *TUBDAI*, similar to other assembly related failure codes like `counterpart`, which may occur when the assembly initiator failed or `item_amount`, which is basically the same as `failed_tools` but for items. The reason for this is most likely that their agents started to send `assemble` and `assist_assemble` actions as soon as they were ready regardless of whether each agent planned for the assembly was ready. With some more communication, this could have been optimized, e.g. by recharging agents. The numbers for *Jason-DTU* show a similar trend, only less pronounced. Finally, all teams showed a considerable number of these failures with *lampe* having the best numbers and *Flisvos 2017* also being very good.

The `job_status` code can have multiple reasons, but most frequently it is due to an agent trying to deliver items towards a job that has already ended, either regularly or having been completed by the opponent. Most teams encountered rather few of those, between 24 and 241, while *TUBDAI* had the highest count with 1825 indicating some potential for optimization.

The `useless` result is only received when an agent tries to assist with assembly but has no items to contribute. This may point to either agents having "forgotten" to execute some sub-task or situations where another agent who is also part of the assembly can also contribute these items. The latter case, being more likely, shows again where agents could have done something more *useful* with their time. Fortunately, this failure was only encountered around 100 to 300 times by *lampe*, *TUBDAI* and *BusyBeaver*.

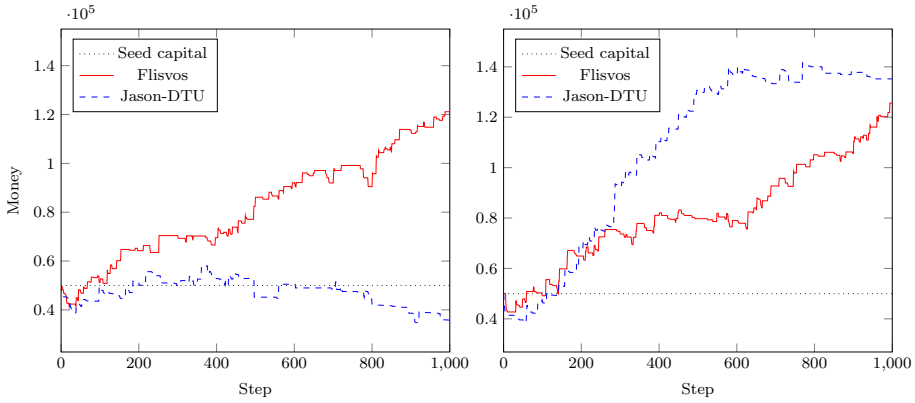
To conclude, we did not see high failure counts in this year's Contest. Even the numbers that appear quite high in comparison cannot be associated with some deep underlying problems but only with some more aspects which could have been optimized. The "best" results have been recorded for *Flisvos 2017* closely followed by *lampe*.

6 Interesting simulations

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

6.1 *Flisvos 2017* vs. *Jason-DTU*, *SMART-JaCaMo*, *lampe*

Looking at all simulations, we can see that teams mostly performed uniformly, i.e. in any given matchup, most often one team won all of its 3 simulations. However, *Flisvos 2017* lost one simulation each against *Jason-DTU*, *SMART-JaCaMo* and *lampe*.



(a) Sim 1 (b) Sim 2

Fig. 5: Money curves *Flisvos 2017* vs. *Jason-DTU*

For example, *Flisvos 2017* clearly won the first simulation against *Jason-DTU*, while *Jason-DTU* was able to narrowly win their second encounter (compare Figures 5a and 5b).

Comparing all three simulations between *Flisvos 2017* and *Jason-DTU*, we see that *Jason-DTU* usually gets to complete around 20% of regular jobs. In the second simulation however, there was a bigger amount of auction jobs. *Flisvos 2017* earned a total of 148k without completing any auction jobs. *Jason-DTU* made 42k from regular jobs and another 106k from auction jobs alone. Additionally, both teams earned 9k from the missions. So, the higher number of auction jobs allowed *Jason-DTU* to receive as much money as *Flisvos 2017*, but *Jason-DTU* still finished about 10k in the lead. Thus, we may conclude that *Jason-DTU* could choose more profitable jobs, i.e. those were the necessary investments (to buy items) were lower.

6.2 *BusyBeaver* vs. *Jason-DTU* simulation 1

In general *BusyBeaver* manages to complete jobs much faster than *Jason-DTU* (an average of 7 steps compared to 33 steps, and similar times for the identical missions in this simulation). This difference could be explained by more proactive behavior. Frequent usage of `give` and `receive` also indicates better coordination.

Nonetheless this simulation was closely contested and the money curve (Figure 6) shows an interesting progression: The strategies differ already in the opening phase. At simulation step 10 *BusyBeaver* has invested more than 20% of their starting capital in a variety of 10 different items. Meanwhile *Jason-DTU* chose a more conservative approach and sent their agents to gather free resources.

The first half of the simulation consists of *BusyBeaver* trying to break even with their initial investment. After that *Jason-DTU* is still ahead and now also has a varied repertoire of collected items at their disposal. However in the second half jobs get more rare, making it more likely that both teams race to finish the same job. This is where the faster job completion rate really started to shine and eventually *BusyBeaver* came out on top. All in all *BusyBeaver* completed 31 regular

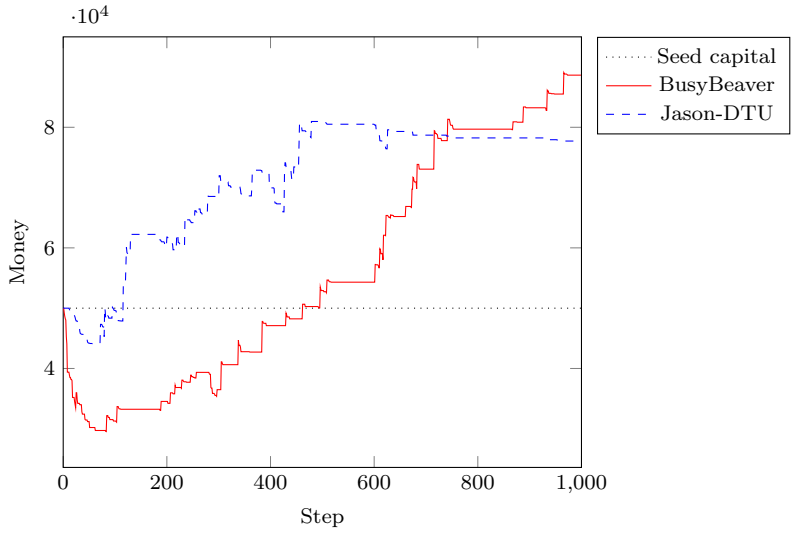


Fig. 6: Money curves for sim 1 of *BusyBeaver* vs. *Jason-DTU*

jobs at an average reward of 2697. *Jason-DTU* completed 17 higher paying jobs at an average reward of 3367 and 4 auctions.

7 Conclusion and outlook

Once again we have seen an interesting Contest with very different teams, using agent-based as well as conventional programming approaches and frameworks. Of course, we are happy that once more an agent-based approach took the crown.

We evolved our scenario based on feedback and observations from the previous edition, however, there are still some points we need to address for future editions. Firstly, the parameters for the current scenario are still difficult to set in the "right" way. The simulations should neither be unmanageable, nor should simple fixed strategies lead to an easy win.

This year, we saw a rather simple approach winning, probably enabled by a too conservative parameter range. Our intention was to not make the scenario too difficult, since last time, assembly was a big problem and not used to a satisfying degree. This time, mandatory assembly already made for a significant increase in difficulty.

Looking at previous scenarios, they only required a handful of parameters. While this was indeed restrictive on the variety of simulation instances, those remaining instances were far more likely to be "good" ones, as the game itself had to be viable in all circumstances.

For example, the *Agents on Mars* scenario only required a graph structure with a certain connectedness. The game itself was more or less the same on any graph then, which has both good and bad sides, as noted above.

One of our goals for our next scenario therefore is to find a better balance between having too many parameters, making the game prone to suboptimal con-

figurations, and too few parameters, leaving only a small spectrum of possible simulation instances.

In the future, we would also like to improve the spectator's experience. In the current scenario, it is very difficult to track what happens, even with a few seconds between steps, so that most people only track the monetary score and otherwise only look for obviously erratic behavior of agents. In previous scenarios, it was much easier to see *where* the interesting behavior was emerging. Of course, this is not only a bad thing. A scenario that is hardly traceable by human users can make for a good one to be solved by agent systems.

As always, we wish (and plan) for a scenario that facilitates automatic analysis. Once again, we used a number of statistics to derive conclusions about each agent team. However, it would be great if the scenario could automatically highlight interesting situations. Some work on automated detection of such situations or emergent behavior in general has already been done, e.g. in [16] or [15].

In a similar direction, we are still mostly looking at agent implementations as opposed to benchmarking agent platforms. To get a notion of the suitability of each platform, we refer to each participating team's subsequent paper and especially the questions and short answers part we had each team fill in.

Last but not least, we still have not found a magic formula that allows us to make the scenario inherently benefit purely agent-based approaches. As we do not prevent traditional programming approaches from entering, such *centralized* systems are still being implemented and achieve good results (although empirically they have a fairly low chance of winning the Contest).

To conclude, we improved last year's scenario to address some of its issues, while the core challenges regarding our scenario design still remain to be solved, hopefully by one of our future games. We are also happy to note that our platform is used in quite a number of educational efforts: most of this years' teams arose from student projects or courses.

Acknowledgements We would like to thank Alfred Hofmann from Springer for his continuous support for more than 10 years now, and for endowing the price of 500 Euros in Springer books.

References

1. 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, Ø.G., Zatelli, M.R.: Multi-Agent Programming Contest 2013: The teams and the design of their systems. In: M. Cossentino, A.E. Fallah-Seghrouchni, M. Winikoff (eds.) Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 8245, pp. 366–390. Springer (2013)
2. Ahlbrecht, T., Dix, J., Köster, M., Schlesinger, F.: Multi-Agent Programming Contest 2013. In: M. Cossentino, A.E. Fallah-Seghrouchni, M. Winikoff (eds.) Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 8245, pp. 292–318. Springer (2013)
3. Ahlbrecht, T., Dix, J., Schlesinger, F.: From testing agent systems to a scalable simulation platform. In: T. Eiter, H. Strass, M. Truszczynski, S. Woltran (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 (2014)
4. Behrens, T., Dastani, M., Dix, J., Hübner, J., Köster, M., Novák, P., Schlesinger, F.: The Multi-Agent Programming Contest. AI Magazine **33**(4), 111–113 (2012). URL `https://www.aaai.org/ojs/index.php/aimagazine/article/view/2439`
5. Behrens, T., Dastani, M., Dix, J., Köster, M., Novák, P.: The multi-agent programming contest from 2005–2010: From collecting gold to herding cows. Annals of Mathematics and Artificial Intelligence **59**, 277–311 (2010)
6. Behrens, T., Dastani, M., Dix, J., Köster, M., Novák, P. (eds.): Special Issue about Multi-Agent-Contest I, *Annals of Mathematics and Artificial Intelligence*, vol. 59. Springer, Netherlands (2010)
7. Behrens, T., Dastani, M., Dix, J., Novák, P.: Agent contest competition: 4th edition. In: K.V. Hindriks, A. Pokahr, S. Sardiña (eds.) Programming Multi-Agent Systems, 6th International Workshop (ProMAS 2008), *Lecture Notes in Computer Science*, vol. 5442, pp. 211–222. Springer (2009)
8. Behrens, T., Köster, M., Schlesinger, F., Dix, J., Hübner, J.: The Multi-agent Programming Contest 2011: A résumé. In: L. Dennis, O. Boissier, R. Bordini (eds.) Programming Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 7217, pp. 155–172. Springer Berlin / Heidelberg (2012)
9. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. John Wiley & Sons (2007)
10. Dastani, M., Dix, J., Novák, P.: The second contest on multi-agent systems based on computational logic. In: K. Inoue, K. Satoh, F. Toni (eds.) Computational Logic in Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 4371, pp. 266–283. Springer Berlin Heidelberg (2007). DOI 10.1007/978-3-540-69619-3_15. URL `http://dx.doi.org/10.1007/978-3-540-69619-3_15`
11. Dastani, M., Dix, J., Novák, P.: Agent contest competition - 3rd edition. In: M. Dastani, A. Ricci, A. El Fallah Seghrouchni, M. Winikoff (eds.) Proceedings of ProMAS '07, Revised Selected and Invited Papers, no. 4908 in Lecture Notes in Artificial Intelligence. Springer, Honululu, US (2008). URL `http://www.aronde.net/uploads/tx_pubdb/AgentContest-2007.pdf`
12. Hübner, J.F., Sichman, J.S., Boissier, O.: S-moise+: a middleware for developing organised multi-agent systems. In: Proceedings of the 2005 international conference on Agents, Norms and Institutions for Regulated Multi-Agent Systems, pp. 64–77. Springer-Verlag (2005)
13. Karakovskiy, S., Togelius, J.: The Mario AI benchmark and competitions. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 55–67 (2012)
14. Köster, M., Schlesinger, F., Dix, J.: The Multi-Agent Programming Contest 2012. In: Programming Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 7837, pp. 174–195. Springer Berlin Heidelberg (2013). DOI 10.1007/978-3-642-38700-5_11. URL `http://dx.doi.org/10.1007/978-3-642-38700-5_11`

15. Parikh, N., Marathe, M., Swarup, S.: Summarizing simulation results using causally-relevant states. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 88–103. Springer (2016)
16. Szabo, C., Teo, Y.M., Chengleput, G.K.: Understanding complex systems: Using interaction as a measure of emergence. In: Proceedings of the 2014 Winter Simulation Conference, pp. 207–218. IEEE Press (2014)